Universidade de Aveiro Departamento de Engenharia Mecânica
2014

**João José
Oliveira Barros**

Háptica cooperativa para teleoperação de um robô humanóide

Cooperative haptics for humanoid robot teleoperation

Universidade de Aveiro Departamento de Engenharia Mecânica
2014

João José Oliveira Barros

Háptica cooperativa para teleoperação de um robô humanóide

Cooperative haptics for humanoid robot teleoperation

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 Jorge Manuel Mateus Martins** Professor Auxiliar do Instituto Superior Técnico
	Prof. Doutor Vítor Manuel Ferreira dos Santos Professor Associado da Universidade de Aveiro (orientador)

Agradecimentos / Acknowledgements

Em primeiro lugar, quero expressar a minha sincera gratidão ao professor Vítor Santos. A sua entrega ao projecto e ao laboratório, a sua dedicação aos alunos, a sabedoria e o copioso entusiasmo que nos transmite, personificam-no como verdadeiro orientador. A paixão que nutre pela ciência e a sua busca incessante pelo conhecimento, incitaram-me a aprender e descobrir cada vez mais.

De igual forma, quero agradecer ao professor Filipe Silva pela sua sempre disposição para ajudar e compreender, e pela valiosa experiência que concedeu para a elaboração deste trabalho. As suas contribuições são imprescindíveis para o sucesso do Projecto Humanóide.

Não posso também deixar de agradecer aos meus colegas do Laboratório de Automação e Robótica, pela ajuda e sempre oportunas ideias, envoltas em gracejos e boa disposição. Em especial, ao Jorge Almeida pela incansável disponibilidade e paciência.

Aos meus colegas e amigos de curso, ilustres Engenheiros que me acompanharam nesta altiva jornada, o meu muito obrigado! Os ensinamentos que me transmitiram e a vontade de prevalecer, ajudaram-me a alcançar o sucesso em alturas mais esforçadas. Os seus conselhos, a companhia e os momentos de alegria e descontração, de onde sempre se erguem discussões pseudo-científicas, jamais serão esquecidos.

Aos meus pais, quero agradecer a oportunidade que me deram para poder alcançar o sonho de ser estudante universitário. Sem eles, nunca teria sido possível. À minha mãe, por tudo o que fez e tem feito por mim, pelo conforto, pela motivação, pela ajuda e pela paciência que tem comigo, muito obrigado! Ao meu pai, pela referência que foi de dedicação, confiança, empenho e perfeição, e por ter feito de mim a pessoa que sou hoje, é com saudade que lhe dedico este trabalho.

À Patrícia, por simplesmente, ainda que não de forma simples, existir na minha vida.

Palavras-chave

Háptica, interface háptica, teleoperação cooperativa, robótica humanóide, ensino tele-cinestético, aprendizagem robótica

Resumo

O elevado grau de complexidade associado às plataformas robóticas humanóides impõe algumas restrições à definição de modelos rigorosos para o estudo de padrões de movimento e locomoção. Assim, no contexto do Projecto Humanóide da Universidade de Aveiro - PHUA, tem sido desenvolvido algum trabalho de forma a tirar partido do conceito de aprendizagem por demonstração, através da teleoperação humana com recurso a um joystick háptico. Os dados extraídos durante os testes serão depois usados em algoritmos de aprendizagem, tentando com que o robô seja capaz de se mover, equilibrar e andar autonomamente. Contudo, a complexidade das cadeias cinemáticas, como as pernas em tarefas de locomoção e equilíbrio, torna o processo de aprendizagem complexo e exigente pela escassez de graus de liberdade no comando, em comparação com aqueles presentes no modelo. Deste modo, a solução possível passa pelo uso de dois joysticks, um por cada perna, de forma a assegurar uma correspondência entre os graus-de-liberdade do joystick e do sistema real. Por outro lado, é também reconhecível a dificuldade em teleoperar o sistema real, não só por esta complexidade em termos de graus-de-liberdade, mas também pelos erros introduzidos por componentes mecânicos e electrónicos. Assim, antes da operação da plataforma real do PHUA será desenvolvida uma plataforma de simulação usando para isso um software livre, o *V-REP*. Simultaneamente, a interface háptica gerará um *feedback* de força no dispositivo háptico que será sentida pelo utilizador, baseada na resposta dinâmica do sistema a movimentos e equilíbrio, bem como outras perturbações externas.

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

Abstract

The high degree of complexity associated with the humanoid robotic platforms places some restrictions on the definition of accurate models to the study of motion patterns. Thus, in the context of the University of Aveiro Humanoid Project - PHUA, some work has been done on trying to take advantage of the new concept of robot learning from demonstration, where a human user teleoperates the robot in different motion and balance tasks using a haptic joystick. The data collected during the tests can be used in learning algorithms, so the robot will be able to move, balance and walk on its own. However, the complexity of its kinematic chains, as are the legs in motion and balance operations, makes the learning procedure a lot more challenging and complex due to the shortage of degrees-of-freedom available on the joystick, when compared to those available on the robot. The potential solution lies in the use of two joysticks, one for each leg, in an attempt to ensure a degrees-of-freedom matching between the joystick and the platform. On the other hand, it is also easily recognizable how difficult it could be to teleoperate the real system, not only because of its complexity in terms of degrees-of-freedom but also due to possible errors introduced by mechanical and electronic components. Hence, before trying this operation in the real PHUA platform, this approach is developed and tested in a simulation scenario, making use of an open-source robot simulator, the *V-REP*. Concomitantly, the haptic interface will generate a force feedback on the haptic device which will be felt by the user, based on the model dynamic response to motion and balance, besides some other disturbances.

Contents

List of Tables

List of Figures

List of Acronyms

PHUA	Projecto Humanóide da Universidade de Aveiro
DOF	Degrees-of-Freedom
V-REP	Virtual Robot Experimentation Platform
ROS	Robot Operating System
CoP	Center of Pressure
CoM	Center of Mass
ZMP	Zero-Moment Point
CMP	Centroidal Moment Pivot
FRI	Foot Rotation Indicator
3D-LIPM	Three-Dimensional Linear Inverted Pendulum Mode
RMP	Reaction Mass Pendulum

Chapter 1

Introduction

1.1 Context and Motivation

Research on humanoid robots and biped locomotion is currently one of the most exciting topics in the fields of robotics, with many ongoing projects. Although a lot of work has already been done in the past several decades, developing a full potential humanoid platform is still a long-term goal. Balance control of humanoid robots in the presence of unforeseen external or internal forces has remained an unsolved problem. The difficulty of this problem is a consequence of the high dimensionality of the action space of a humanoid robot, due to its large number of degrees of freedom (joints), and non-linearities of its kinematic chains. The approaches that have been proposed to solve some of these problems can be classified into two categories. The first group uses knowledge of dynamic parameters of a robot e.g. mass, location of center of mass and inertia of each link to prepare balance control algorithms and walking patterns. Therefore, it mainly relies on the accuracy of the model data. Contrary, the second group uses limited knowledge of dynamics. Since the controller knows little about the system structure, this approach much relies on a feedback control.

Robot learning by demonstration is a powerful tool 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. The demonstrations are typically provided either by teleoperating the robot, or by vision and motion sensors that are able to monitor the robot state and environment conditions.

The main focus of this work is centered on the University of Aveiro Humanoid Project (PHUA), that currently consists of 27 degrees-of-freedom (DOF) whole body small-sized humanoid platform. In the context of this project, some work has been done on trying to take advantage of the concept of robot learning from demonstration, where a human teleoperates the robot in different motion and balance tasks using a haptic joystick. The data collected during the tests can be used in learning algorithms, so the robot will know how to perform such balance and motion operations without the operator's control. This methodology enables a natural interaction with the robot for tele-kinesthetic teaching [1] and, at the same time, a force feedback on the user that are able to provide functional guidance and corrections, while being aware of robot's dynamics and constraints. Thus, the user can feel the robot's stability and deduce the control behavior to keep it balanced.

1.2 Objectives

The complexity of PHUA humanoid kinematic chains, as are the legs in motion and balance operations, makes the learning procedure a lot more challenging and complex due to the shortage of degrees of freedom available on the joystick, when compared to those available on the model. The potential solution lies, and is the main objective of the present work, in the implementation of two joysticks, one for each leg, in an attempt to ensure a DOF matching between the joystick and the platform. However, it is easily recognizable how difficult it could be to teleoperate the real system, not only because of its complexity in terms of DOF but also due to possible errors introduced by mechanical and electronic components. Thus, before trying this operation in the real PHUA platform, the approach is tested in a simulation scenario, making use of an open-source robot simulator, the Virtual Robot Experimentation Platform (V-REP). Concomitantly, the haptic interface will generate a force feedback in the haptic device, which will be felt by the user, based on the simulator dynamic response to motion and balance besides some other disturbances. The main objectives are presented below:

- Creation of a humanoid model in V-REP and definition of its kinematic chains based on the PHUA robotic platform, and definition of the force feedback developed towards the user.
- Setting up the communication between the two haptic devices.
- Basic teleoperation of the V-REP model in real-time simulation, basic movements with one joystick.
- Teleoperation of the V-REP model in more complex tasks, such as hard balance tasks, with two joysticks
- Test, evaluation and recording of motion patterns parameters by haptic teleoperation;

1.3 State of the Art

1.3.1 Principles of Haptic Interaction

Haptic interaction with the world refers to sensing and manipulation using our sense of touch. Since the early part of the 20th century, the term *haptics* has been used by psychologists for studies on the active touch of real objects by humans. In the 1970s and 1980s, significant research efforts in robotics field also began to focus on manipulation and perception by touch, aspiring the creation of devices with dexterity inspired by human abilities.

Haptics also includes kinesthesia (or proprioception), the ability to perceive one's body position, movement and height. It has thus become common to speak of the "haptic channel" to collectively designate the sensory and motor components of haptics. This is because in certain anatomical parts, in particular the hands, perceiving the world and acting upon it are activities that take place together. Basically, the haptic channel contains tactile and kinesthetic information that are combined to provide humans with means to perceive and act on their environment [2].

In the early 1990s a new usage of the word *haptics* began to emerge. The confluence of several emerging technologies made *virtualized haptics*, or *computer haptics*, possible. Haptic technology interfaces the user with a virtual environment via the sense of touch by applying forces, vibrations and/or motions to the user. This mechanical simulation may be used to assist in the creation of virtual objects (objects existing only in a computer simulation), that can be physically palpated and controlled, and to enhance remote control of machines and devices.

This new sensory display modality presents information by exerting controlled forces on the human hand through a haptic interface, rather than, as in *computer graphics*, via light from a visual display device. Unlike *computer graphics*, where audio and visual channels feature unidirectional information and energy flow (from the system to the user), haptic interaction is bidirectional, with energy and information flowing from and toward the user [3]. It generates mechanical signals that stimulate both human kinesthetic and touch channels. This is one of the most important features of the haptic interaction modality.

The subsystems and information flow underlying interactions between human users and force-reflecting haptic interfaces are shown in Figure 1.1. Haptic interaction occurs between two dynamical systems, the haptic interface with a computer and the human user with a central nervous system [4].

- *Human sensorimotor loop*: when a human user touches a real or virtual object, forces are imposed on the skin. The associated sensory information is conveyed to the brain and leads to perception. After interpreting the environment, the brain motor commands activate the muscles and result in hand and arm motion.

- *Machine sensorimotor loop*: when the human user manipulates the end-effector of the haptic interface device, the position sensors on the device convey its tip position to the computer. The models of objects in the computer calculate in real-time the torque commands to the actuators on the haptic interface, so that appropriate reaction forces are applied on the user, leading to tactual perception of virtual objects.

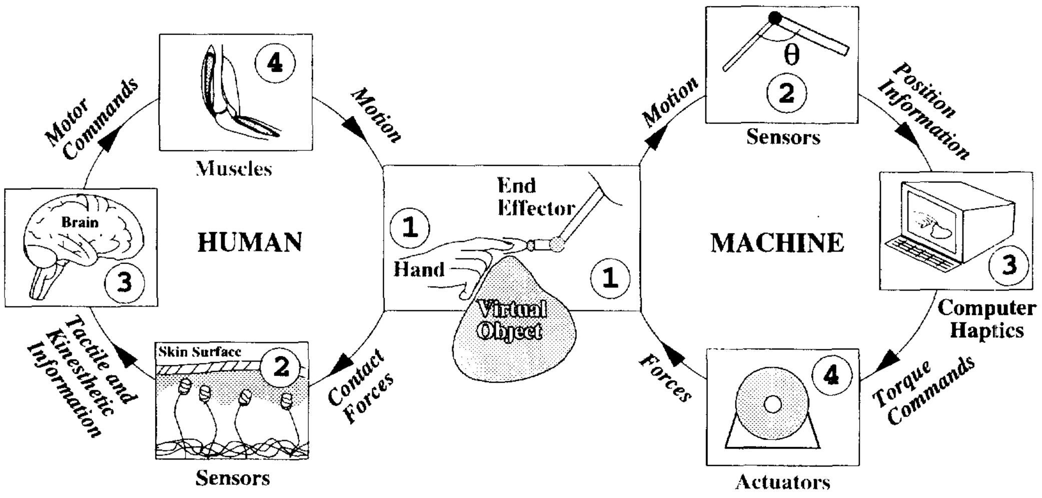


Figure 1.1: Haptic interaction between humans and machines [4].

Virtual reality is the technology which allows the user to interact with computer-simulated objects, or virtual objects, in real time, and under real or imaginary conditions. Figure 1.2 shows the basic structure of a virtual reality application incorporating haptic feedback in addition to most current visual and auditory experiences. This technology is often used to describe a wide variety of applications. The development of CAD software and graphics software acceleration helped the growth of powerful 3-D simulators. Medical, gamming and graphical art applications are very common.

Application's main elements include:

- *simulation engine*, responsible for computing the virtual environment's behavior over time.

- *visual*, *auditory*, and *haptic rendering algorithms*, which compute the virtual environment's graphic, sound, and force responses toward the user.

- *transducers*, which convert visual, audio, and force signals from the computer into a form the operator can perceive.

- *human operator*, who typically interacts (bidirectional) with the haptic interface device and perceives audio and visual feedback, from computer, headphones, visual displays, etc.

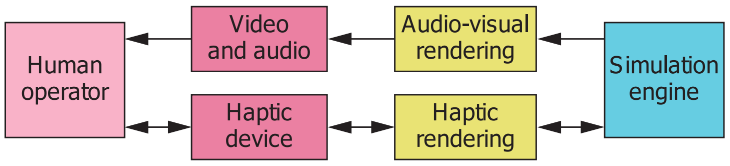


Figure 1.2: Basic architecture for virtual reality application incorporating visual, auditory, and haptic feedback [3].

Being able to touch and manipulate objects in an environment, in addition to seeing (or hearing) them, provides a sense of immersion in the environment to user, otherwise not possible. A greater immersion in a virtual reality environment can be achieved by the synchronous operation of a haptic interface with visual and auditory display, than by large improvements in the fidelity of the visual display alone.

This force feedback is synthesized on the user through computer algorithms, in a process known as haptic rendering. Haptic-rendering algorithms compute the correct interaction forces between the haptic interface representation inside the virtual environment and the objects populating the environment [3]. Moreover, haptic-rendering algorithms ensure that the haptic device correctly renders such forces on the human operator. The main components of a typical haptic-rendering algorithm are illustrated in Figure 1.3.

Assuming the interaction tool between the haptic system and the human user is a force-reflecting joystick, the operator controls the avatar moving the joystick's end-effector. Low-level control algorithms sample the position sensors at the haptic interface device joints. Avatar's position inside the virtual environment is the position of the device-body interface in Cartesian space, obtained through the sensors information and forward kinematics algorithms. An *avatar* is the virtual representation of the haptic interface, through which the user physically interacts with the virtual scene.

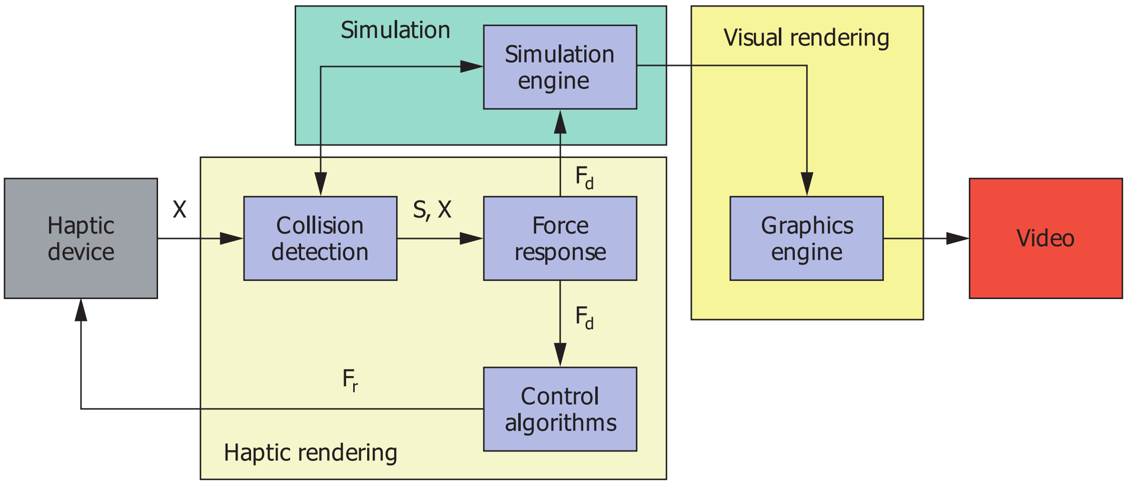


Figure 1.3: Haptic rendering architecture split into three main blocks. *Collision-detection* algorithms provide information about contacts S occuring between an avatar at position X and objects in the virtual environment. *Force-response* algorithms return the ideal interaction force F_d between avatar and virtual objects, while *Control* algorithms return a force F_r to the user [3].

Collision-detection algorithms detect collisions between objects and avatars in the virtual environment and process information about the nature of such collisions. *Force-response* algorithms compute the interaction force between avatar and virtual objects involved in a collision. This force approximates as closely as possible the contact forces that would normally exist during contact with real objects. Desired force and torque vectors computed by force-response algorithms feed the control algorithms that return the actual force and torque vectors. These are then converted into device joint's torque through inverse dynamics algorithms.

The simulation engine then uses the same interaction forces to compute their effect on objects in the virtual environment. Although there are no firm rules about how frequently the algorithms must repeat these computations, 1 kHz servo rate is common. This rate allows presentation of reasonably complex objects with reasonable stiffness. Higher servo rates can provide crisper contact textures sensations, 1 kHz seems to be a subjectively acceptable frequency value for the haptic rendering to be seamless and natural.

Haptics can be found in a wide variety of applications. Surgical simulation and medical training, military training in virtual environment, ground vehicle simulators, games, arts and creation, touchscreens, computer aided design/engineering/manufacturing (CAD/CAE/CAM), and of course, telerobotics, are just some examples of haptic interface applications.

Various haptic interfaces for medical simulations may prove especially useful for training of minimally invasive procedures and remote surgery using teleoperators. Ultimately, it can avoid the need for real-time presence of surgeons, and allow performing many operations of a similar type, with less fatigue and more accurately. In reality-based modeling for surgical simulation, the surgeon interacts with the haptic interface to control the surgical robot and instrument that will operates on the patient [5].

Today, the military uses virtual reality techniques not only for training and safety enhancement, but also to analyze military maneuvers and operate vehicles. Flight simulators are very common and useful to simulate a lot of different scenarios. As in ground vehicle simulators, the operator commands the airplane with a joystick controller attached to a console. These are very close to game applications, where force feedback is applied to the player, reproducing placing, hitting or bouncing states [5]. Some of the dynamics phenomena reproduced in virtual reality applications are way beyond of the simple rumble or vibration created in video games or touchscreens of smartphones.

In a telerobotic system, a human operator controls the movements of a robot that is located away from him. Such a system can be very hard to control, so that a teleoperated robot is generally limited to very simple tasks. Haptics provides new insight into teleoperation robotics, including touch cues in addition to audio and visual cues in telepresence models. The ability to perceive robot states by feeling them, will naturally improve the quality of the operation.

When interacting with virtual objects through haptic devices, most of the time only one hand is involved. However, this project aims to implement a teleoperation system including two haptic devices, through which the user would be capable of controlling the robot, first in a virtual environment, and later in reality. The field which encompasses all studies of the haptic interaction with either remote or virtual environments using both hands of the same person is referred to as bimanual haptics [6] The different steps that allow a user to perform a bimanual task in virtual, or remote, environment are represented in Figure 1.4.

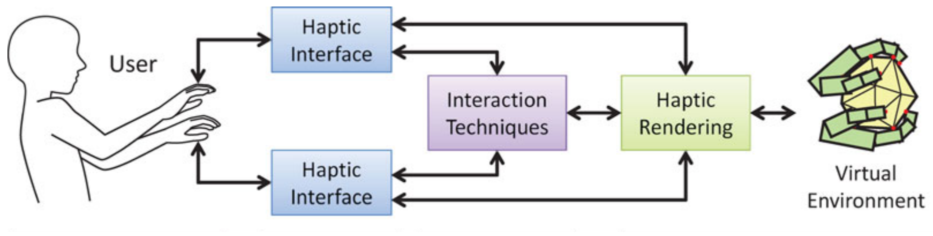


Figure 1.4: Different steps allowing a user to perform a bimanual haptic task in a virtual or remote environment. The user (human layer) interacts with the haptic interfaces (hardware layer), which are coupled to the virtual environment through haptic rendering (software layer). Interaction techniques are linked to all of these elements and further allow the user to perform a given task [6].

The domain of two-handed haptics is starting to emerge, not only because of the increase of computational power and the decrease of costs of haptic devices, but mainly due to the current studies focusing the benefits of the dual interaction. The use of two hands acting in a integrated way allows to perform tasks that could hardly be done with a single hand. Besides the larger number of tasks that can be executed faster, and simultaneously, bimanual actions involve a major form of haptic feedback: ability to locate our hands relatively to each other [6]. This greatly increases the user's capabilities when compared to a single hand interaction, providing a very important tool, particularly in the absence of visual feedback. However, when the tasks applied by the two hands are not sufficiently integrated, the operator may have to divide his attention, and the the performance may decrease significantly.

Interaction techniques combine software and hardware solutions to allow a user to perform a specific task, including navigation, selection, manipulation, and system control tasks. Although the amount of two-handed haptic interaction techniques remains limited, the concept starts to emerge, and numerous applications can be though for medicine and industry that take advantage of the remarkable sense of cooperation of the human hands. Figure 1.5 shows an example of a haptic interaction through both hands of the same person.

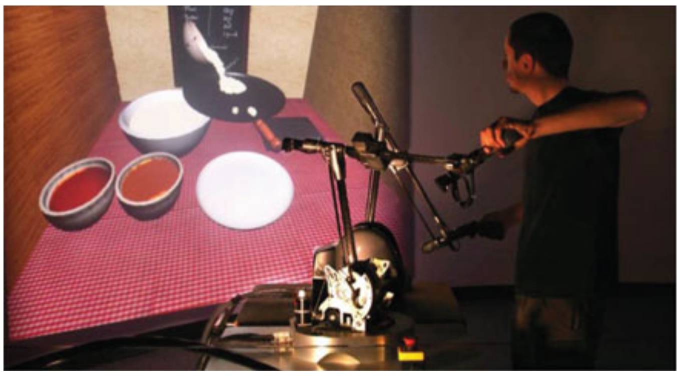


Figure 1.5: An example of bimanual haptics: preparing virtual crepes with a virtual bow and pan, with force feedback on both hands [6].

1.3.2 Teleoperation, Kinesthetic Teaching and Learning from Demonstration

Teleoperation of mobile robots, including humanoid robots, benefits from the incorporation of force feedback in addition to remote images and navigation sensor data provided by usual applications. Additional force feedback, rendered through haptic interfaces, provides inputs to the teleoperator in order to improve the remote control performance. The study of modern telepresence systems analyzes the integration of multimodal data input, the compensation of time delay effects, and most important, the synchronization of teleoperations/autonomous control reactions. Figure 1.6 shows a small mobile robot teleoperated using a haptic interface. A broad variety of sensors is used for navigation and for characterization of the working environment. Between them, is a force sensor located at the front of the car, which measures the force when the car pushes against an obstacle. Based on this measurement, the teleoperator's haptic joystick, which also commands the vehicle, generates proportional forces [7].

The joystick is the main input device for the teleoperator to commands the robot. It is connected by a PCI interface board to client's computer, which enables a high data rate between the program (here a workstation with Linux operating system) and the joystick. This is an important feature when referring to haptic feedback generation, as already mentioned. In order to provide a fast an easy communication, an interface in object-oriented programming language C++ is implemented. Moreover, the communication between the client and server computers is based on sockets, using the Internet user datagram protocol (UDP). Unlike TCP/IP Protocol, used as Internet standard, no

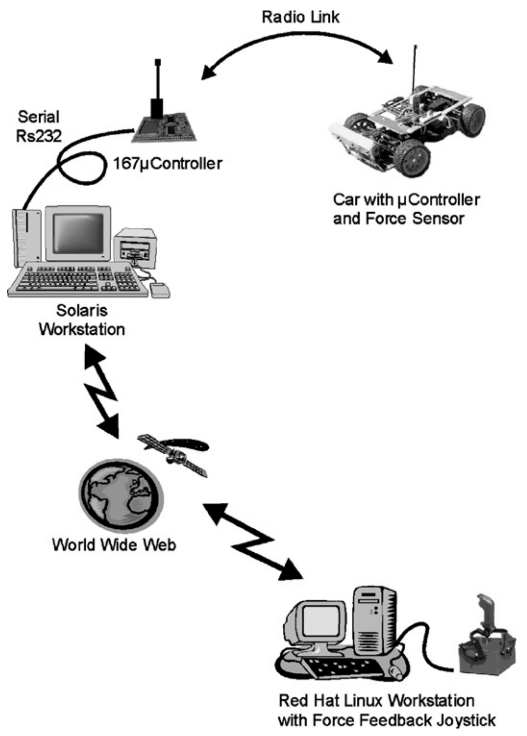


Figure 1.6: The telematic system [7].

acknowledgment of received data packages is provided. Thus, in case of lost packages no retransmission is required. The data transfer reaches with UDP better real-time performance, by continuously send and receive the latest information possible. It avoids delays associated with the attempt of recovering lost information, what greatly improves the synchronism of the telematic system.

Haptics can be also used for skills training, which is called *haptic guidance*. In the haptic guidance paradigm, the subject is physically guided through the ideal motion by the haptic interface, thus giving the subject a kinesthetic understanding of what is required [8]. Subject learns some 3D motions while having haptic and visual feedback, and then tries to manually reproduce those movements under two recall conditions, with vision and without vision. Kinesthesis is crucial in haptic training, because it is one of the most powerful ways of perceiving incoming stimuli. Furthermore, kinesthetic memory, or the ability to remember limb position and velocity has proved to be very important in haptic training. Humans have a remarkable ability to remember positions of their limbs quite accurately and for long periods [9].

In contrast, the focus of learning from demonstration is to develop an intuitive and interactive method based on user demonstrations to create human-like and autonomous behavior for a robot. By capturing the sophisticated operation of a human expert using high degree-of-freedom haptic interfaces, a user's skill can be stored, analyzed, and transferred to robot or even other humans. At first, the human teleoperates the robot to perform a certain task. After the robot executes the commands from the human operator, learning algorithms, such as artificial neural networks (ANN) and support vector machines (SVM), are used to train learning modules over the temporal sequences of the trajectory. Then, the learned modules are used to control the robotic system which will try to autonomously perform the trained task [10]. Skill transfer mechanisms have become very common in several fields of robotics. For instance, in the previously referred paper, the experiment consists of a reality-based haptic interaction system for knowledge transfer by linking an expert's skill with robotic movement in real time. The goal was to successfully recreate the operator's handwriting, and then transfer haptic forces to an untrained user, through a haptic device. Figure 1.7 shows and example of letter "b" demonstrated twice for the robot.

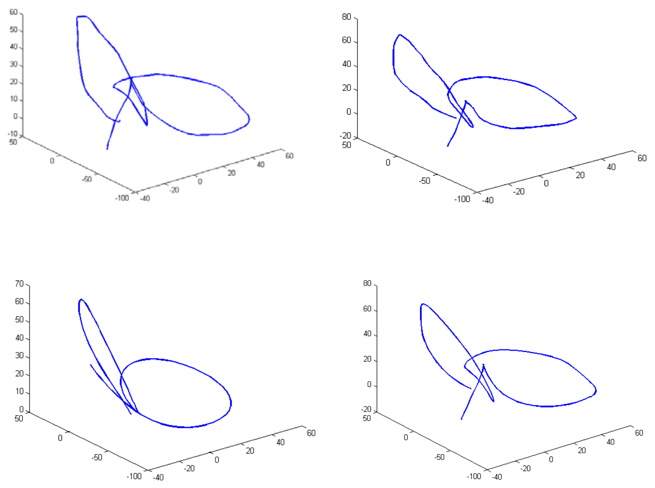


Figure 1.7: Letter "b": human writing patterns (top) and robot's writing patterns (bottom). Bottom left: result with NN learner, bottom right: result with SVR learner [10].

In another experiment, illustrated in Figure 1.8, a user teleoperates a humanoid robot to demonstrate how to perform a lifting task, collaboratively with another user [11]. The robot can learn through observation to perform a collaborative manipulation task, first demonstrated by a user who controls the robot's hand via a haptic device (Phantom device in figure). The proposed learning approach can be used to ensure the robot's retrieval of the task, reproducing the overall dynamics of the task, namely the force patterns for both lift the object and adapt to the human user's hand motion. This shows the potential of teleoperation to transmit both dynamic and communicative information.

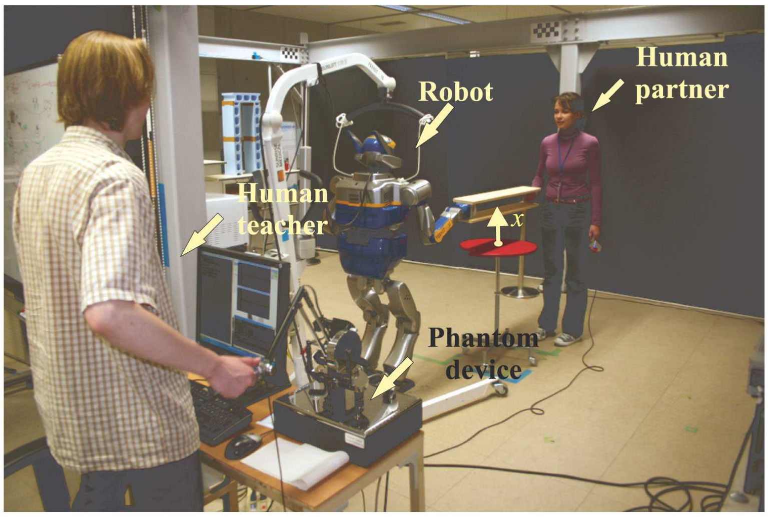


Figure 1.8: A user (on the left) teleoperates a humanoid robot to demonstrate how to perform a lifting task collaboratively with another user (on the right) [11].

Controlling a full-body humanoid is an extremely difficult task, especially if the robot is standing free on its legs. Physical human-robot interaction with humanoid has been studied in the context of assisted balancing or walking. The development of full potential humanoid platforms depends largely on their ability to learn new tasks by themselves or to imitate human demonstrations of tasks. One experiment leads a free-standing humanoid robot to acquire new motor skills by kinesthetic teaching. Imitation learning is used for training the upper body of the humanoid via kinesthetic teaching, while at the same time Reaction Null Space method is used for keeping the robot balanced [12]. During demonstration, as shown in Figure 1.9(a) and (b), along with the kinematic information, a force/torque sensor is used to record exerted forces. In reproduction phase (Figure 1.9(c)), the acquired force values are used to build a controller that will apply the learned trajectories in terms of positions and force to the end-effector.

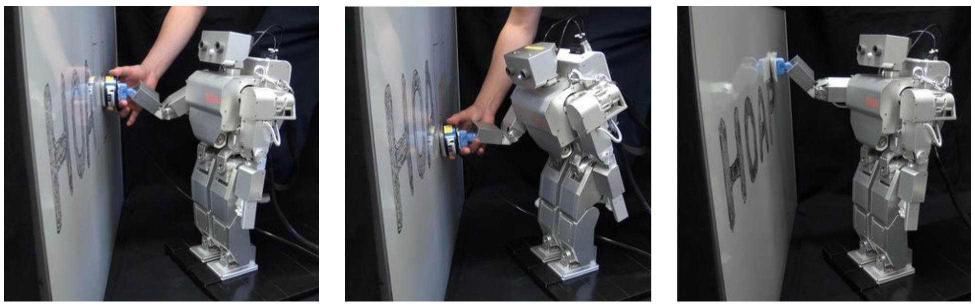


Figure 1.9: Upper-body kinesthetic teaching of a free-standing robot [12].

1.3.3 University of Aveiro Humanoid Project (PHUA)

Started in 2003/2004, the PHUA was the outcome of a joint effort between the Mechanical Engineering Department (DEM) and the Electronics, Telecommunications and Informatics Department (DETI), of the University of Aveiro, to create a humanoid platform suitable for research in control, perception, and behavior. Its main objective is the development and integration of hardware and software components in a functional low-budget platform, to perform studies in balance and locomotion tasks. The robot's current form is depicted in Figure 1.10.

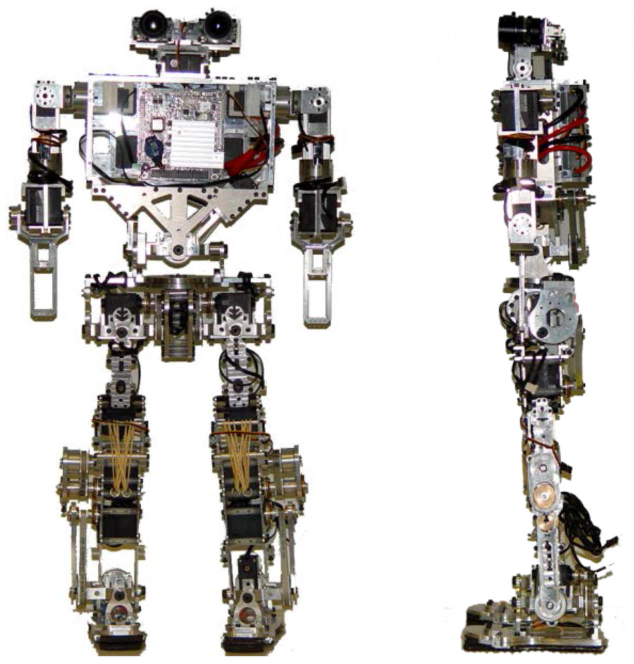


Figure 1.10: Front and side views of the PHUA platform's full body [13].

The first model was finished in 2008, with several thesis and published papers during the 4 years of development. This platform was capable of identifying and following objects with its head and performing several exercises with its legs. In 2009, the project made a major step forward with the design and development of a new small-size humanoid platform based on hybrid actuation, as shown in the above figure. The mechanical structure was re-designed and new actuators and sensors were chosen.

This robot is whole-body humanoid platform with a total of 25 active and 2 passive degrees-of-freedom, as shown in Figure 1.11. Table 1.1 complements the visual information. The joint actuation is composed of *HITEC®* analog and digital servomotors, combined with elastic elements. The robot uses a belt and pulley adjustable transmission system on legs and torso, providing torque to the joints with higher requirements. The platform is also equipped with a set of 8 force sensors, 4 on each foot, from which valuable data for balance demonstrations can be collected.

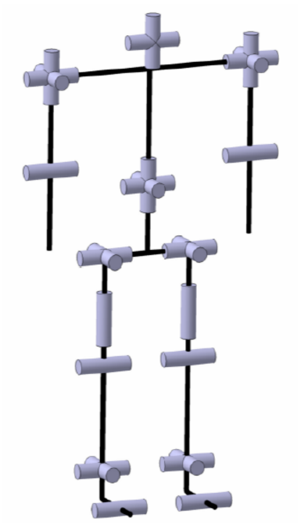


Figure 1.11: PHUA kinematic chains [14].

Table 1.1: PHUA platform's DOF

Joint	**Number of DOFs**
Toe	1(×2)
Ankle	2(×2)
Knee	1(×2)
Hip	3(×2)
Trunk	3(×1)
Shoulder	3(×2)
Elbow	1(×2)
Neck	2(×1)
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. 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. These value are later used in the model's construction. Figure 1.12 illustrates the mentioned angles for the lower limbs.

The last couple of works developed in this project introduced the concept of tele-kinesthetic teaching by haptic teleoperation, joining haptics and learning from demonstrations algorithms [14, 15]. The data recorded through the haptic interface, during the demonstrations, can later be used by robot learning from demonstration algorithms, thus disposing of complex dynamic models of the robot. By controlling the platform via a haptic interface, the operator has the ability to "feel" the robot's state, and react to it, adapting his own operation to changes in the environment.

Currently, control methodologies and data gathering alternatives are being tested for later use in learning procedures. This work follows the first steps toward the long-term goal of autonomous balancing and walking.

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

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

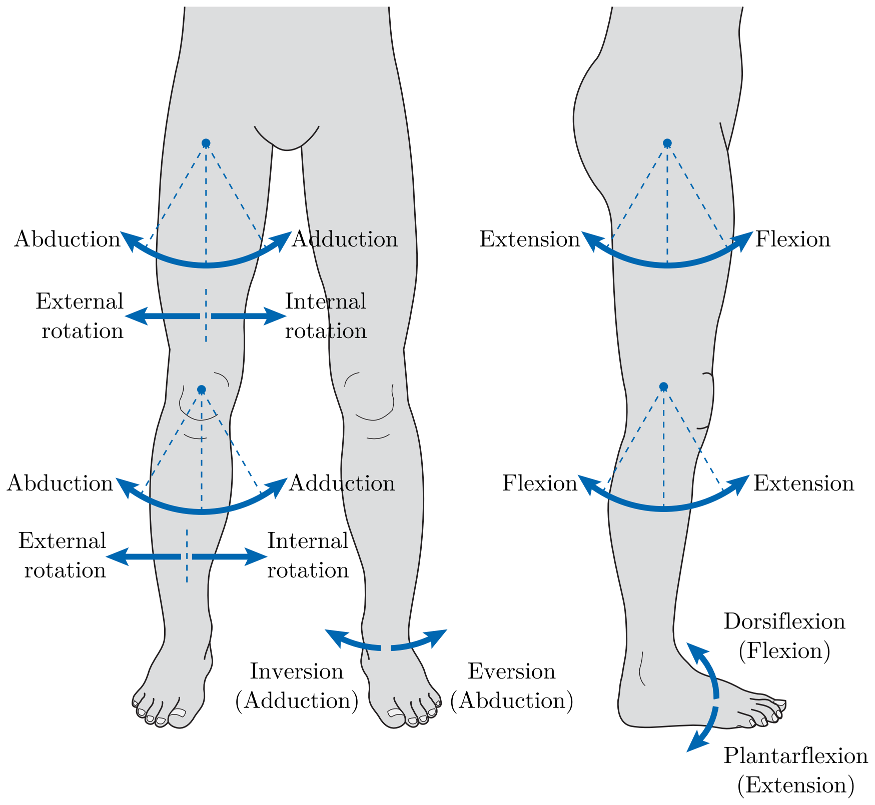


Figure 1.12: Movements about the hip, the knee, and the ankle joints (adapted [16]).

1.4 Thesis Guide

The stages of this work are presented in the following chapters, organized as follows:

- Chapter 1 introduced the PHUA project and briefly explained the context and the objectives of the current work. A state of the art review on haptic interaction, teleoperation, and learning from demonstration methods was also exposed;
- Chapter 2 is completely devoted to the V-REP simulator, and the PHUA model construction. After the presentation of the simulator's main features and its capabilities, a detailed description of the procedure to create the simulated model is carried out, referring to existing CAD model import, model's optimization for dynamic simulations, kinematic chains, etc;
- Chapter 3 presents the haptic device used in this project and its dedicated software libraries, and introduces a strategy to implement the dual teleoperation based on a ROS framework. The software applications developed to interact the joysticks with the simulator are fully described.
- Chapter 4 explains some of the existing models and methodologies explored to define balance, and describe the dynamics in legged robots. The study of these models and their comparison with the PHUA approach show the relevance of applying the project's methodologies in humanoid robots control. Strategies to teleoperate the robotic model with single and dual joystick configurations are presented, as well as the force feedback formulation implemented.
- Chapter 5 presents the experiments undertaken to test and validate the methodologies described in this work. Some of the data gathered during the demonstrations are presented and discussed.
- Chapter 6 discusses the main conclusions of this dissertation work, and presents some proposals for future works.

Chapter 2

Virtual Robot Experimentation Platform, V-REP

The V-REP simulator and the PHUA robot full body model are presented in this chapter. An extended overview of the simulator is given in order to familiarize the reader with the functionalities and all the important concepts and notation adopted by V-REP, and referred in the construction of the robotic model. This model is built according to the actual robot mechanical and dimensional specifications, and designed to simulate the real robot's behavior, as close as possible. An *embedded script*, associated with the model itself, allows for communication between the simulator and external applications, enabling simulation's control and data retrieving, simultaneously. The interface with the external environment is designed to largely exploit the ROS functionality of V-REP, which enables the communication with other ROS modules running on the network. Keeping the ROS-based communication, the model's portability to different simulation scenarios is perfectly ensured.

2.1 Overview and Main Features

The V-REP, Virtual Robot Experimentation Platform, is a relatively new robot simulator with integrated development environment, containing incredible features, lots of functions and elaborate APIs. The robot simulator V-REP is based on a distributed control architecture, which means each object/model can be individually controlled via an embedded script, a plugin, as ROS node, a remote API client, or a custom solution. These programming approaches are also mutually compatible and can even work hand-in-hand, what makes the software very versatile and ideal for multi-robot and multi-function applications. Controllers can be written in C/C++, Python, Java, Lua, Matlab, Octave or Urbi [17].

Thus, V-REP can be used as a stand-alone application or easily combined with other applications. An integrated script interpreter and elaborate API allows the user to customize almost every aspects of a simulation, enabling fast algorithm development, remote monitoring and hardware control, among others. The scripting language is Lua, which is an extension programming language designed to support general procedural programming.

The software basic structure is composed of three central elements: *control mechanisms*, *scene objects* and *calculation modules*.

2.1.1 Simulation Controllers

Building complex simulation scenarios involves, almost certainly, a distributed control framework. It simplifies the task by partitioning control entities, speeds-up simulation by distributing the CPU load over several cores or several machines and it allows a simulation model be controlled by native code execution [18].

The execution of the control code of a simulation or a simulation model is usually handled using three techniques [18]: control code is executed on another machine, on the same machine, but in another process (or another thread) than the simulation loop, or on the same machine and in the same thread as the simulation loop:

- Executing the control code on another machine means a distinct machine or a robot is connected to the simulator machine via a specific network (e.g. socket, serial port, etc). The main advantages of the approach are the originality of the controller (the control code can be native and running on the original hardware) and the reduced computing load on the simulation machine. However, this imposes serious limitations on the synchronization with the simulation loop and the communication delay/lag dictated by the network. This is even more important when low-level controllers, such as real-time motion level controllers, require synchronization with the simulation loop (i.e. executed at the same moment at each simulation pass).

- If the control code is executed on the same machine but in another process, there is the benefit of a balanced load on the CPU cores, but it also comes with a lack of synchronization with the simulation loop.

- The main advantages of executing the control code on the same machine and in the same thread as the simulation loop are the inherent synchronization with the simulation loop and the fact that any communication or thread switching lag or delay are virtually inexistent. On the other hand, there is an increasing load on the simulation loop CPU core.

The latter two control modes are often implemented via external executables or plug-ins loaded by the simulator.

As already mentioned, V-REP allows the user to choose among various programming techniques and languages [17, 18]. A brief explanation of the control possibilities provided by V-REP is presented next, as a complement to the Figure 2.1.

Embedded scripts represent the most powerful and distinctive feature of V-REP. Two major type of embedded Lua scripts are supported: one *main script* and an unlimited number of *child* scripts. By default, each scene has a main script that controls the simulation loop, handling general functionality (inverse kinematics, collision detection, dynamics, etc.). The regular part of this script will be executed at each simulation pass. The main script is also responsible for calling child scripts in a cascaded way (with respect to the scene hierarchy). Although the main script can be customized, it is preferable not to do it, leaving all the customization work to child scripts, and thus avoiding any risk of scenes not operating properly.

Unlike the main script, a child script is attached to a specific object in the simulation scene, and handles a particular part of the simulation. While the main scripts cannot be threaded, child scripts can be executed in a thread and non-thread fashion. When a non-threaded child script is called by the main script, at each simulation step, they perform some operation and then return control to the calling script, which continues the calculation processes. The most common use is to have a non-threaded child script defining and controlling a model.

Child scripts play a very important role in simulation, they can load/unload plug-ins, register ROS publishers/subscribers, open and handle communication lines by socket or serial port, launch executables or start remote API server services. A powerful feature which is implemented in the designed model.

A **plugin** is a shared library, automatically loaded by V-REP's main client application (V-REP executable) at program start-up. Plugins are used in V-REP as a customization tool for a particular simulation and/or the simulator itself. They can register custom Lua commands, allowing the execution of callback functions from within an embedded script and also extend the functionality of a model or object. Often, plugins are also used to provide an interface to a hardware device (e.g. real robot). The *remote API* interface and the ROS interface are implemented via plugins.

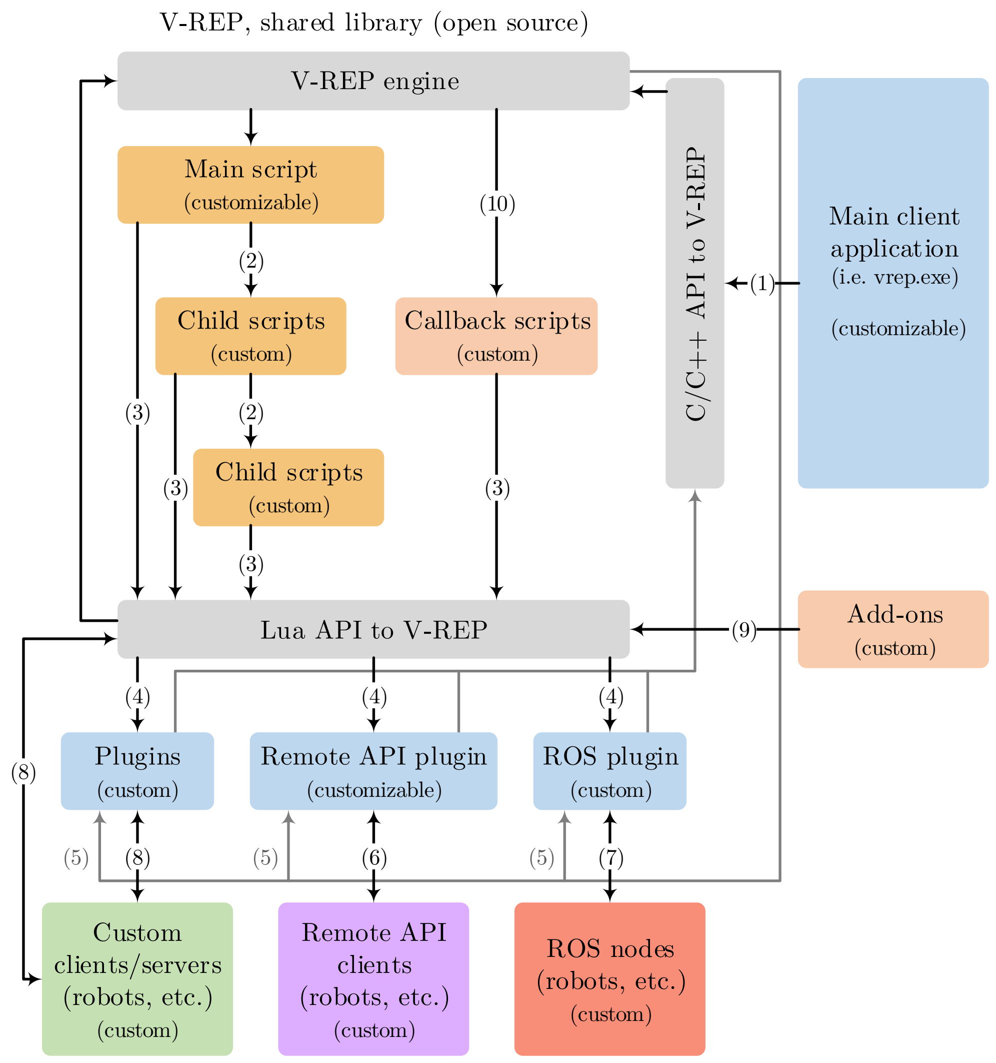

V-REP, shared library (open source)
V-REP engine
Main script (customizable)
Child scripts (custom)
Child scripts (custom)
Callback scripts (custom)
C/C++ API to V-REP
Main client application (i.e. vrep.exe) (customizable)
Lua API to V-REP
Add-ons (custom)
Plugins (custom)
Remote API plugin (customizable)
ROS plugin (custom)
Custom clients/servers (robots, etc.) (custom)
Remote API clients (robots, etc.) (custom)
ROS nodes (robots, etc.) (custom)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)

Figure 2.1: V-REP control architecture (adapted [18]).

- (1) - C/C++ API calls
  (2) - cascaded child script execution
  (3) - Lua API calls
  (4) - custom Lua API callbacks
  (5) - V-REP event callbacks
  (6) - remote API function calls
  (7) - ROS transit
  (8) - custom communication (socket, serial, pipes, etc.)
  (9) - Add-on calls to Lua API
  (10) - Script callback calls

An **add-on** in V-REP is quite similar to a plugin. It is automatically loaded at program start-up and allows the extension of some functionalities, using user-defined functions.

The **remote API** interface in V-REP allows the interaction with a simulation, or the simulator itself, from an external entity via socket communication (e.g. an external application, a remote computer, a real robot, etc). It is composed by the *remote API server*, V-REP, and *remote API client(s)*, user's application(s). The remote API on the server side is implemented via a V-REP plugin, loaded by default at program's start. The client side can be developed in many different programming languages (C/C++, Python, Java, Matlab and Urbi), and allows remote function calling and fast data streaming back and forth.

The **ROS** functionality in V-REP is enabled via a plugin that can be easily adapted to user's own needs. This allows ROS to call V-REP commands via ROS services and/or stream data via ROS publishers and subscribers. The V-REP ROS interface is the key element of the present project, around which is based the entire work. A very detailed description of ROS framework will be presented in the following chapters.

2.1.2 Scene Objects

A V-REP simulation scene contains several scene objects that are attached or assembled to each other in tree-like hierarchy to build a simulation model [17, 18]. Supported scene objects and a brief function description are presented next:

- **Shapes**: *shapes* are rigid meshes composed of triangular faces, used for body simulation and visualization.
- **Joints**: *joints* are elements that link two or more scene objects together and can also act as actuators. Four types are supported: revolute joints, prismatic joints, screws and spherical joints.
- **Graphs**: *graphs* can record and visualize a large variety of predefined or custom simulation data. Data streams can be displayed directly in a time graph, or combined with each other to display X/Y graphs or 3D curves.
- **Dummies**: a *dummy* is a reference frame, a point with orientation. Dummies are multipurpose objects that can have many different applications.
- **Proximity sensors**: a *proximity sensor* detects objects in a geometrically exact fashion within its detection volume, performing an exact minimum distance calculation.
- **Vision sensors**: *vision sensors* are camera-type sensors, reacting to light, colors and images of a simulation scene. A built-in filtering and image processing function enables the composition of blocks of filter elements. Vision sensors make use of hardware acceleration for the raw image acquisition (OpenGL).
- **Force sensors**: *force sensors* represent rigid links between shapes, that are able to measure forces and torques applied to them and can conditionally break apart when a given threshold is overshoot.
- **Mills**: a *mill* is a customizable convex volume that can be used to simulate surface cutting operations on *shapes.*
- **Cameras**: *cameras* allow scene visualization from various view points.
- **Lights**: *lights* illuminate a scene or individual scene objects and directly influence *cameras* and *vision sensors.*
- **Paths**: a *path* is an object that defines a path or a trajectory in space, allowing complex movement definitions is space (succession of freely combinable translations, rotations and/or pause). They can be used for guiding a welding robot's torch along a predefined trajectory, for example.
- **Mirrors**: *mirrors* can reflect images/light, but also operate as an auxiliary clipping plane.

Some of the described objects have special properties that allow their interaction with other scene objects or calculation modules: *collidable* objects can be tested for collision against other *collidable* objects; *measurable* objects can have the minimum distance between them and other *measurable* objects calculated; *detectable* objects can be detected by proximity sensors; *cuttable* objects can be cut by mills; *renderable* objects can be *seen* or *detected* by vision sensors; *viewable* objects can be *looked through*, looked at, or their image content can be visualized in views.

Each object has a configured position and orientation in the simulation scene. Objects can be attached to other objects (or built on top of each other), and thus have a parent-child relationship. If an object A is built on top of object B, then object B is the parent and object A is the child. Although the objects' configuration remains unchanged, if object B moves, object A will automatically follow, since object A is attached to B. This daisy chain connection provides a very practical tool to build simulated models when robotic arms, or other serial manipulator, are intended to be simulated. This methodology was of great use during the PHUA model construction.

Every object has an absolute configuration (or cumulative configuration) that is relative to the world's reference frame, and a local configuration (or relative configuration) that is relative to the parent's object reference frame. In the present example, if both objects keep their position, object A's absolute configuration will not change, but its local configuration will be modified.

2.1.3 Calculation Modules

Scene objects are rarely used on their own, they rather operate in conjunction with other scene objects. In addition, V-REP offers powerful calculation functionalities, or calculation modules, can directly operate on one or several objects [17, 18]. Calculation modules include:

- *collision detection* module.
- *minimum distance calculation* module.
- *inverse kinematics calculation* module.
- *geometric constraint solver* module.
- *dynamics* module.
- *path planning* module.
- *motion planning* module.

With the exception of *dynamics* module, all other calculation modules allow registering *calculation objects* that are user defined. These object are different from scene objects, but are indirectly linked to them by operating on them. Calculation objects can be defined as the interaction, with respect to a calculation module, that occurs between scene objects. For example, if *collision object* A is defined by a *collidable object* B and a *collidable object* C, it represents the collision detection performed between object B and object C [17].

The **collision detection** module allows fast interference checking between any collidable shape or collection of shapes. This module will only detect collisions, it will not react to them. It is fully independent from the collision response calculation algorithm of the dynamics module.

V-REP can measure the **minimum distance** between two measurable entities in a very flexible way. The mesh-mesh distance calculation module allows fast calculations for any shape (convex, concave, open, closed, etc.) or collection of shapes. The module uses the same data structure as the collision detection module. A distance limit can be specified to speed-up calculations when the pair distance are far apart and distance calculation can be neglected.

V-REP's **inverse kinematics** calculation module is a very powerful and flexible tool. It allows kinematics calculations, both forward and inverse, for any type of mechanism. The problem of inverse kinematics consists of finding a set of joint values corresponding to some specific position and orientation of a body element, generally the end-effector. More generally, it is a transformation of coordinates from the task space into the joint space. The inverse problem is finding the end-effector, or any system element position, given the joint values - referred to as forward kinematics problem. Last is often perceived as an easier task, especially when dealing with open kinematic chains. In this context, calculations are based on the damped least squares pseudo-inverse method. It supports conditional, damped/undamped, and weighted resolution. Since this is a module used in this work, a more detailed explanation on how it work is given.

V-REP uses IK groups and IK elements to solve inverse and forward kinematic tasks, and an IK group contains one or more IK elements [17]. An IK element represents one simple kinematic chain, which is a linkage containing at least one joint object.

Basically, an IK element is made up by:

- **a base**, representing the start of the kinematic chain.
- **several links**, generally any type of object, except joints.
- **several joints**, in the inverse kinematics mode.
- **a tip**, which is always a *dummy*, and is the last object in the considered chain, the end-effector.
- **a target**, which is always a *dummy* and represents the position/orientation the tip should follow during the simulation.

The tip *dummy* should be linked to the target *dummy*, defining a "tip-target" link type. IK elements are specified by a kinematic chain, and a target to follow (Figure 2.2).

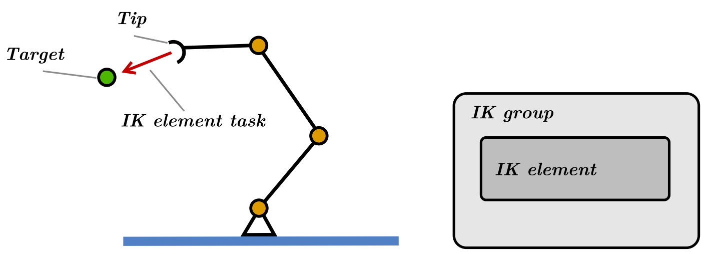


Figure 2.2: IK element and corresponding model of the IK solving task (adapted [17]).

If the parameters have been set correctly and the simulation is running, then the mechanism should move towards the target. This represents the most basic case of IK task. When two separate kinematic chains are simulated the procedure remains the same, but two independent IK groups with one element each must be created. The calculations are sequential, and the solving order is not important. On the other hand, when two or more kinematic chains share common joints, as in Figure 2.3, the difficulty level slightly increases. The kinematics solving cannot be sequential, and a simultaneous method is needed. In this case, the two IK elements should be grouped into one common IK group.

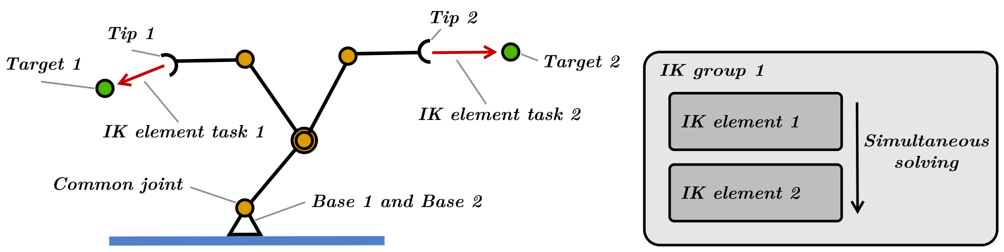


Figure 2.3: IK chains sharing one common joint and the model of the IK solving task (adapted [17]).

Performing inverse kinematics tasks in a biped robot, such as PHUA, is somewhat different. The two existing kinematic chains, robot legs, do not share a common joint, but instead, a common base (considering the chains are built from the pelvis to the feet). The kinematics involved is no longer serial, but parallel. The solution found to the IK solving is explained with the model construction.

The **geometric constraint solver** is an alternative to fast and accurate inverse kinematics calculation module, sometimes difficult to implement depending on the mechanism's complexity. This module is slower and less precise at solving kinematic problems, however it might be easier and more intuitive.

V-REP's **dynamics** module allows handling rigid body dynamics calculation and interaction. It currently supports three different physics engines: the *Bullet Physics Library*, the *Open Dynamics Engine*, and the *Vortex Dynamics Engine*. Physics simulation is a very complex task that can be achieved with different degrees of precision and speed, it is important to not to rely on one single physics engine, in order to validate results. During the experiments, both ODE and Bullet were used to simulate the mechanism.
The dynamics module makes possible the simulation of a wide variety of scenarios (objects falling, collisions, grasping objects, rolling vehicles, etc.) in such a realistic way, that they are near to real-world behavior, even though V-REP is not a pure dynamics simulator. It can rather be seen as a hybrid simulator which combines kinematics and dynamics to improve the response for various simulation scenarios.

The **path planning** module handles path planning tasks in 3D-space and in 2D-space for vehicles with non-holonomic motion constraints via an approach derived from the Rapidly-exploring Random Tree (RRT) algorithm. A path planning task usually takes a start position (or start configuration), a goal position (or a goal configuration) and obstacles, objects that the device should not be colliding with, while following a path.

Motion planning tasks for kinematic chains are handled by the **motion planning** module. A motion planning task allows to compute trajectory, usually in the configuration space of the robot, from a start configuration to a goal configuration. It takes into account the robot kinematics, the joint limits, self-collisions and collision between robot and environment.

The above modules are integrated in the simulator and implement by V-REP in a general way, without making any assumptions on the simulation scenes or models. This clearly improves the flexibility and portability of the models, moving away from a reliance on external libraries and recompiling or installing needs. The inverse kinematics and dynamics modules are used in the simulations, according to conditions further described.

2.2 Model Construction

The PHUA robot full body model construction, described and discussed in detail in this section, was supported by the V-REP User Manual [17].
In order to build all the body elements that compose the robot, both shapes and geometrical information, such as physical dimensions (lengths and center of gravity), masses, and inertial matrices were taken from the robot CAD model, developed in *CATIAv5* environment. Joint properties, such as actuation type, position, and limitations are defined according to the real-world PHUA platform.

CAD model and imported *shapes*

Starting from the CAD model developed in past works [19, 20], the lower and upper bodies' numerous parts and physical constraints were rearranged and redefined to match the real robot's body links and degrees-of-freedom. This reorganization groups the different CAD *parts* into logical CAD *products* in line with the robot fundamental elements. The CATIAv5 model was built from the ground up, assembling elements consecutively from a reference frame defined at the contact point of the left foot with the ground. Thus, since global frames coincide, all body shapes will be correctly positioned when imported into V-REP. Also, the correct definition of physical constraints between these products (contact, concentricity and distance constraints), will allow to establish the joints' position, unequivocally.

V-REP uses triangular meshes to describe and display shapes. Because of this, V-REP will only import formats that describe objects as triangular meshes. *CATPart* and *CATProduct* files, that describe objects as parametric surfaces, have to be converted to an appropriate triangular mesh format. CATIAv5 supports part conversion to *STL* (ASCII or binary) format. The different body parts were then saved as *CADPart* and converted to *STL* format.

Each one of the CAD model parts is imported individually, requiring only the mesh scale and orientation confirmed (Z vector is up). It avoids any further subdivision or grouping operations of the imported objects. Since CATIAv5 and V-REP reference frames coincide, the exact location of all shapes is guaranteed. The coordinate frames of all *shapes* are then aligned with the world reference, in case X and Y directions do not coincide.

Import operation leaves several simple shapes that compose the different rigid entities. Despite color and texture properties of the outside faces, the robot's visual appearance is shown in Figure 2.4.

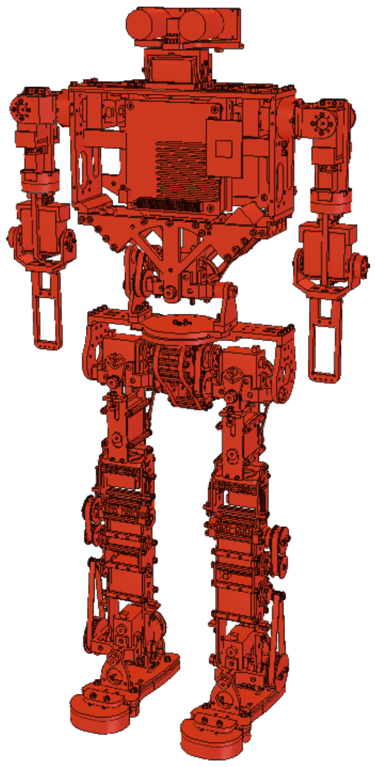


Figure 2.4: Visual appearance of the V-REP model.

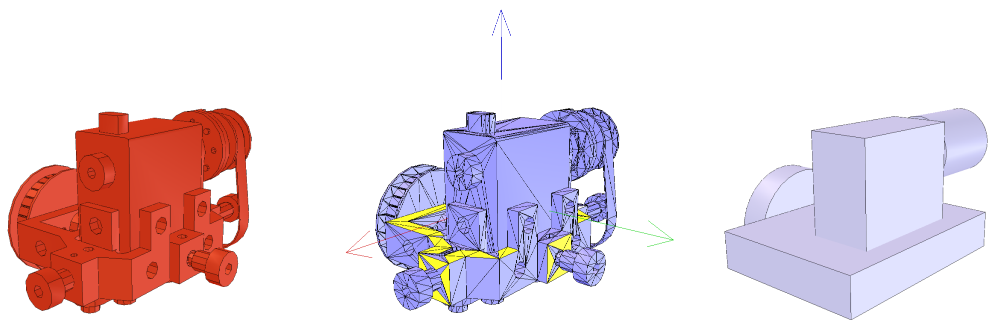
At this point, shapes cannot interact with each or with the environment. By default, all imported shapes are simple static shapes that will not be influenced during dynamic simulation. In V-REP only shapes, joint and force sensors will be dynamically simulated, however depending on the scene structure and object properties.

Shapes can be classified into four groups according to their behavior during dynamic simulation: *static*, *non-static*, *respondable* and *non-respondable*. Unlike static shapes, not-static shapes will be directly influenced by gravity or other constraints as dynamically enabled joints and force sensors. Respondable shapes influence each other during dynamic collision (i.e. they produce a mutual collision reaction, they will bounce off each other). Although dynamics can directly operate on triangular meshes, calculations would take much calculation time and results wouldn't be satisfying. The best practice is using pure shapes for dynamic simulations, whenever possible. Pure shapes are much more stable and faster during simulation, and should be always preferred to approximate triangular meshes as non-static and respondable shapes.

Pure *shapes* extraction

A simplified model is then created that will be used only for dynamics calculations. Pure shapes (i.e. primitive shapes as cuboids, cylinder or spheres), can be extracted from complex non-pure shapes, through triangle edit mode. In this mode all triangles composing the shape are individually displayed. Cuboids, cylinders and spheres can be created by selecting triangles that clearly delimit the desired shape. This procedure will orient a new pure shape in a same way as the minimum bounding box encompassing the selected triangles.

When the original shapes are too complex, as in this case, they can be approximated with a series of pure primitive shapes that are then grouped together to build each one of the robot parts already described, Figure 2.5.



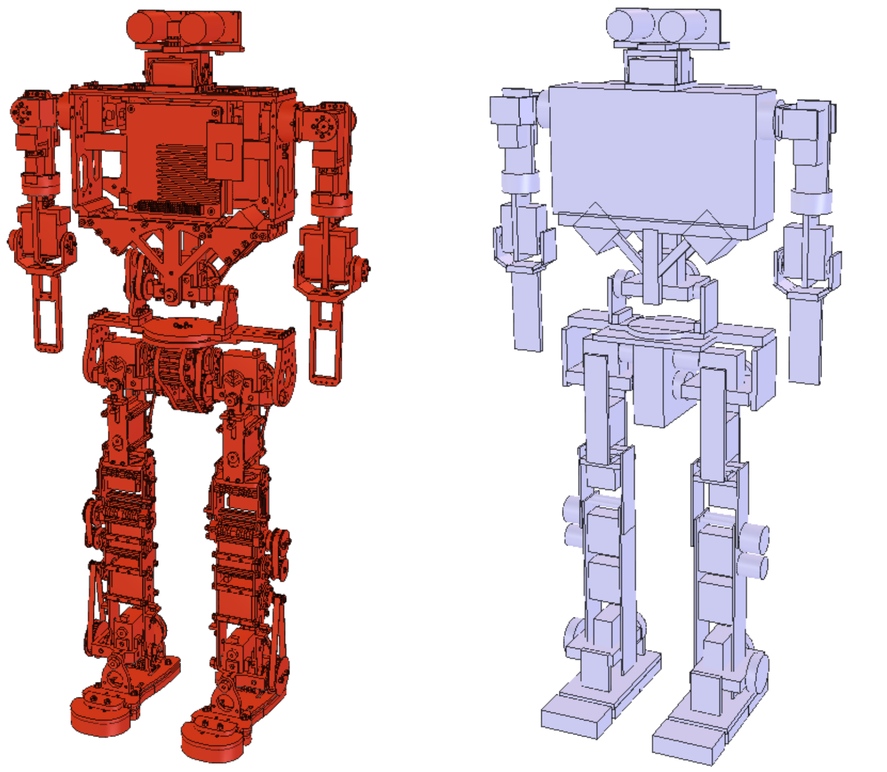
(a) Imported shape (b) Triangle edit mode (c) Grouped pure shapes

Figure 2.5: Example of pure shapes extraction from triangular meshes.

These shapes differ from the earlier in dynamics properties, and will be *falling* and react to *collisions* when physics engine performs on them. To summarize, pure shapes are mainly functional shapes, only used by the physics engine that performs on them much better and faster than on non-pure shapes. For collision detection (different from collision response), distance calculations, vision sensor detection, or proximity sensor, detailed shapes (the non-pure triangular meshes) are used, however for dynamics calculations pure shapes are chosen. Grouped shapes will behave as a rigid entity as share identical dynamic properties.

There are times when the extraction of a pure shape can be a bit complicated. When it is too time-consuming to extract pure shapes, or to arrange them appropriately, convex shapes could be a viable alternative too. The convex decomposition tool will calculate and add the convex decomposition of specified shapes. Although they are not as efficient as pure shapes, they are much better than random shapes.

The simplified model used for dynamic simulation looks as illustrated in Figure 2.6. The outcome of the triangular meshes edition is very accurate. Almost all of the geometries were extracted from the originals, in order to maintain the model as close as possible to the reality in dynamic simulation. Although the *pure-shaped* model construction was a time-consuming process, the results from the first simulations were significantly better in this model, when compared to the results from the convex model. Specially in this case, since intricate shapes are involved.



(a) Model's external appearance.

(b) Model optimized for dynamic simulation.

Figure 2.6: V-REP *pure-shaped* model optimized for dynamic simulation.

Inertial parameters and linked *shapes*

Once shapes are organized in the scene hierarchy, the masses, moments of inertia, position of the center of mass, friction coefficients and other pure shape properties have to be set, for best simulation performance. Table 2.1 details the geometrical data used to build each rigid entity of the lower limbs, according to Figure 2.7. The different links constituting the upper body are also displayed in Figure 2.8, and their inertial parameters registered in Table 2.2, for future reference. The first values correspond to the left body parts, and the second to the right side. The coordinates of the centers of mass are measured with respect to the CATIAv5 reference frame, placed at the bottom of the left foot, with the x axis pointing towards the right foot, in the lateral plane, and the y axis pointing forward. The z axis follows the positive convention. The inertia matrices are expressed relative to the center of mass.

Non-static shapes will fall (i.e. be influenced by gravity), and move independently from each other during dynamic simulation, if they are not otherwise constrained. Dynamic constraints between two shapes can be set-up by attaching two shapes together with a *dynamically enabled joint* or with a *dynamically enabled force sensor*. Dynamically enabled joints are joints that are in force or torque mode or that operate in hybrid fashion, and that have a shape as parent object as exactly one child object which must be a non-static shape. Dynamically enabled sensors are force sensors that also have a shape as parent object and exactly one child object which must be a non-static shape. This is the "regular" configuration. Figure 2.9 shows the valid situations, given that the joint/force sensor and the two shapes are located in a model that is dynamically simulated.

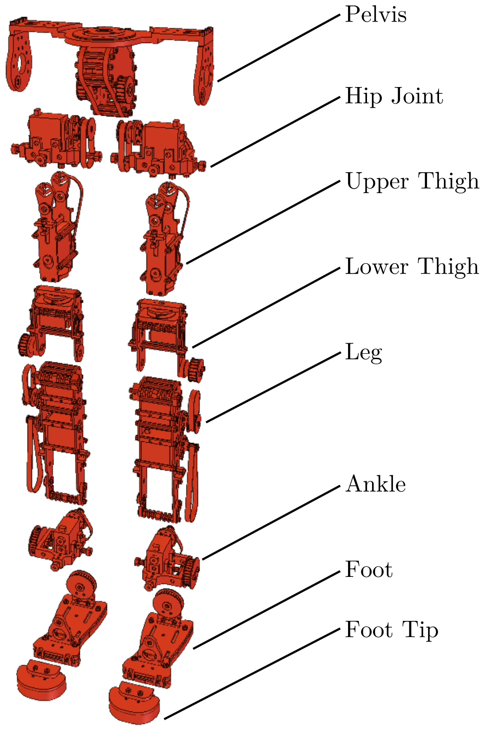


Figure 2.7: Lower limbs parts.

Table 2.1: PHUA robot model geometrical data used to build the lower body parts.

Body Part	**Mass** $[kg]$	**Center of Mass** $[m]$	**Inertia Matrix** $[kg \cdot m^2]$
Pelvis	0.445	$[0.059; -0.008; 0.353]$	$\begin{bmatrix} 3.550 \cdot 10^{-4} & -2.037 \cdot 10^{-6} & 4.640 \cdot 10^{-6} \\ -2.037 \cdot 10^{-6} & 6.344 \cdot 10^{-4} & -5.053 \cdot 10^{-5} \\ 4.640 \cdot 10^{-6} & -5.053 \cdot 10^{-5} & 5.451 \cdot 10^{-4} \end{bmatrix}$
Hip Joint	0.093	$[0.006; -0.004; 0.342]$ $[0.112; -0.004; 0.342]$	$\begin{bmatrix} 4.205 \cdot 10^{-5} & \mp 2.908 \cdot 10^{-6} & \mp 9.533 \cdot 10^{-6} \\ \mp 2.908 \cdot 10^{-6} & 4.855 \cdot 10^{-5} & -4.694 \cdot 10^{-6} \\ \mp 9.533 \cdot 10^{-6} & -4.694 \cdot 10^{-6} & 5.280 \cdot 10^{-5} \end{bmatrix}$
Upper Thigh	0.072	$[0.000; -0.004; 0.290]$ $[0.118; -0.004; 0.290]$	$\begin{bmatrix} 6.087 \cdot 10^{-5} & \mp 1.279 \cdot 10^{-8} & \pm 1.203 \cdot 10^{-8} \\ \mp 1.279 \cdot 10^{-8} & 4.058 \cdot 10^{-5} & -4.374 \cdot 10^{-6} \\ \pm 1.203 \cdot 10^{-8} & -4.374 \cdot 10^{-6} & 2.648 \cdot 10^{-5} \end{bmatrix}$
Lower Thigh	0.083	$[-0.010; 0.002; 0.227]$ $[0.128; 0.002; 0.227]$	$\begin{bmatrix} 6.558 \cdot 10^{-5} & \mp 4.537 \cdot 10^{-7} & \mp 2.600 \cdot 10^{-5} \\ \mp 4.537 \cdot 10^{-7} & 9.358 \cdot 10^{-5} & -4.866 \cdot 10^{-9} \\ \mp 2.600 \cdot 10^{-5} & -4.866 \cdot 10^{-9} & 3.667 \cdot 10^{-5} \end{bmatrix}$
Leg	0.156	$[-0.006; 0.002; 0.145]$ $[0.124; 0.002; 0.145]$	$\begin{bmatrix} 2.907 \cdot 10^{-4} & \mp 1.001 \cdot 10^{-6} & \pm 3.426 \cdot 10^{-6} \\ \mp 1.001 \cdot 10^{-6} & 3.341 \cdot 10^{-4} & -7.394 \cdot 10^{-6} \\ \pm 3.426 \cdot 10^{-6} & -7.394 \cdot 10^{-6} & 6.043 \cdot 10^{-5} \end{bmatrix}$
Ankle	0.084	$[-0.005; -0.004; 0.058]$ $[0.123; -0.004; 0.058]$	$\begin{bmatrix} 3.597 \cdot 10^{-5} & \pm 1.985 \cdot 10^{-6} & \mp 3.809 \cdot 10^{-6} \\ \pm 1.985 \cdot 10^{-6} & 3.616 \cdot 10^{-5} & 5.306 \cdot 10^{-6} \\ \mp 3.809 \cdot 10^{-6} & 5.306 \cdot 10^{-6} & 4.550 \cdot 10^{-5} \end{bmatrix}$
Foot	0.064	$[0.000; 0.009; 0.024]$ $[0.118; 0.009; 0.024]$	$\begin{bmatrix} 1.167 \cdot 10^{-4} & \mp 4.483 \cdot 10^{-15} & \pm 1.292 \cdot 10^{-10} \\ \mp 4.483 \cdot 10^{-15} & 2.834 \cdot 10^{-5} & 1.963 \cdot 10^{-5} \\ \pm 1.292 \cdot 10^{-10} & 1.963 \cdot 10^{-5} & 1.122 \cdot 10^{-4} \end{bmatrix}$
Foot Tip	0.058	$[0.000; 0.075; 0.012]$ $[0.118; 0.075; 0.012]$	$\begin{bmatrix} 6.801 \cdot 10^{-6} & \mp 1.224 \cdot 10^{-22} & \mp 8.831 \cdot 10^{-22} \\ \mp 1.224 \cdot 10^{-22} & 1.529 \cdot 10^{-5} & 7.217 \cdot 10^{-7} \\ \mp 8.831 \cdot 10^{-22} & 7.217 \cdot 10^{-7} & 1.785 \cdot 10^{-5} \end{bmatrix}$

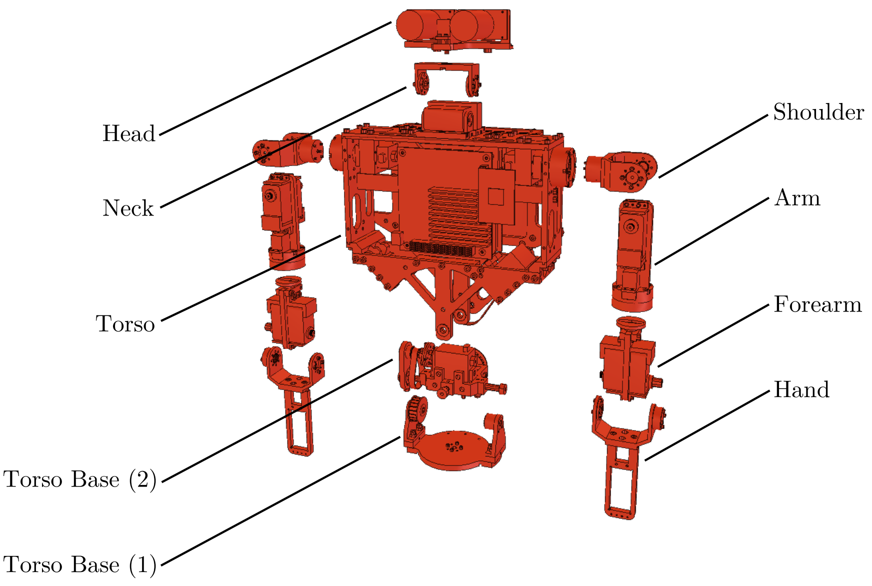
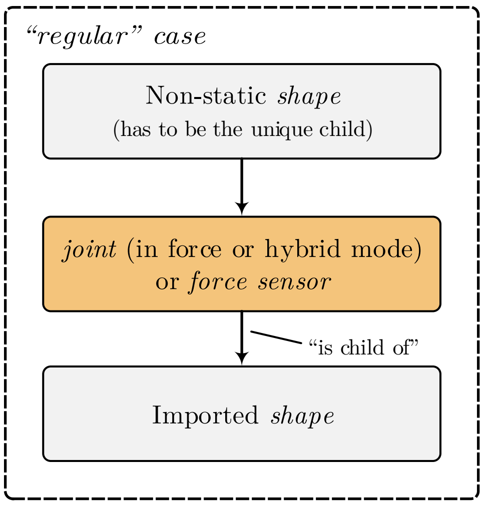


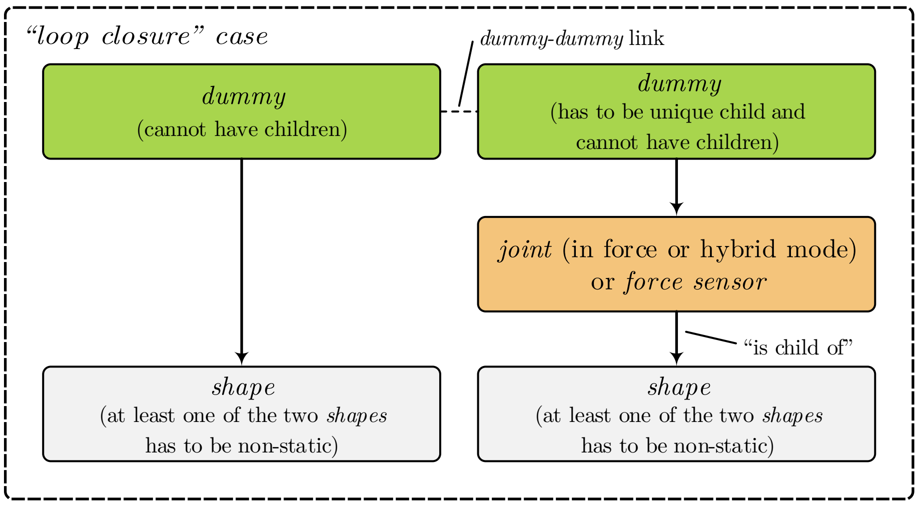
Figure 2.8: Upper body parts.

Table 2.2: PHUA robot model geometrical data used to build the upper body parts.

Body Part	**Mass** $[kg]$	**Center of Mass** $[m]$	**Inertia Matrix** $[kg \cdot m^2]$
Torso Base (1)	0.060	$[0.064; 0.001; 0.404]$	$\begin{bmatrix} 2.463 \cdot 10^{-5} & 7.514 \cdot 10^{-8} & -5.496 \cdot 10^{-6} \\ 7.514 \cdot 10^{-8} & 1.002 \cdot 10^{-4} & 3.176 \cdot 10^{-7} \\ -5.496 \cdot 10^{-6} & 3.176 \cdot 10^{-7} & 8.970 \cdot 10^{-5} \end{bmatrix}$
Torso Base (2)	0.122	$[0.073; -0.006; 0.432]$	$\begin{bmatrix} 5.595 \cdot 10^{-5} & -1.143 \cdot 10^{-5} & -1.113 \cdot 10^{-5} \\ -1.143 \cdot 10^{-5} & 9.060 \cdot 10^{-5} & -6.956 \cdot 10^{-6} \\ -1.113 \cdot 10^{-5} & -6.956 \cdot 10^{-6} & 1.055 \cdot 10^{-4} \end{bmatrix}$
Torso	1.638	$[0.057; -0.020; 0.543]$	$\begin{bmatrix} 4.000 \cdot 10^{-3} & 2.340 \cdot 10^{-5} & -2.447 \cdot 10^{-4} \\ 2.340 \cdot 10^{-5} & 8.000 \cdot 10^{-3} & 2.359 \cdot 10^{-5} \\ -2.447 \cdot 10^{-4} & 2.359 \cdot 10^{-5} & 6.000 \cdot 10^{-3} \end{bmatrix}$
Shoulder	0.049	$[-0.064; 0.003; 0.579]$ $[0.182; 0.003; 0.579]$	$\begin{bmatrix} 1.817 \cdot 10^{-5} & \mp 3.147 \cdot 10^{-6} & \pm 1.502 \cdot 10^{-9} \\ \mp 3.147 \cdot 10^{-6} & 1.310 \cdot 10^{-5} & -2.065 \cdot 10^{-9} \\ \pm 1.502 \cdot 10^{-9} & -2.065 \cdot 10^{-9} & 2.744 \cdot 10^{-5} \end{bmatrix}$
Arm	0.146	$[-0.083; 0.006; 0.548]$ $[0.201; 0.006; 0.548]$	$\begin{bmatrix} 1.285 \cdot 10^{-4} & -9.154 \cdot 10^{-9} & -5.317 \cdot 10^{-16} \\ -9.154 \cdot 10^{-9} & 1.196 \cdot 10^{-4} & -1.405 \cdot 10^{-6} \\ -5.317 \cdot 10^{-16} & -1.405 \cdot 10^{-6} & 2.386 \cdot 10^{-5} \end{bmatrix}$
Forearm	0.105	$[-0.081; 0.005; 0.472]$ $[0.199; 0.005; 0.472]$	$\begin{bmatrix} 3.853 \cdot 10^{-5} & 3.728 \cdot 10^{-8} & \mp 1.042 \cdot 10^{-7} \\ 3.728 \cdot 10^{-8} & 4.692 \cdot 10^{-5} & \pm 5.421 \cdot 10^{-20} \\ \mp 1.042 \cdot 10^{-7} & \pm 5.421 \cdot 10^{-20} & 1.674 \cdot 10^{-5} \end{bmatrix}$
Hand	0.048	$[-0.084; 0.005; 0.429]$ $[0.202; 0.005; 0.429]$	$\begin{bmatrix} 4.184 \cdot 10^{-5} & -8.227 \cdot 10^{-12} & \pm 4.341 \cdot 10^{-6} \\ -8.227 \cdot 10^{-12} & 5.758 \cdot 10^{-5} & \pm 8.366 \cdot 10^{-12} \\ \pm 4.341 \cdot 10^{-6} & \pm 8.366 \cdot 10^{-12} & 1.867 \cdot 10^{-5} \end{bmatrix}$
Neck	0.021	$[0.059; 0.010; 0.625]$	$\begin{bmatrix} 1.908 \cdot 10^{-6} & 3.207 \cdot 10^{-18} & -4.337 \cdot 10^{-19} \\ 3.207 \cdot 10^{-18} & 9.692 \cdot 10^{-6} & 4.403 \cdot 10^{-10} \\ -4.337 \cdot 10^{-19} & 4.403 \cdot 10^{-10} & 8.959 \cdot 10^{-6} \end{bmatrix}$
Head	0.029	$[0.059; 0.001; 0.650]$	$\begin{bmatrix} 8.813 \cdot 10^{-5} & 3.008 \cdot 10^{-17} & -5.204 \cdot 10^{-18} \\ 3.008 \cdot 10^{-17} & 1.615 \cdot 10^{-4} & -1.758 \cdot 10^{-5} \\ -5.204 \cdot 10^{-18} & -1.758 \cdot 10^{-5} & 2.110 \cdot 10^{-4} \end{bmatrix}$



(a) *Regular* configuration.



(b) *Loop closure* configuration.

Figure 2.9: Valid *dynamically enabled joint* or *dynamically enabled force sensor* situations, the "regular" case and the "loop closure" case (adpated [17]).

Two non-static shapes that are not linked properly will behave strangely, compromising the entire simulation, especially when these shapes are used to define long kinematic chains such as humanoid robot legs.

Joints* and *force sensors

A joint has two reference frames. The first one is the regular reference frame that is fixed and general to any object. The second reference frame is not fixed, and will move relative to the first depending on the joint angle (value) that defines its configuration.

When joints are added into the scene and their exact positions are unknown, those locations have to be extracted from the existing shapes. Imported shapes containing servo motors can give the correct position and orientation of robot joints. As with the pure shapes definition, the imported shapes are manipulated in triangle edit mode to extract shapes defining the servo motors' output shafts. These extracted shapes can be used to precisely position model joints, as shown in Figure 2.10.

All 27 joints of the humanoid model are revolute joints, used to describe rotational movements

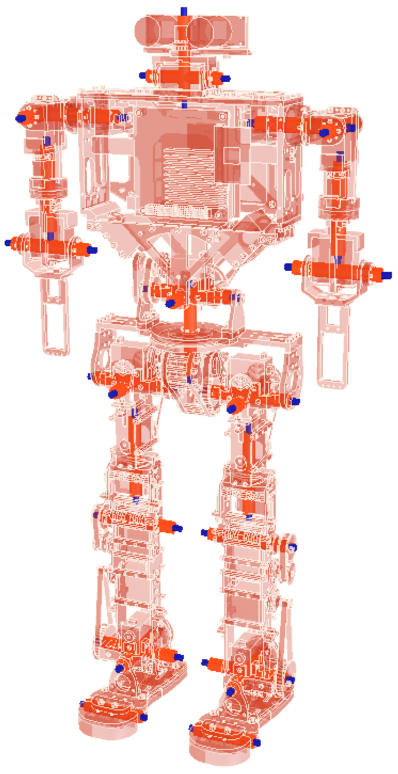


Figure 2.10: PHUA model kinematic chains.

(1 DOF) between the objects. A joint is used to provide relative movement between its parent and its children. When a parent-child relationship is built between a joint and an object, the object is attached to the joint's second reference frame, thus, a change of the joint's configuration (intrinsic position) will directly be reflected in the relative position of its children.

When an inverse kinematics model simulation is intended, with exception of the foot tip and torso base joints, which are controlled in torque/force mode, all the remaining joints are controlled in inverse kinematics mode. While operating in this fashion, all the joints constituting the kinematic chains must be defined accordingly, so that the V-REP can perform the calculations properly. In torque mode, the joint is directly controlled. This means that, if a zero value is sent to the joint it will act as a fixed link. To dynamically simulate the robot in the inverse kinematics mode, the joints must operate in an inverse kinematics hybrid fashion. Only thus, the physics engine can perform on them, and the dynamic linkage can be ensured.

Alternatively, when simulation in torque/force mode is desired, the joints have to be set to operate in this mode. Then, if dynamically enabled, they will be simulated by the dynamics module, reacting to gravity and model constraints. This fashion allows the joint to be controlled in position by a PID controller. The programmable PID parameters will try to drive the joint to the desired target position. Also, when in passive mode, a joint can optionally also be operated in a hybrid fashion, which allows the joint to operate in a regular way, but additionally, be simulated by dynamic module.

Force sensors present in PHUA platform are also incorporated in V-REP model. Force sensors as simulation objects, are initially rigid links between to shapes that are also able to measure transmitted forces and torques during simulation. The rigidity of force sensors is conditional, in the sense that they can be broken if a force or torque threshold is overshot. A force sensor is only operational during simulation if it is dynamically enabled.

In the present case, four sensors are placed in each foot to reproduce, as closely as possible, the real world model and its conditions. They are placed exactly in the same position as the real ones, establishing contact between the feet and their soles (see Figure 2.11).

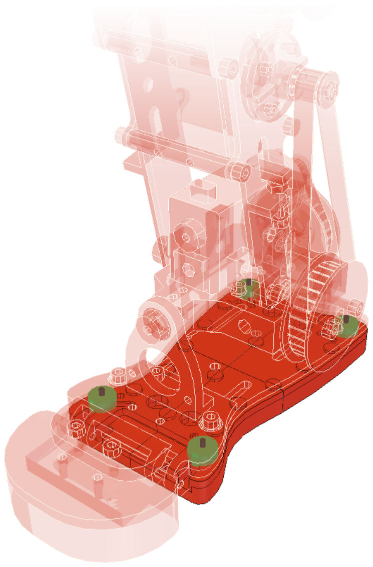


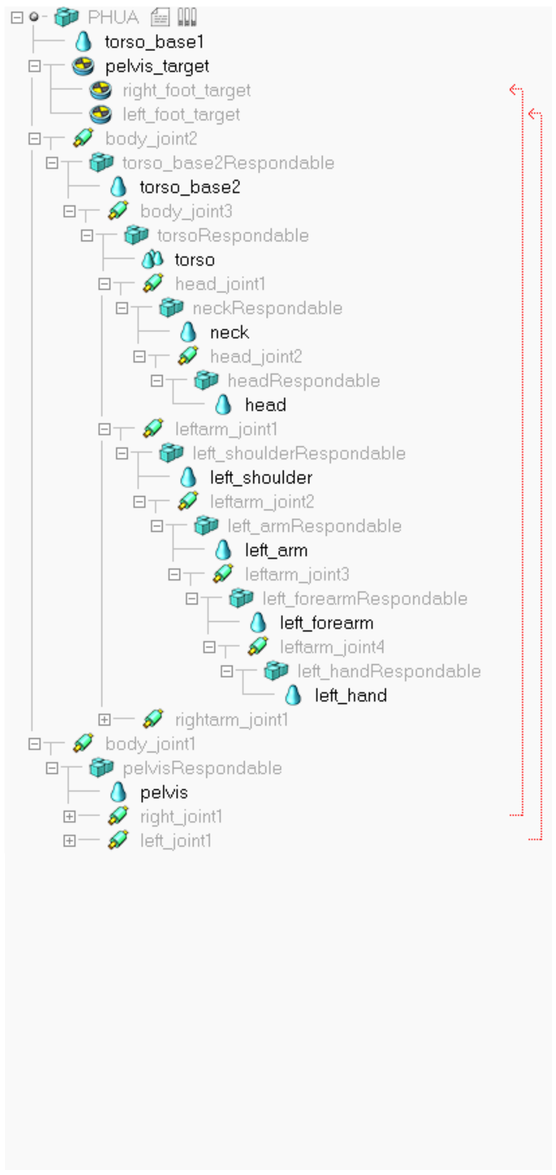
Figure 2.11: Detail of the force sensors' implementation in the model.

Thus, besides measuring the forces transmitted between these objects, they perform as rigid links necessary to define objects interaction. Because dynamically enabled force sensors must have exactly one shape as child object, feet soles have to be divided into four shapes each. To each foot are then connected in parallel, four force sensors, and their corresponding child shapes.

Kinematic chains

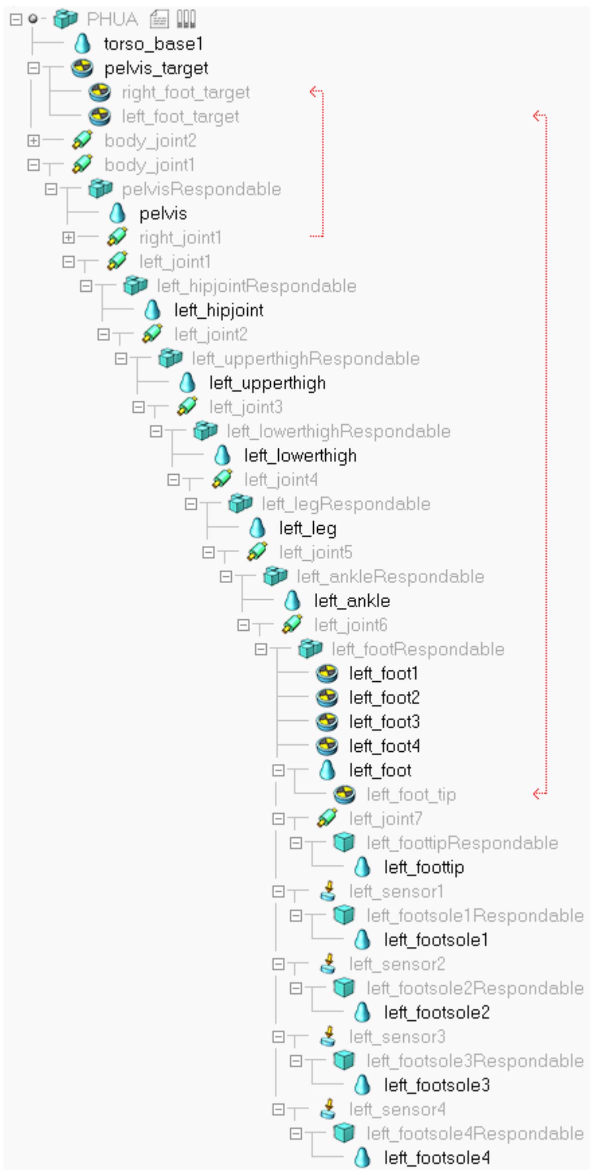
All elements of the mechanism are linked together to build the legs and arms kinematic chains, going from the center body (*Torso Base (1)*). Even when the model does not operate in inverse kinematics mode, but in torque mode instead, the kinematic chains construction must also be implemented to define the correct interaction between consecutive shapes and guarantee a natural behavior at simulation time. Successive *parent-child* relationships are defined until the scene hierarchy is completed, as shown in Figure 2.12. This figure describes the model general structure, illustrating important features, such as the parallel arrangement of the force sensors, and the "tip-target" links for inverse kinematics simulation (Figure 2.12(b)). The four *dummy* objects linked to each foot represent the shape limits. Their position is used to calculate the support base of the robot.

In one of the simulation experiments, later described, the user teleoperates the robot's hip while keeping the feet together, and fixed to the ground. A traditional inverse kinematics task associated with humanoid robots. When inverse kinematics calculations are intended, this mode must be enabled, and the links between tips and targets must be established. This robot is characterized by two parallel kinematics chains, the left and the right legs. To each one, a tip and a foot target *dummy* are defined, and connected. However, in order to simulate the hip's motion, it is the pelvis position that must be controlled, and not the feet. To make it possible, a fifth *dummy* object, denominated *pelvis_target*, is created. By establishing a *parent-child* relationship between this object and the existing targets, as represented in Figure 2.12, both feet are controlled by the *pelvis_target*. Due to the model's weight, when the *pelvis_target* moves relatively to its parent frame (model frame), and consequently the inverse kinematics calculations for both legs are performed, the feet are kept on the ground and the hip moves. But, in the opposite direction.



```
PHUA
  torso_base1
  pelvis_target
    right_foot_target
    left_foot_target
  body_joint2
    torso_base2Respondable
      torso_base2
      body_joint3
        torsoRespondable
          torso
          head_joint1
            neckRespondable
              neck
              head_joint2
                headRespondable
                  head
          leftarm_joint1
            left_shoulderRespondable
              left_shoulder
              leftarm_joint2
                left_armRespondable
                  left_arm
                  leftarm_joint3
                    left_forearmRespondable
                      left_forearm
                      leftarm_joint4
                        left_handRespondable
                          left_hand
          rightarm_joint1
  body_joint1
    pelvisRespondable
      pelvis
      right_joint1
      left_joint1
```

(a) Extended torso and left arm kinematic chains.



```
PHUA
  torso_base1
  pelvis_target
    right_foot_target
    left_foot_target
  body_joint2
  body_joint1
    pelvisRespondable
      pelvis
      right_joint1
      left_joint1
        left_hipjointRespondable
          left_hipjoint
          left_joint2
            left_upperthighRespondable
              left_upperthigh
              left_joint3
                left_lowerthighRespondable
                  left_lowerthigh
                  left_joint4
                    left_legRespondable
                      left_leg
                      left_joint5
                        left_ankleRespondable
                          left_ankle
                          left_joint6
                            left_footRespondable
                              left_foot1
                              left_foot2
                              left_foot3
                              left_foot4
                              left_foot
                                left_foot_tip
                              left_joint7
                                left_foottipRespondable
                                  left_foottip
                              left_sensor1
                                left_footsole1Respondable
                                  left_footsole1
                              left_sensor2
                                left_footsole2Respondable
                                  left_footsole2
                              left_sensor3
                                left_footsole3Respondable
                                  left_footsole3
                              left_sensor4
                                left_footsole4Respondable
                                  left_footsole4
```

(b) Extended left leg kinematic chain.

Figure 2.12: V-REP model kinematic chains construction, detailing the "tip-target" inverse kinematics links.

Chapter 3

Experimental Setup

In this chapter, the primary hardware and software solutions necessary for the implementation of the haptic demonstration are presented. First, a detailed description of the chosen haptic joystick and its dedicated library is given, making reference to hardware and software limitations. Then, the experimental setup adopted is explained, including the libraries and programming platforms used to develop the software applications.

3.1 PHANToM Omni Haptic Device

Although there have been many devices developed for haptic applications, SensAble's PHANToM Omni and Desktop models have gained wide popularity among the researchers in haptics community. Recent haptics applications such as medical simulations, dental restorations, stroke rehabilitation, machine interface design, robotic control and skills assessment are just a few examples of applications for which those devices can be used [21].

Force feedback devices exist earlier than 1990s, but the first successful force reflecting haptic interface device PHANToM, came into existence only in 1994. Its wide use in a multitude of applications is predominantly due to its generic in nature, large workspace, low inertia, low friction and high position precision characteristics.

The haptic device used in this work is a Geomagic Touch (formerly SensAble PHANToM Omni) ground-based haptic joystick, as shown in Figure 3.1.

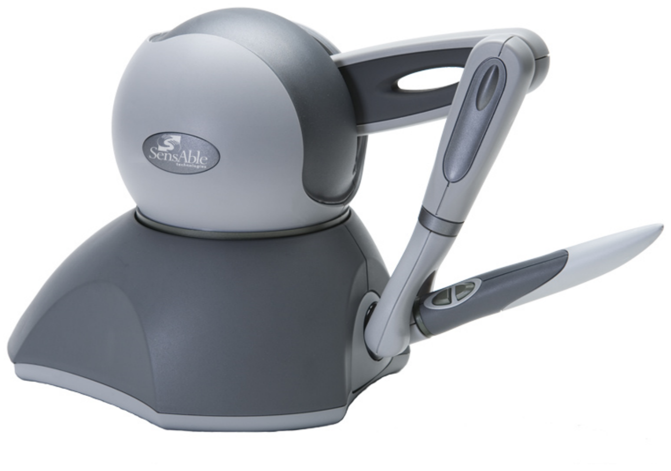


Figure 3.1: Geomagic Touch (formerly SensAble PHANToM Omni) [22].

This model is currently the most cost-effective haptic device available. It is a 6-DOF *impedance-based* joystick, capable of rendering three-dimensional force vectors at high frequencies. Its intrinsic mechanical behavior simulates mechanical impedance, reading position and sending forces. *Admittance* haptic devices, on the other hand, simulate mechanical admittance - they read force and send position [3]. Impedance-type architecture are most common, due to their simple and low-cost design. The Omni model connects to the computer station via IEEE 1394 FireWire port, and can, hypothetically, be connected in chain to another identical equipment [22].

Despite possessing six DOF, from which it is possible to retrieve a correspondent three-dimensional Cartesian position and orientation, only the first three joints are actuated. This means it is only possible to generate haptic forces in the translational directions, but not torques (individually) with the last three joints. Rendering torques to the last three DOF is not possible. This is due its mechanical construction. This device is built with a capstan drive actuating two out of the three base degrees-of-freedom, allowing for its compact form and good force feedback level on its actuated workspace, coupled with high precision optical encoders. Active force feedback on the axis x, y and z relies on the measurement of instantaneous position and velocity provided by these joint proprioceptive sensors (encoders). The last three DOF are passive joints, equipped with potentiometers to track device's orientation [22]. A detailed list of PHANToM Omni specifications is shown on Table 3.1.

Table 3.1: PHANToM Omni device specifications [22].

Force Feedback Workspace	160 W × 120 H × 70 D [mm]
Footprint (physical area the base of the device occupies)	168 W × 203 D [mm]
Weight (device only)	~1.47 kg
Range of Motion	Hand movement pivoting at wrist
Nominal Position Resolution	450 dpi ~0.055 mm
Backdrive Friction	0.26 N
Maximum Exertable Force (at nominal position)	3.3 N
Continuous Exertable Force (24 hours)	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	x, y, z (digital encoders) roll, pitch, yaw (±5% linearity potentiometers)
Interface	IEEE-1394 FireWire port (6-pin to 6-pin)

The six degrees of motion are provided by six axis points. All the degrees of motion have physical limits that are reached when the user feels a sudden stop. This is the mechanical stop designed into the device. In fact, the workspace where the device is capable of moving and exerting forces is quite small. When operating the Omni the user must be very cautious, because forcing the device to past any of these stops risks damaging the device. The operator should never leave the device "free" to exert forces, because it will push itself beyond its own physical limits.

The PHANToM Omni is basically a 6-DOF manipulator, with six rotational joints. The first three joints give translational movements for the end-effector while the last three provide the rotational movements or Euler angles. Therefore, its base architecture can be visualized as 3-bar Revolute-Revolute-Revolute (RRR) spatial manipulator. The kinematic diagram is shown in Figure 3.2. Device's forward, inverse and differential kinematics can be derived from this configuration.

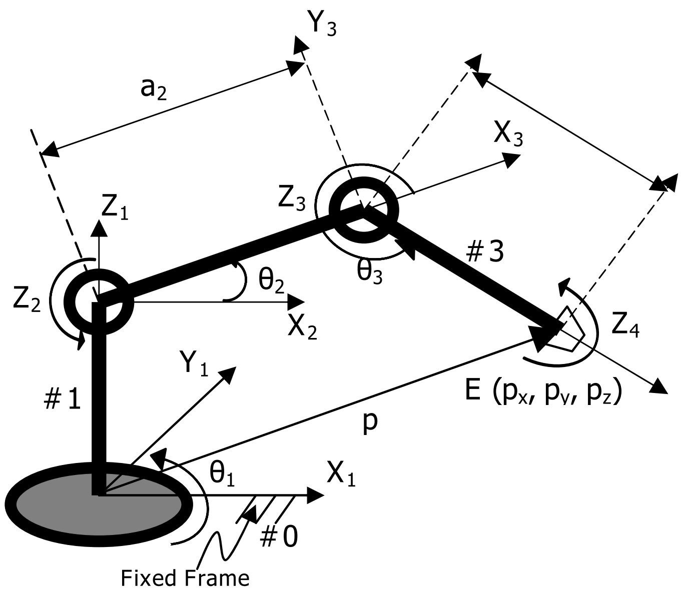


Figure 3.2: Kinematic diagram of PHANToM Omni [21].

The contributions of this haptic device require its forward and inverse kinematic models. They allow knowing the human operator performance and his representation in the virtual world, as well as defining the reaction forces produced at the interaction with virtual objects. However, because neither haptic applications' development nor proper control methods are needed in this work, kinematics and dynamic equations of motion for the device are not presented in the document. To an approach to explicit mathematical models and alternative dynamic considerations, refer to [21, 23].

The software supplied by SensAble, described in the next section, is well built and provides all the means necessary to implement the device's control.

3.2 OpenHaptics Toolkit

The OpenHaptics Toolkit is a proprietary programming solution developed in C/C++, offered by SensAble to build haptically enabled applications. Although it is available for Windows, Linux and MacOS X, its latest version, OpenHaptics 3.2.2, is only available for Windows machines. Linux based machines have to count with an earlier version of the software, OpenHatics 3.0.

This haptics toolkit handles complex calculations, provides low-level device control and supports different polygonal objects, material properties and force effects. Integration with OpenGL application is facilitated.

The OpenHaptics toolkit includes [24, 25]:

- QuickHaptics micro API;
- Haptic Device API (HDAPI);
- Haptic Library API (HLAPI);
- PHANTOM Device Drivers (PDD);
- Utilities;
- Source Code Examples;
- Documentation;

QuickHaptics/HDAPI/HLAPI

OpenHaptics 3.0 allows to ease programming by encapsulating the basic steps common to all haptics/graphics applications in the C++ classes of the QuickHaptics micro API, which makes use of the Standard Template Library (STL). By anticipating typically used scenarios and using built-in geometry parsers and intelligent default parameters, it makes possible to set up haptics/graphics scenes with a minimal amount of code, and very efficiently.

The following diagram illustrates the relationship between the QuickHaptics micro API and the existing HD and HL layers of OpenHaptics. Experts can still use HLAPI and HDAPI in conjunction with QuickHaptics to take advantage of all SDKs.

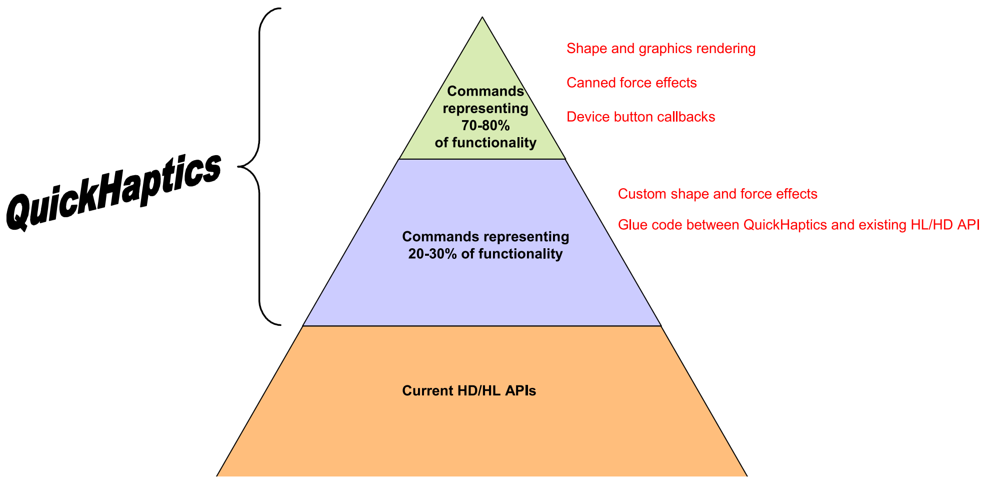

QuickHaptics
Commands representing 70-80% of functionality
Shape and graphics rendering
Canned force effects
Device button callbacks
Commands representing 20-30% of functionality
Custom shape and force effects
Glue code between QuickHaptics and existing HL/HD API
Current HD/HL APIs

Figure 3.3: The OpenHaptics structure diagram [24].

The primary functional classes defined by QuickHaptics API are: *DeviceSpace*, the workspace through which the haptic device can move; *QHRenderer*, on-screen window that renders shapes from a camera viewpoint and lets the user feel those shapes with haptic device; *Shape*, which is a base class for one or more geometric objects that can be rendered both graphically and haptically and *Cursor*, the graphical representation of the end point of the second link on the PHANTOM device. This point is called the haptic interface point, previously denominated *avatar* in this document. These are the core elements of a haptic interaction.

The HDAPI is the foundational layer for haptics, and it is best suited for developers who are already familiar with haptic paradigms and sending forces directly. It provides low-level access to the haptic device, 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.

The HLAPI is designed for high-level haptics scene rendering. It is mainly targeted for advanced OpenGL developers, who are however less familiar with haptics programming, but desire to easily incorporate haptics to existing graphics applications. Synchronization of the haptics and graphics threads is greatly simplified.

The typical application structure under OpenHaptics is depicted in Figure 3.4. Its usual implementation incorporates object visualization with haptic interaction on a virtual environment.

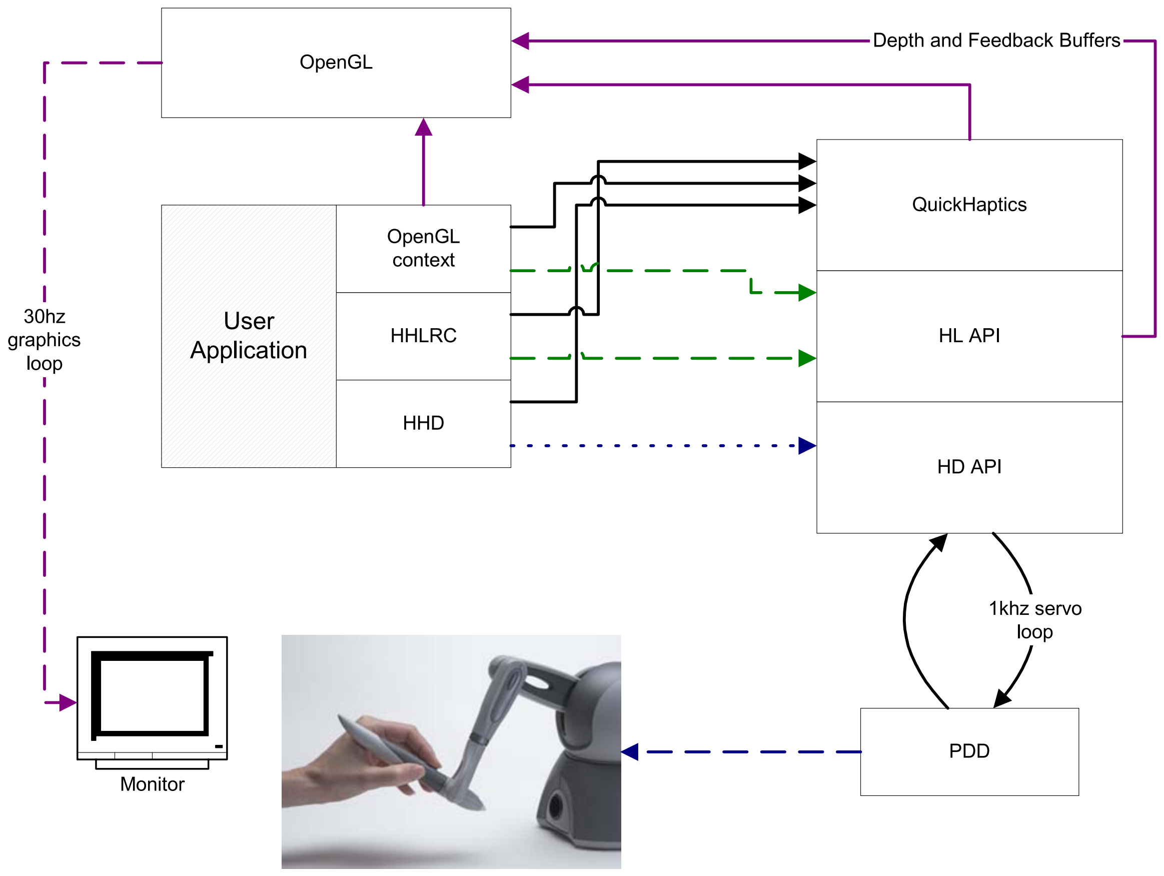


Figure 3.4: A typical OpenHaptics application struture [24].

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. Thereby, HLAPI programmers will not have to concern themselves with such lower level issues as designing force equations, handling thread safety, and implementing highly efficient data structure for haptic rendering. HLAPI also avoids the developer from managing the synchronization of *servo loop*, unlike HDAPI users. Direct force rendering with HDAPI requires efficient force rendering/collision detection algorithms and data structures. This is due to the high frequency of force refreshes, required for a stable closed-loop control of the haptic device.

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. This value was already mentioned, as subjectively acceptable to create compelling and stable force feedback. Lower values can decrease the amount of CPU used for force rendering, but have the expense of reducing stability of the device and lower max nominal stiffness. In order to maintain such a high update rate, the servo loop is generally executed in a separate, high-priority thread - the servo loop thread.

HDAPI offers access to lower-level control spaces, and thereby can be used to query properties and capabilities of the device, for instance: raw encoder and motor joint torque values, nominal force applied, workspace dimensions, etc.

In the context of the present work, the typical structure of haptic/graphics applications is not followed. The aim of the project is to use the PHANToM Omni haptic device, only to generate a force feedback on the user, in response to a particular state of the robot. Collision detection algorithms and physics/dynamics calculations will be carried out by a third-party software, the V-REP simulator, in simulation context. Since no shapes or workspace interactions need to be rendered, graphically and haptically, HDAPI is the main API used in this work.

PHANToM Device Drivers (PDD)

The OpenHaptics toolkit still includes the PHANTOM Device Drivers (PDD). The PHANToM Omni communicates with the computer through the FireWire port. To enable this, a proper software driver must be install. Due to its long and tricky installation procedures, the relevant steps are described in detail in Appendix B, for future reference.

The main goal of this project is to implement a dual PHANToM Omni configuration, to fulfil the ambition of co-teleoperating the PHUA model, at an initial stage, and then the robot itself.

According to proprietary user's guide [22], paired configurations can be set up to make possible a pair of PHANTOM devices to work in tandem with each other. Daisy-chaining two PHANToM Omni devices require the two devices to be initialized and configured individually. The primary device must be connected directly to the computer, and configured. For the second device, the FireWire cable must be connected into the back of the first device rather than to the computer. The computer should be able to recognize and name the two devices properly, according to setup definitions.

Even though preceding conditions were met, and the Omnis were correctly defined, the computer could only put to work one of the two devices. Before connecting both Omnis together, they were each configured individually, and two different configuration files were created. The test application could run successfully on both equipment, when in single configuration. However, when the second device is plugged into the first, *PHANTOMConfiguration* can no longer recognize the two Omnis. The serial number of the primary device is also assigned to the second, and it cannot be changed. As a result, dual configuration cannot be enabled, and the *PHANTOMTest* application can only run successfully on the second device. In another test, both devices were directly connected to the PCI express card of the computer, but with the same outcome.

In fact, dual Omni configurations are currently not supported under Linux. Apparently, dual configuration mode can only be achieved (in Linux) when involving a PHANToM Omni and a PHANToM Desktop. This is due to the old and inappropriate Linux drivers for the PHANToM Omni device. Unfortunately, and despite many requests for support addressed to SensAble, new Linux drivers are not expected to be developed.

3.3 ROS Framework

3.3.1 Introduction and Concepts

The Robot Operating System (ROS) framework is the unifying element of this project. It defines the interaction between the haptic devices and the V-REP simulation, using the functionality already present on the simulator and other software personally developed.

ROS is a framework that is widely used in robotics. The philosophy is to create functionalities that can be shared and used in other robots without much effort. ROS was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory (SAIL). The platform is now maintained by the open-source community and Willow Garage, responsible for the Personal Robots Program (PR/PR2 robots). A lot of research institutions and companies have started to develop projects and adapt their products to be used in ROS. NAO robot is just an example of a fully supported platform. Sensors and actuators used in robotics have also been adapted to be used with ROS. This pseudo-operating system provides standard operating system facilities such as hardware abstraction, low-level device control, implementation of commonly used functionalities, message passing between processes, and package creation and management [26].

Designed to run under a Unix-like system (Ubuntu is currently listed as supported, while some other variants are considered experimental), ROS eases the interaction of hardware and software, using a modular architecture. This architecture is based on a centralized topology where all the processes are connected in a network. Any node in the system can access this network, interact with other nodes, read the information that they are sending, and transmit data to the network [26]. A system built using ROS consists of a number of processes, potentially on a number of different hosts, connected at runtime in a peer-to-peer topology. The fundamental concepts of ROS implementation, regarding this peer-to-peer network, are [27]:

- *Nodes*;
- *Master*;
- *Parameter Server*;
- *Messages*;
- *Topics*;
- *Services*;
- *Bags.*

The definition of these concepts belongs to the realm of *Computation Graph Level* of ROS, where the communication between processes and systems is described.

Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale, so a robot control system is typically comprised of many nodes that control different functions. For example, one node can control the wheel motors, while other performs localization and path planning, etc. In this context, the term ”node” is interchangeable with ”software module”. A ROS node is an executable compiled with the written code. Nodes are written with the use of ROS client libraries that currently fully support C++ and Python languages. Nodes can operate individually or communicate with each other by passing messages.

A **message** is simply a data structure, comprising typed fields. Standard primitive types (such as integer, floating point, boolean, etc) are supported, as are arrays of primitive types and constants. User-defined types can also be included. Messages are routed via a transport system with publish/subscribe semantics.

A node sends a message by publishing it to a given **topic**. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. An interesting fact about ROS is that, in general, publishers and subscribers are not aware of each others’ existence, meaning the production and consumption of data are decoupled.

Figure 3.5 is a common graphic representation of two nodes communicating with each other, using the publish/subscribe semantics.

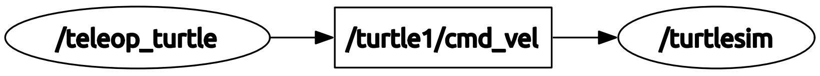


Figure 3.5: Example of two ROS nodes communicating with each other over a ROS Topic.

The */turtlesim* node and the */teleop_turtle* node are communicating with each other over a ROS topic, named */turtle1/cmd_vel*. The */teleop_turtle* is **publishing** information on a topic, while */turtlesim* **subscribes** to the same topic to receive that information.

Although the topic-based publish-subscribe model is a flexible communication paradigm, its "broadcast" routing scheme is not appropriated for synchronous transactions. Request/reply interactions, which are often required in a distributed system, are done via **services**. A providing node can offer a service under a name, and a client can use that service by sending the request message and awaiting for the reply. Services are always defined by paired structure, a request and a reply.

The peer-to-peer topology requires some sort of lookup mechanism to allow processes to find each other at runtime. It is called the *name service*, or **Master**. The ROS Master provides name registration and lookup for the rest of the nodes in the system. The role of the Master is to enable individual ROS nodes to locate one another. Without it, nodes would not be able to find each other, exchange messages, or invoke services. However, it is possible to have it running in a computer, while nodes work in other computers.

Another interesting feature of ROS is that the information of the messages, topics and services can be saved and then played back. A **bag** is a file created with a specific format for storing data that can later be visualized or used to perform other operations, such as reproducing that data to control a real system, online.

ROS is distributed under the terms of a BSD license (permissive free software license), which allows the development of both non-commercial and commercial projects. This makes ROS a very flexible platform, capable of being implemented in any robot.

For humanoid robots, ROS hardware abstraction and low-level device control are valued features, allowing for an easy and generic code conception. Its interconnection between nodes, or modules, means that the PHUA can have different modules performing different tasks, such as monitoring the robot state, computing algorithms to generate force feedback and commanding the humanoid, running at the same time. This gives the user an enormous control over the robot actions when teleoperating it, since he will be able to control it, and also sense how the robot reacts to his orders. The inclusion of ROS in this work allows for a quite efficient robot system construction and maintenance. The software developed in that sense is fully described next.

It should be noted that ROS has several other features and utilities that were not, and will not be, fully described and exploited in the present work.

3.3.2 PHANToM Dual Configuration

Since proprietary PHANToM Omni driver for Linux does not support multiple Omni configuration on a single station, the solution for the dual Omni configuration lies in the use of two computers. Each one of these computers will be responsible for configuring and handling the communication with the haptic device connected to it. Information exchange with the other computer and other hardware/software components, including V-REP, will be done by means of a well-designed ROS framework.

The architecture of ROS is designed with distributed computing in mind. A well written node makes no assumption about where in network it runs, allowing computation to be relocated at run-time to match the available resources (there are exceptions; a driver node that communicate with a piece of hardware must run on the machine the hardware is connected to). The entire framework of this project is designed on top of this principle.

Each one of the haptic devices is connected and configured in only one of the two computers available for the purpose. At ROS level, they run perfectly independent of each other, on two different nodes. These modules were created to handle both hardware and software communication, securing the access to joystick and the connection to V-REP, simultaneously.

Deploying a ROS system across multiple machines can be easy. However, some aspects must be taken into consideration.

To set up the communication between joysticks and V-REP, running on an arbitrary machine, the following conditions must be met [27]:

- Only one ROS master is needed. Therefore, one of the two machines must be selected to run it on.
- All nodes (joystick nodes and others) must be configured to use the same master.
- There must be complete, bi-directional connectivity between the machines, on all ports.
- Each machine must advertise itself by a name that all other machines can resolve.

Full connectivity

In the configuration process, the first thing to do is to check the full connectivity between the machines. There are several different ways of network connecting two computers, for sharing data and resources. The traditional method involves making a dedicated link by plugging one cable into the two systems. An Ethernet crossover cable, a null modem serial cable (or parallel peripheral cable), or a special-purpose USB cable are some of the alternatives for networking two computers in this manner. Of these choices, the Ethernet method is preferred as it supports a reliable, high-speed connection with minimal configuration required. If one of the computers possesses an Ethernet adapter but the other has USB, an Ethernet crossover cable can still be used, but an *USB-to-Ethernet converter* unit must be plugged to the computer's USB port. The serial type of cabling, called *Direct Cable Connection (DCC)*, offers lower performance and they are never used to network more than two computers (though this would not be a problem here).

Rather than cable two computers directly, they may instead be connected indirectly through a central network fixture. This method requires two network cables, one connecting each computer to the fixture, which can be an Ethernet hub, a USB hub, a switch or a router. It is a general-purpose solution that accommodates any reasonable number of devices. Most cabled networks utilize Ethernet technology, direct or indirectly. However, connecting two computers wirelessly is also possible. Several wireless technologies, such as *Wi-Fi*, *Bluetooth* or *Infrared*, exist to support basic computer connection.

Thinking on the equipment and its capabilities, a wired Ethernet Local Area Network (LAN) is the only option, and the most appropriate solution in the context of this project. Wired connections offers better performance and reliability too.

The initial setup considered a crossover cable to directly connect the two computers. The common Ethernet cable used for connecting a computer with a router is a straight cable, with the wires connected to the same pins of the connectors at the two ends. Since the transmitter wires need to be connected to the receiver, and vice-versa, the crossing of wires is done internally inside the router. In the case a computer to computer connection, a crossover cable would be necessary. This is a specialized cable that is wired in reverse, meaning the crossing is done in the cable itself. In a crossover cable, the transmitter pins of one connector are connected to the receiver pins of the other, and vice-versa. The output from one computer connects to the input of the other. However, modern Ethernet interfaces of current computers use *Auto-Medium Dependent Interface Crossover (Auto-MDIX)* technology to sense whether crossover is required and do it internally if necessary. Thus, if both computers support Auto-MDIX, either a crossover cable or a default straight cable can be used. Otherwise, a crossover cable should be used.

Although physical interface is easier to set up, configuring the network settings on each computer for this type of connection may be a bit complicated, particularly when Internet connection sharing is intended. Internet connection sharing provides the ability for one computer to share its Internet connection with another computer. To do this, a computer with an Internet connection must be configured to function as an Internet gateway. Then, a second computer (or a network of computers) connects to the Internet indirectly via the first computer. In order to share an Internet connection, the gateway computer, the one that do the sharing, must have two network cards or ports. This is an additional requirement that cannot always be satisfied. Usually, the computers possess only one Ethernet network adapter, and they cannot be physically prepared to incorporate an additional PCI network card. Even though the equipment available allows for this flexibility, configuring the network using the gateway method (iptables) can be difficult and entails some complications. Thus, joining the two computers indirectly through a central router was preferred over this option. Connect two computers through a central infrastructure, such as a router, allows to share not only data and resources between the machines, but also an Internet connection far more easily (Figure 3.6).

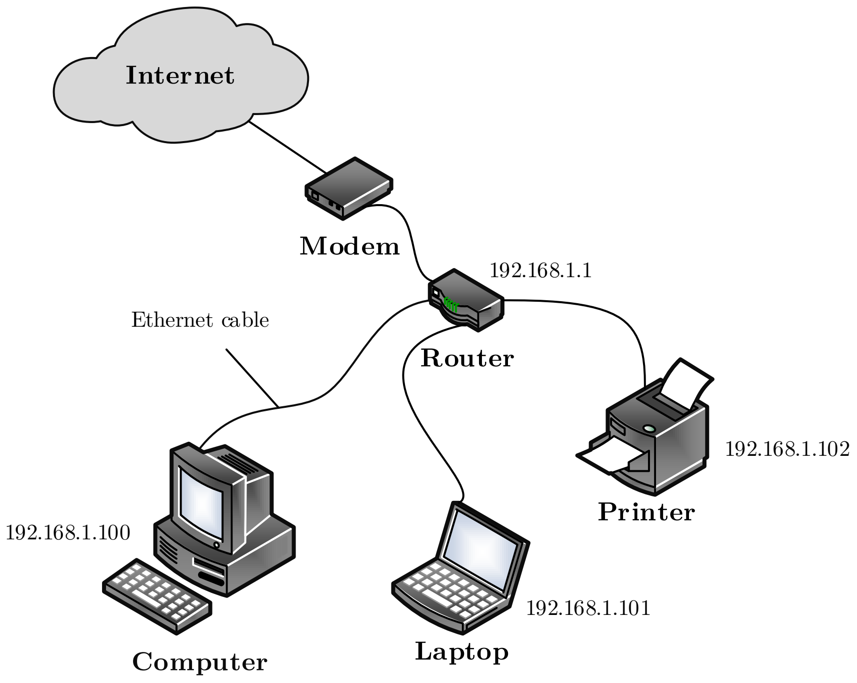


Figure 3.6: Example of a network using a router.

A router is a more sophisticated network device than either a switch or hub. It offers features like firewall support and DHCP server capability that hubs and switches do not. Basically, routers rely on the Internet Protocol to define a common and unified addressing system, used to forward data to other computers across the network. They can send information to any computer as long as it has an uniquely assigned IP address. The computer sending information over the Internet, or over the local area network, has to package that data into a packet. The data packet contains a header which includes the IP address of the destination computer. Routers read this address and then forward the packet in the direction of its destination.

Thus, the work station assembled for this project includes:

- two desktop computers.
- two PHANToM Omni haptic devices.
- two Firewire PCI express cards.
- one network router.

One of the computers will be running V-REP, while ROS modules will be running on both of them. Additionally, besides joining the local area network internally, the router will also join the LAN to the Wide-Area Network (WAN) of the Internet. The complete setup is schematically represented by Figure 3.7.

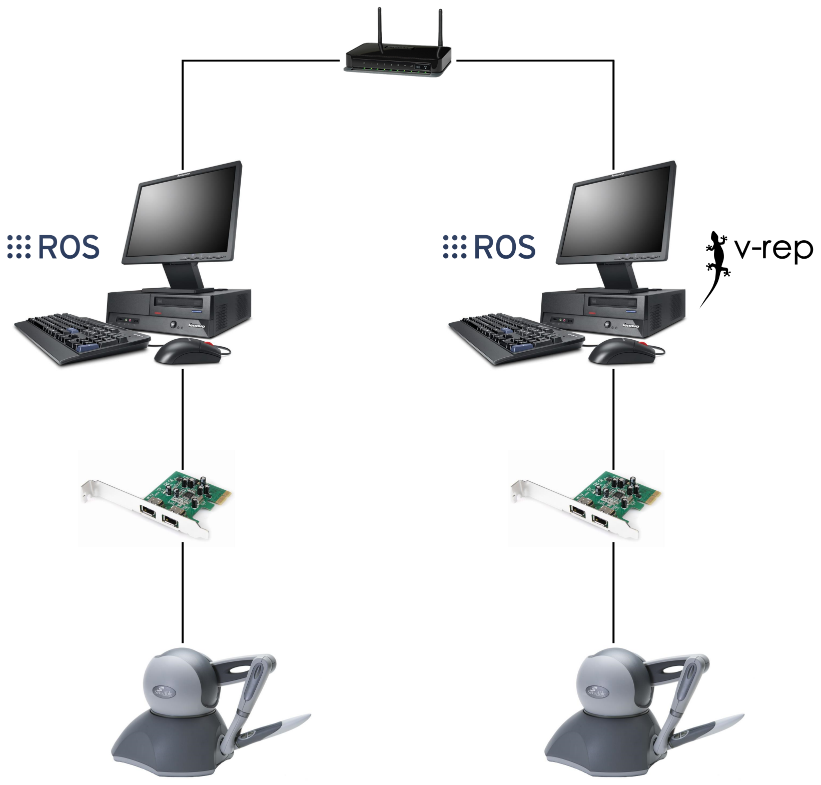


Figure 3.7: Experimental setup for cooperative haptics teleoperation [13].

The cable from Internet Service Provider (ISP) or modem is plugged into the Internet (WAN) port of the router. This connection typically uses an Ethernet (CAT5 or CAT6) cable. Each of the two computers is also connected to the router via a network cable. After the connection with the computers is established, the router have to be properly configured to assure the connectivity between the pair of machines.

Both computers are also equipped with a Firewire PCI express card that serves the communication with the joysticks. IEEE 1394a 6 pin cables are used to connect the devices to the computers. Details on how to successfully enable the connection and configure the haptic joysticks are described in Appendix B.

Name resolution

When a ROS node advertises a topic, is provides a *hostname:port* combination that other nodes will contact when they want to subscribe to that topic. It is thus crucial that all the machines can report a hostname addressable by other machines. One of the ways to ensure this is to configure the */etc/hosts* file on each computer. The hosts file tells each machine how to convert specific names into an IP address. Besides having the *localhost* name, each file must contain the IP address and the hostname of the other computer, in order to the machines to find each other. After this configuration the machines should be able to ping one another by name, and so they would be capable of contact nodes running on each other.

The name of each machine is now associated with the respective IP address, assigned by the router. If the connectivity between the machines cannot be checked, the computers are not on the same network, so that it needs to be reconfigured.

Same ROS master

Another requirement imposes that all nodes have to be configured to use the same master. This configuration can be done immediately before any node starts, defining, or initializing, a particular environment variable. *ROS_MASTER_URI* is a required setting that tells nodes where they can locate the master. It should be set to the XML-RPC URI of the master, for example: *export ROS_MASTER_URI=http://phua-lar-desktop:11311*, where "phua-lar-desktop" denotes the hostname of the machine where the master is launched. Using the "localhost" notation can lead to unintended behaviors with remotely launched nodes, as in this case.

This variable must be set on both machines, before any nodes starts running. Thus, it may be very useful to add this command to the *.bashrc* file, so that it will be executed automatically as a shell starts.

Considering that all the preceding conditions are fulfilled, nodes or modules running on different machines should be able to communicate with each other, without any concern.

Single ROS master

The ROS functionality in V-REP is enabled via a plugin, which is successfully loaded if the *roscore* was running at that time. For organization, it is assumed that the master will be running on the same machine as V-REP, although this will not affect the system's behavior.

In addition to dealing with the hardware restrictions, a solution like this is useful if a certain process consumes too much computational resources. With the exception of the joystick node running on the V-REP computer, the other nodes will be running on the second machine, sharing the necessary computational effort.

3.3.3 Package *humanoid_simulation*

The PHANToM module(s) and the V-REP simulator are not the only ROS nodes operating in this framework. In fact, they are part a ROS *package* developed for this work. Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. The *humanoid_simulation* package contains modules designed to handle the communications between hardware and software, and also to reproduce the control loop idealized for robot teleoperation. The modules are written in C++ language, and define general *classes* that can be ported to other platforms, or be easily recycled into new applications or algorithms. A *Doxygen*-based documentation was created to help the understanding of the code structure, the APIs used and the functions developed. How these modules operate and interact with each other, is presented next.

The *humanoid_simulation* package is basically composed of four modules, since V-REP is not considered to be part of this package: *phantom_control*, *robot_state*, *haptic_feedback*, and *record_data*. These modules interact with each other, and with V-REP, in a defined structure that involves control and force feedback, as represented in Figure 3.8.

The *robot_state* module is in charge of receiving data from the V-REP simulation, monitoring robot and simulation states. It receives the force values measured by the sensors and uses them to calculate the CoP. It also defines the robot's ground base support, when one or both feet are in contact with the ground. The CoP and support polygon positions are then transmitted to the *haptic_feedback* module. This module translates sensor information into force feedback according to the equations described in Chapter 4. Currently, the only sensory information that comes from the simulation is provided by the force sensors. However, accelerometers and gyrosensors can also be adapted to the model, providing inertial data. Vision and proximity sensors can be integrated too. Computed feedback forces are sent to the *phantom_control* module, which connects to the haptic device to reproduce them on the user. The *phantom_control* module connects to V-REP and joystick simultaneously, returning the device status to the simulator and updating the received forces, closing the control loop. The *record_data* module does not play an important role in the real-time teleoperation. Its function is to record the information transmitted between the other nodes, concerning further visualization and analyses.

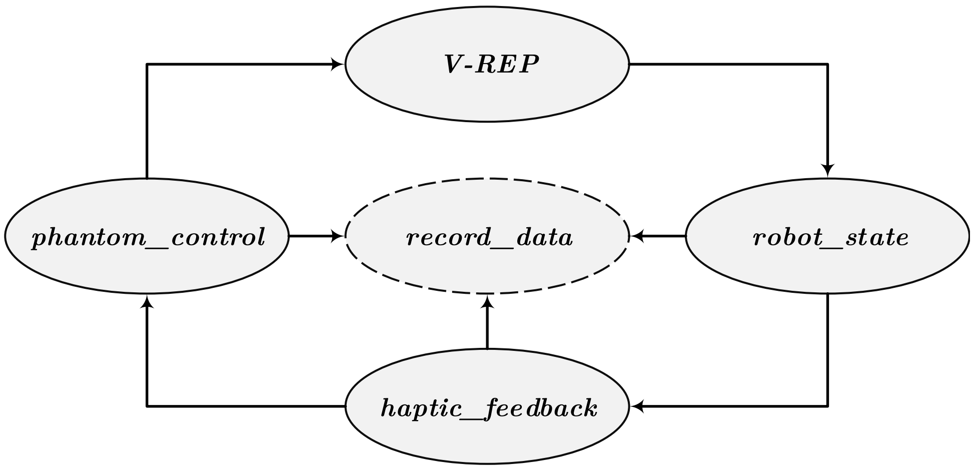


Figure 3.8: Interactions between the designed ROS modules.

Module *phantom_control*

As stated before, the PHANToM Omni is the primary interface for robot control. The *pahantom_control* application is able to perform joint-by-joint robot control and inverse kinematics control when combined with the joystick, or inverse kinematics control when alone. So, it uses both ROS and OpenHaptics. The feedback force generation depends on the correct operation of the remaining nodes, however their failure will not affect the commanding orders.

The module's structure follows the typical organization of an HDAPI program [24], but including the ROS functionality. The HDAPI consists of two main components: the *device* and the *scheduler*. The scheduler manages a high frequency thread for sending forces and retrieving state information from the device, the so-called *servo loop* thread. This is a high priority thread designed to exponentiate the low-level interaction between the libraries and the PHANToM hardware, which secures that force updates are sent at the desired 1000 Hz frequency. The scheduler interface allows the application to communicate effectively with the servo loop thread in a thread-safe manner.

The ***servo loop*** thread opens an infinite loop structure, where user functions can be assigned to run consecutively in an ordered, prioritized, and safe manner. These functions are conveniently called "callbacks", and allow the programmer to enter commands that will be executed within the servo loop thread. Device operations, meaning operations that are associated with getting and setting states, should exclusively be performed using scheduler callbacks. Direct making calls to getting state, starting and ending frames, or enabling and disabling capabilities within the application is not thread safe and will typically result in an error.

Scheduler calls can be either synchronous or asynchronous. Synchronous calls only return after they are completed. The application thread waits for a synchronous call before continuing. This type of callback is primarily used for getting the state of the scheduler for the application. With asynchronous functions, the program execution is not halted, allowing for the callbacks to run in loop at maximum frequency possible. They are often the best mechanism for managing the haptics loop. An asynchronous callback can persist in the scheduler to represent a haptics effect, which means that during each iteration, the callback applies the effect to the device.

For every scheduler tick, each callback is always executed. However, the execution order in the queue is defined by setting the priority (maximum, default or minimum) property, which determines what order they are run in the scheduler.

Before the device's loop starts, a series of routines take place that verify calling arguments, hardware state and communications. The general pattern of use for the HDPAI is to initialize the device, enable forces, create scheduler callbacks to define force effects, and start the scheduler. When the application is terminated, the device and scheduler are cleanup. If the joystick is detected by the PDD and correctly initiated, the HDAPI allows to create the hardware-dedicated C/C++ classes to manage and communicate with the device, as well as its dedicated thread (*servo loop*).

After the device and scheduler initialization, the module will also start a parallel *main loop*, where the ROS node will be running. The diagram in Figure 3.9 represents the application's main structure.

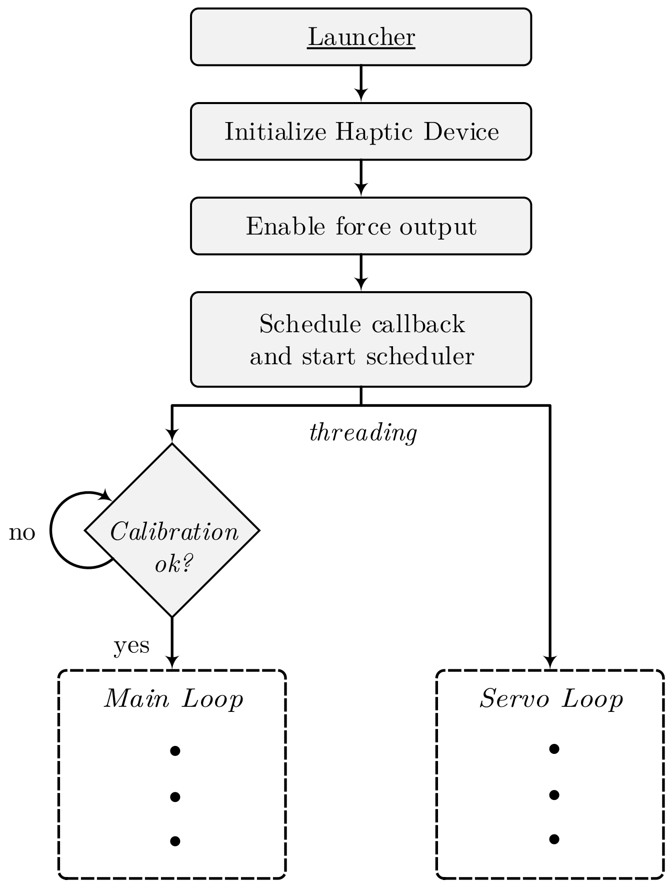


Figure 3.9: Simplified schematic diagram of *phantom_control* module, which runs in two separate threads, the *main loop*, and the *servo loop*.

The node starts by initializing the haptic device and enabling the force output property, for force rendering. Two asynchronous scheduler callbacks are created. The first is a maximum priority call, responsible for haptic device data extraction. The second runs at default priority and is responsible for rendering force vectors. A synchronous callback is also created to check the joystick internal calibration parameters. If it requited, the user must proceed to it. After callbacks declaration, the scheduler is started, and the threading happens. The operations carried out within the loops are shown graphically in Figure 3.10.

Servo loop

The *servo loop* includes several operations. Calling operations within a haptic frame ensures consistency for the data being used because device state remains the same within the frame. At the start of the frame, the device state is updated and stored for use in that frame. At the end of the frame, new state such as forces is written out to the device. Getting or setting state outside of a frame typically returns outdated data or results in an error.

Data collected from the joystick includes, the end-effector position, joint angles, button states, and motor temperatures. These information is stored into an exclusive dedicated data structure, which can be accessed by the main loop functions. Gathering motor temperature information allows for safety routines preparation that ensure the device's integrity. These are essential if the underlying safety mechanisms controlled by the device drivers fail.

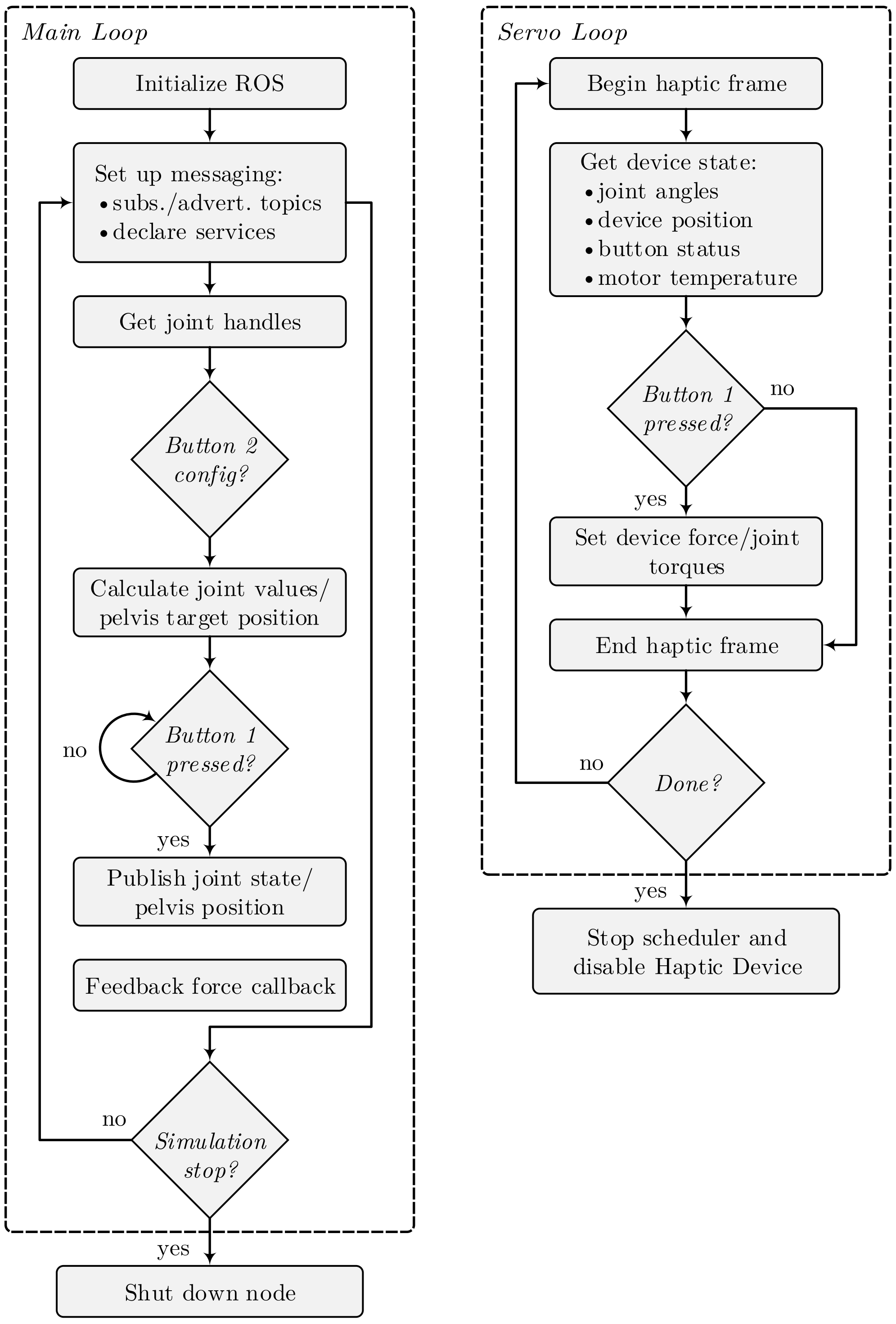


Figure 3.10: Simplified scheme of the *main loop* and *servo loop* operations.

Calibration

The main loop is started only if the device is properly calibrated. Calibration allows the device to maintain an accurate idea of its physical position, and should be done before any use. Typically it is carried out by the user, when he proceeds to device configuration and test (*PHANToMConfiguration* and *PHANToMTest*), however a further check is made. If the device is not calibrated, the callback takes actions to calibrate it. For certain devices, such as the PHANToM Omni, the calibration is successfully performed when the gimbal is placed into the device inkwell. This form of calibration is always performed after the servo loop start, since hardware evaluation is required. Because of this, the main loop always starts after the servo loop, but since main loop functions need joystick position and joint values, it is not a problem.

Main Loop

If the calibration routine has been successful, or no calibration is needed at all, the ROS node is initiated and ROS messaging and servicing are configured. The *phantom_control* node uses one of the many ROS services offered by V-REP to request for the joint handles. This function executes only once. Since these are scene's intrinsic, they remain unchanged. The remaining communications are based on publishing/subscribing protocol. Pelvis positions or joint values are sent to V-REP by publishing them to a topic that the simulator will subscribe. The same goes for the feedback force values computed on the *haptic_feedback* module.

The joystick buttons provide a very useful tool for teleoperation. Their state is used to define the commanding method characteristics, and to indicate whether or not the user desires to send his instructions to the model, and reproduce feedback forces. *Button 2* defines an asynchronous state machine that allows the operator to choose between four states/modes of control:

- inverse kinematics control;
- right leg joint-by-joint control;
- left leg joint-by-joint control;
- double leg joint-by-joint control.

This selection can be done online, without compromising the success of the simulation. Depending on the selected configuration, the module calculates the hip position or the joint values that will be sent to the robot. The inverse kinematics mode defines a three-dimensional point which represents the hip's target position as function of the current position retrieved by the joystick. The joint-by-joint mode defines a more complex data structure. It defines two vectors, whose size corresponds to the number of active joints of each leg. Each element of the vector contains three individual entries: the *joint handle*, the *joint control type*, and the *joint value*. The commanding methodology implements a position control to drive the robot joints, thus it is their target position that must be specified. The kinematic correspondence between the joystick angles and the robot joints is described in Chapter 4.

Joint and position calculations are continuously performed, however the output is only sent to V-REP when the *button 1* is pressed. This allows for great flexibility and control of the user's own actions in simulation, serving also as safety measure. It avoids undesired actions that may arise in case of uncomfortable or unknown operation scenarios. Moreover, if the button is not pressed, the force feedback is not generated, which prevents the device from exerting excessive or undefined forces when the simulation breaks up, i.e., when the robot falls.

Simultaneously, the *phantom_control* node subscribes to the topic where the *haptic_feedback* node publishes a message containing the force, or torque, vector that should be rendered by the device. When a message is received, it starts a callback that updates an array structure with the required force. Likewise, if the *button 1* is pressed, the haptic device will apply the force on the user, otherwise, it is not reproduced. The force update is done within the scheduler callback defined for this purpose.

The *main loop* is dependent on the ROS state. This node also subscribes to a simulation state topic that indicates if the simulation is playing, paused, or stopped. If in the previous iteration the simulation was playing, but its current state indicates a stop, a shutdown signal is given to ROS and the node terminates - main loop is over. When ROS stops *spinning*, the servo loop terminates too, and a series of clean-up routines along with thread closing are performed. Before application exit, the scheduler should be stopped and all the scheduler operations terminated. Finally the device should be disable.

Module *robot_state*

The *robot_state* module's main purpose is to retrieve information about the current state of the robot, and to define relevant indicators of the system's kinematics and dynamics.

A general representation of the module's structure is shown in Figure 3.11. This module basically defines a C++ class responsible for monitoring and register the main variables involved in the robot simulation. Available data from the simulation is used to determine the CoP position and define the ground base support of the robot. Although it was particularly defined to interface with V-REP, its general structure can be adapted to other applications.

After the *launcher* initiates the ROS node, the *Humanoid* class is instantiated. The ROS messaging configuration is done on the class constructor, when the new object is created. This node subscribes to several different topics, in which the V-REP publishes the robot state. State information is published at each simulations pass, so that the incoming data is never repeated or missing. The class defines five primary callbacks as public members, which update the necessary information for CoP and support polygon calculations, and two secondary callbacks for simulation state evaluation and additional data logging.

The center of pressure is one of the most common ground reference points, and has proved to be a very useful indicator of postural stability in legged robots. In fact, it has been one of the most studied indicators in the PHUA project. This reference point is of particular importance in this work, given that the algorithm for force feedback generation relies on its positioning. Its location is mathematically defined and can be analytically calculated, but in practice, it is estimated via pressure/force sensors. The position of the CoP can be calculated as in Equation (3.1), where $\mathbf{r}_i$ is the position vector of the *ith* sensor in the feet, and f_i is the force component normal to the ground, at contact point i.

$$\boldsymbol{COP} = \frac{\sum\limits_i \boldsymbol{r}_i \cdot f_i}{\sum\limits_i f_i} \tag{3.1}$$

Robot coordinate frame

Determining CoP, or other reference points, demands the definition of a local coordinate system on the robot. The dependence on a absolute coordinate frame should never be considered, specially when gait generation is desired. In fact, when dealing with humanoid robots it does not make any sense at all, since the robot can be at multiple positions and configurations over time. The first choice for the local frame definition would be the robot pelvis, or hip, which is the natural reference for biped locomotion. Legs, more precisely feet, and even torso movements are described on this reference point. However, this work puts a special value on the CoP calculation, and how it influences the balance of a robot. The exact location of this point is required to establish the balance conditions used to control the robot, and generate the force feedback on the user, as later described. Thus, it becomes clear the importance of defining a coordinate system on the feet, from which the CoP position can be determined unambiguously.

V-REP retrieves the absolute sensor and *dummy* positions, (i.e. with respect to the world frame), so that it is necessary to transform these coordinates to the feet system. ROS provides a very interesting package that lets the user keep track of multiple coordinate frames over time - *tf* [27]. This is extremely useful in the robotics field, since a robotic system typically has many 3D coordinate frames that change over time. In this case, the world, the left foot, and the right foot frames.

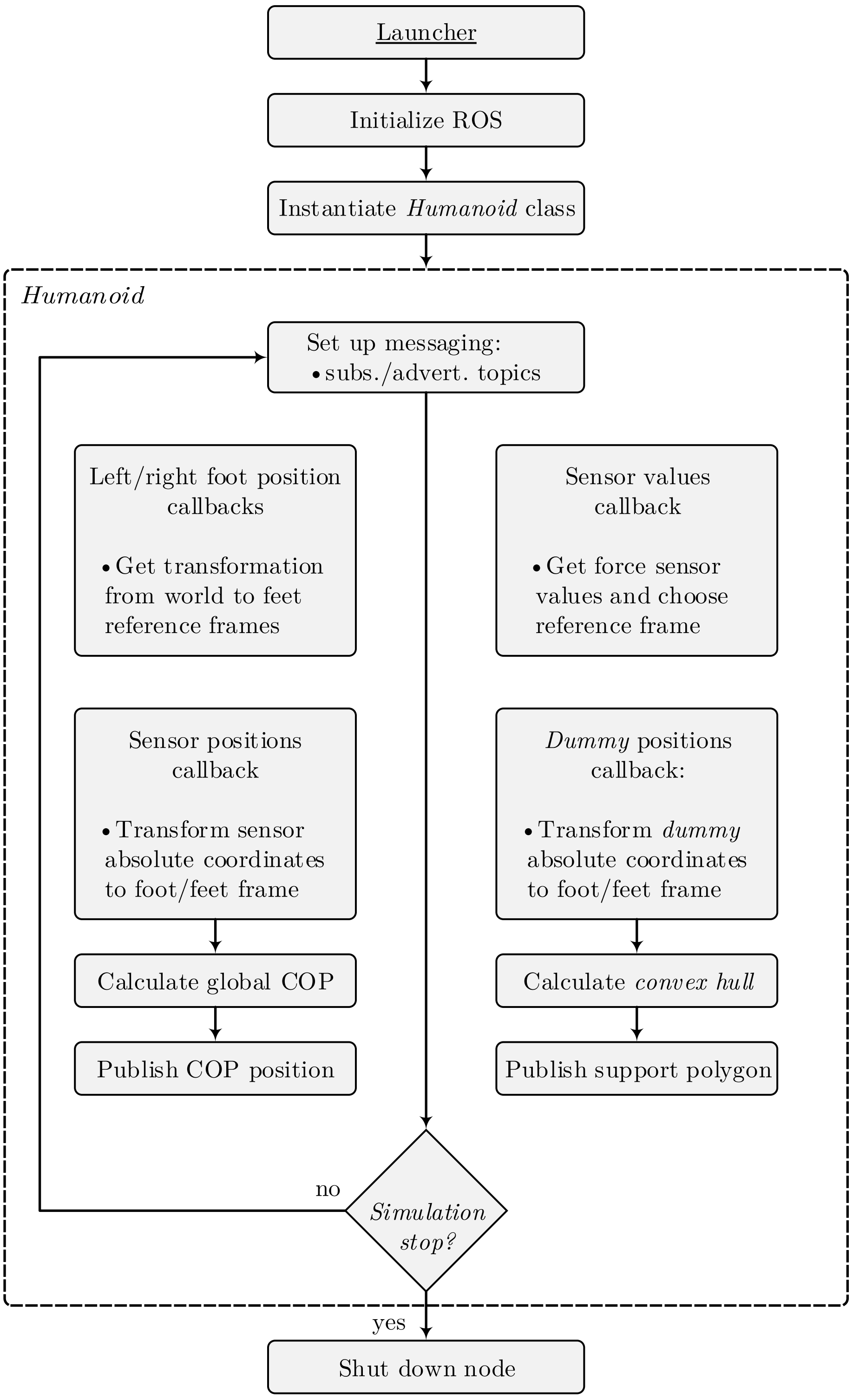

Launcher
Initialize ROS
Instantiate *Humanoid* class
Humanoid
Set up messaging:
• subs./advert. topics
Left/right foot position callbacks
• Get transformation from world to feet reference frames
Sensor values callback
• Get force sensor values and choose reference frame
Sensor positions callback
• Transform sensor absolute coordinates to foot/feet frame
Dummy positions callback:
• Transform *dummy* absolute coordinates to foot/feet frame
Calculate global COP
Calculate *convex hull*
Publish COP position
Publish support polygon
no
Simulation stop?
yes
Shut down node

Figure 3.11: Simplified schematic diagram of the *robot_state* module.

The *tf* maintains the relationship between coordinate frames in a tree structure buffered in time, allowing the user to transform points, or vectors, between any two coordinate frames at any desired point in time. The *robot_state* module subscribes to topics containing information about the feet position and orientation, in the absolute frame, and uses it to declare and broadcast a transformation that describes their position in the world frame. The local frame can be defined either by the left foot or the right foot position, when only one foot is in contact with the ground, or by the mid-pose of both feet, when both stay on the floor. In practice, this is achieved by evaluating the state of the force sensors on each foot. If all the 8 sensors send reliable information to the node, meaning force data available (contact with the ground), and "unbroken" state, the reference frame is defined on the mid-point of the feet. Otherwise, if any of the sensors fails, the reference frame coincides with the location of the second foot, considering its 4 sensors are measuring forces correctly.

The robot frame is computed at each time step, provided the information by the simulator. Broadcasting the transformation sends out the relative pose of the coordinate frame to the rest of the system. Since *tf* can operate in a distributed system, all the information about the coordinate frames of a robot is available to all ROS components on any computer in the system. The callbacks responsible for getting the sensor and *dummy* positions from V-REP are simultaneously listening for the calculated transformation, in order to transform the absolute coordinates published by V-REP to the robot frame.

After converting the force sensor coordinates into the local frame, the ROS callback calls a class member that calculates the position of the general CoP, based on the force sensor local coordinates and on the force values, according to Equation (3.1). In similar way, the *dummy* positions callback transform their position to the local system, so that they can be used to compute the feet convex-hull in the same reference frame. This calculation is based on the Andrew's monotone chain convex-hull algorithm [28], which constructs the convex-hull of a set of 2-dimensional points in $O(n \log n)$ time.

In the third event the CoP location and the points defining the the support-base are published to defined topics, which the *hapti_feedback* module will subscribe to compute the force rendered on the user. Along with this data, the node also publishes the total ground reaction force (the sum of all force sensor values), and the robot hip's position for data logging purposes. References to this data are crucial to describe the robot's behavior when teleoperated in inverse kinematics mode.

The application's loop will continue until the simulation stops. By subscribing to the */vrep/info* topic, the module can access information about the simulation's time and state. This information is used not only for state evaluation, but also to attach a time stamp to the published data. This time stamp, which corresponds to the simulation time, allows comparison of results and assessment of simulation changes over time.

Module *haptic_feedback*

The idea of the *haptic_feedback* module is to calculate the feedback force sent to the joystick, based on the CoP location, and on its distance to the support polygon boundary. Figure 3.12 represents the basic structure of the *haptic_feedback* module.

This module operates in a similar way to the *robot_state*, defining a C++ class responsible for subscribing to CoP and support polygon information, published by the state module. After receiving this data, the application evaluates if the CoP is within the support polygon, or not [28]. If the CoP is within the limits, the minimum distance to the polygon edges is computed, applying an algorithm that calculates the distance from a point to a segment [28]. This is repeated to all the segments composing the polygon, and the shortest distance is selected. The feedback force is calculated next, using the the algorithms presented in Chapter 4, and then published. If the CoP is outside the support base, the force rendering calculations do not proceed, since this condition already indicates the robot had lost its balance. Again, when the simulation stops, the node is shutted down.

Launch file

Since all ROS modules are designed to work together, a *launch* file is used to initiate the four nodes of this package, at the same time. This is another utility of great use provided by ROS, which avoids the need to start each node individually, every time the user runs a new simulation.

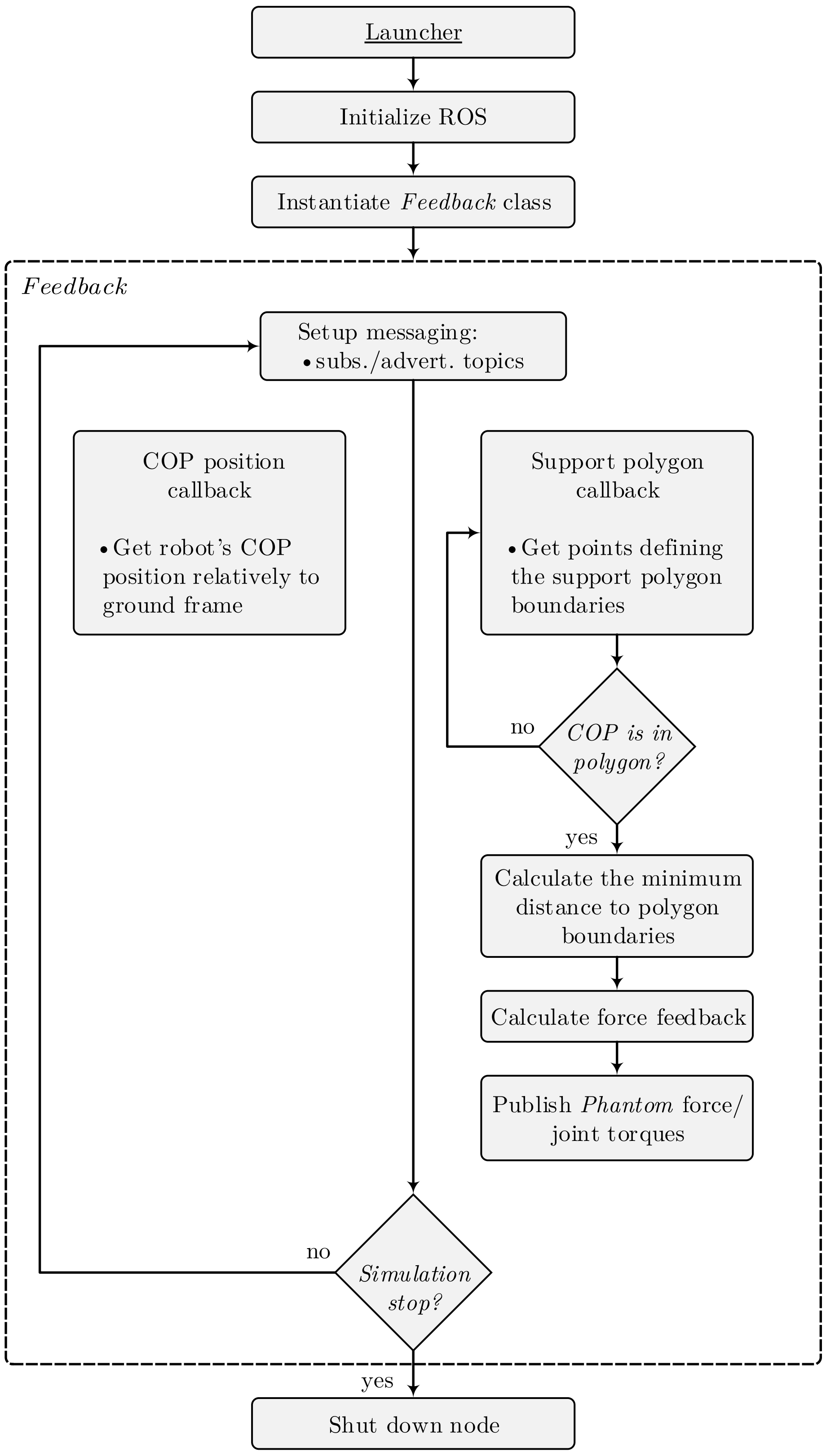

Launcher
Initialize ROS
Instantiate *Feedback* class
Feedback
Setup messaging:
• subs./advert. topics
COP position callback
• Get robot's COP position relatively to ground frame
Support polygon callback
• Get points defining the support polygon boundaries
no
COP is in polygon?
yes
Calculate the minimum distance to polygon boundaries
Calculate force feedback
Publish *Phantom* force/ joint torques
no
Simulation stop?
yes
Shut down node

Figure 3.12: Simplified schematic diagram of the *haptic_feedback* module.

Chapter 4

Robot Kinematics and Interactive Command

This chapter presents a literature review on some of the existing models and methodologies explored to define balance, and describe dynamics in legged robots. Then, the PHUA approach is introduced, emphasizing the renewed strategy developed with this work. Two control methodologies for robot command in teleoperation are detailed. The inverse kinematics methodology actuates the robot joints indirectly, in response to hip's location. With the torque methodology, the robot joints are directly actuated, so that the user is able to control each one individually. Complementary to the user commands, a feedback force is synthesized to transmit to the operator the current state of the robot.

4.1 Dynamic Stability and Control Methodologies

A human posture is said to be balanced if the ground projection of its center of mass (CoM) is within the convex-hull of the foot-support area. Otherwise, an uncompensated moment on the foot causes it to rotate about the polygon boundary, leading to fall [29]. Balance control and walking pattern generation for humanoid robots are widely recognized as a difficult problem due to the high dimensionality of their action space, since a large number of degrees of freedom and non-linearities inherent to any kinematic chain are involved [30]. Because of its importance, several decades of research have been invested and numerous techniques for biped balance control based on different approaches have been proposed.

4.1.1 Existing Models

Zero-Moment Point

A first step was made by Vukobratovic and Juricic by introducing the concept of what was later known as the Zero-Moment Point (ZMP) criterion [31]. It simplifies the high dimensional problem by reducing the influence of all forces acting on the mechanism to one single force [32]. Considering the locomotion mechanism in the single-support phase, statically unstable, the physical interaction between foot and ground results in a ground reaction force, with opposite sign, at the exactly same point where the resultant force acts. The two-dimensional point on the ground, called ZMP, where this resulting force acts, can be seen as the overall indicator of the system behavior. The foot relies on the support and the only contact with the environment is via the friction force and the vertical force of the ground reaction. Dynamic balance, or as also said, dynamic stability, is achieved by ensuring the foot's whole area, and not only the edge, is in contact with the ground. In other words, if the ZMP lies within the support polygon of the robot, the state of the robot is called dynamically stable. This notion is applicable for both single and multi-leg ground support phases. The support polygon, or ground base support, is the actual foot-ground contact surface when only one foot is in contact with the floor, or the convex hull of the two or more discrete contact surfaces when two or more "feet" are in contact with the ground. It is the area covered by the feet.

Given some limitations of the ZMP definition, other ground reference points have been proposed, for example, the Foot Rotation Indicator (FRI) or the Centroidal Moment Pivot (CMP) [33]. Ground reference points are very useful in motion identification and control biomechanics and legged robotics. The Centroidal Moment Pivot is another reference point, recently introduced in literature [34], with similar results to those presented by the ZMP methodology.

Centroidal Moment Pivot

A legged control system must continually modulate moments about the center of mass to control spin angular momentum and whole body angular excursions. Throughout a walking gait, the moment about the center of mass must be continually adjusted to prevent the body angular excursions from becoming appreciably large, causing loss of balance. The introduction the CMP concept aims to address spin angular momentum and the moment about the center of mass in connection with various postural balance strategies.

The Centroidal Moment Pivot is defined as the point where a line parallel to the ground reaction force, passing through the body's center of mass, intersects with the external contact surface [33]. This condition can be expressed mathematically, and the CMP location can be written in terms of the center of mass location and the ground reaction force.

The term CMP is used by the previous definition's authors, however other terms are used to name the specified quantity. In other words, the CMP is the point where the ground reaction force would have to act to keep the horizontal component of the whole body angular momentum constant (see Figure 4.1).

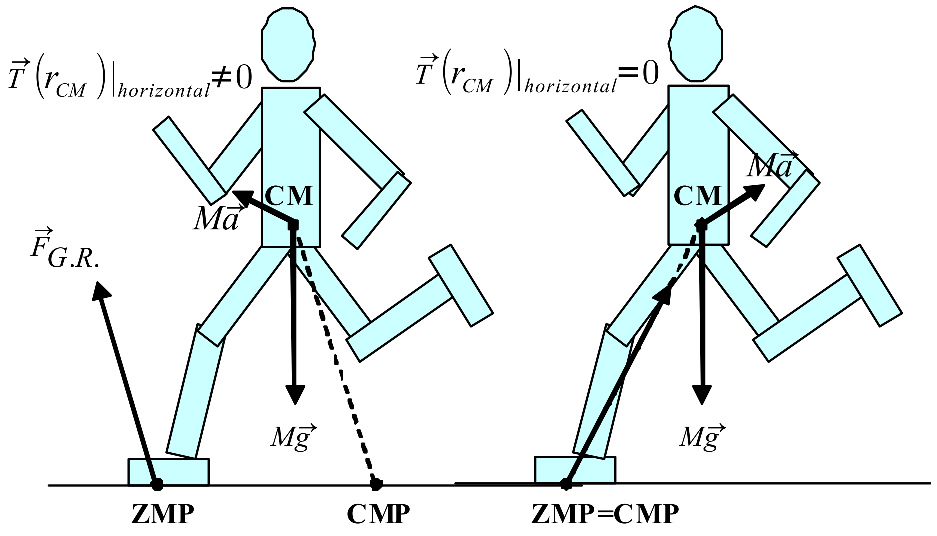

$\vec{T}(r_{CM})|_{horizontal} \neq 0$
$\vec{T}(r_{CM})|_{horizontal} = 0$
CM
CM
$M\vec{a}$
$M\vec{a}$
$\vec{F}_{G.R.}$
$M\vec{g}$
$M\vec{g}$
ZMP
CMP
ZMP=CMP

Figure 4.1: Centroidal Moment Pivot (CMP). When the moment about the center of mass (here designated by CM) is zero, as shown in the right figure, the CMP coincides with the ZMP. However, when the CM moment is non-zero (figure on the left), the extent of separation beteween the CMP and ZMP is equal to the magnitude of the horizontal component of moment about the CM, divided by the normal component of the ground reaction force [33].

When the CMP coincides with the ZMP, the ground reaction force passes directly through the center of mass of the body and the rotational equilibrium condition is satisfied (zero moment about the center of mass). Hence, when the CMP differs from the ZMP there is a non-zero moment about CM, causing variations in spin angular momentum. Applicable for both single and double ground support phases, the CMP gives valuable information about whole body rotational dynamics when supported by the ZMP location (excluding body rotations about the vertical axis).

Although the ZMP and CMP can coincide, these points cannot be considered equivalent. However, some results published in the above-mentioned paper [33], show the CMP trajectory during the walking gait cycle closely tracks the ZMP trajectory during both single and double support phases. These tests were based on the hypothesis, later verified, that the CMP trajectory will never leave the ground support throughout the entire human walking gait cycle. This can be assumed, since recent biomechanical investigations (cited in the same article) have determined that total spin angular momentum, or the body's angular momentum about the CM, remains small through the cycle.

Foot Rotation Indicator

In addition to these two standard reference points, the foot rotation can also provide an indication of postural instability. Although the position of the ground projection of the center of mass is sufficient to determine the occurrence of foot rotation in a stationary robot, it is not for a robot in motion. Instead, the location of the FRI point, that indicates the existence of an unbalanced torque on the foot, will be sufficient. The FRI point is a point on the ground surface, within or outside the support polygon, where the resultant ground-reaction force would have to act to keep the foot stationary [29]. If the FRI point remains within the foot-support area, there is no foot rotation and consequently no unbalanced moment or instability. An unbalanced torque and imminent foot rotation will emerge only if this point is out of the foot limits, or, more precisely, the FRI point will leave the support polygon whenever there is an unbalanced torque on the foot. And the further this point is from the support polygon boundary, the larger the torque's value is. Despite its utility, this reference point can only provide information on stance-foot angular accelerations when only one foot is on the ground.

To formally describe the FRI point, the biped robot is treated as a general n-segment extended rigid-body kinematic chain, and the robot is considered to be in the single-support of the locomotion cycle, when only one foot is in contact with the ground, as represented in Figure 4.2.

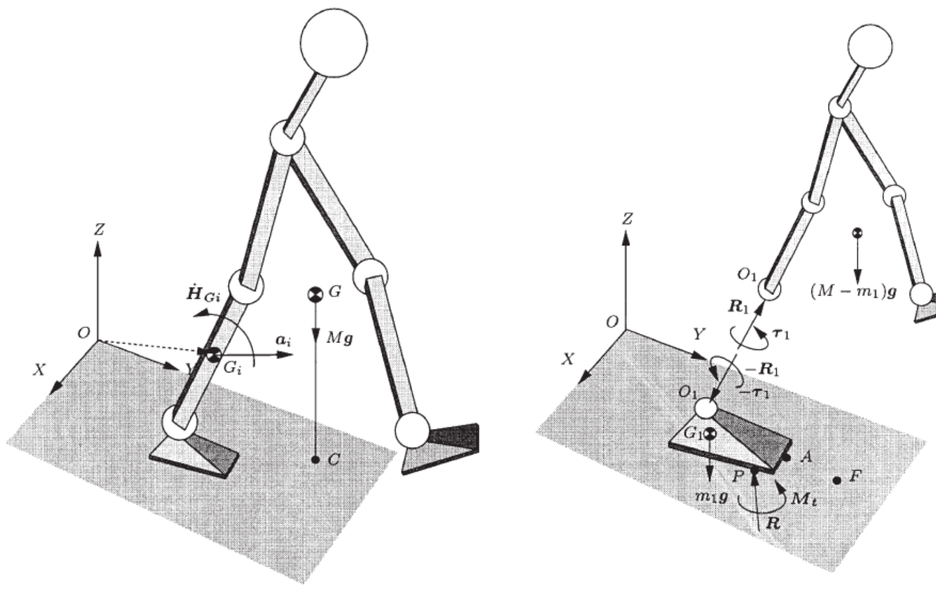


Figure 4.2: The sketch of a 3-D extended right-body biped robot (left), and a view with its support foot artificially disconnected from the shrank to show the intervening forces (right). The CoP, CoM, and the FRI are denoted by P, C, and F, respectively [29].

The external forces acting on the system are the resultant ground-reaction force/torques, R and M, acting at the center of pressure, denoted by P, and gravity. As the only robot segment interacting with the ground is the foot, it is subjected to joint forces and the external forces already described, so the dynamics of the rest of the body can be completely represented by the ankle force/torque R_1 and

τ_1. The equations show the FRI point location depends on some parameters: the mass of the foot, the location of its center of mass and the gravity acceleration; and for all the other robot elements: the mass, the CoM location, the CoM linear acceleration, and the angular momentum about CoM.

This method indicates a stability margin that may be quantified by the minimum distance of the support-polygon from the current location of FRI point, when it is within those limits, and a measure of instability when the FRI point is outside. In summary, an imminent foot rotation will be indicated by a motion of the FRI point toward the support polygon boundary.

Unlike the ZMP, which may not leave the support area, the FRI point may leave this area. In fact, the position of the FRI point outside the footprint indicates the direction of the impending rotation and the magnitude of rotational moment acting on the foot, as already mentioned.

This ground reference point proves to be very useful in the anticipation of imbalance scenarios. However, some studies [33] find that the mean separation distance between the FRI and ZMP during the period of single support is significant, and thus conclude that the FRI point is not an adequate measure of the foot rotational acceleration. Besides, the FRI point methodology is only suitable for the single-support phase of locomotion.

Inverted Pendulum Model

In alternative to ground reference points, some other approaches have been proposed that are also based on a reduced model of the robot. For example, the Inverted Pendulum Model. It considers an ideal biped model composed of a single rigid body with a mass m and moment of inertia J, and two massless legs of variable length, representing the function of the knee. Besides, the motion of the model is constrained to a sagittal plane which is defined by the vertical axis and the axis of walking direction, and the robot is considered to be in the single leg support phase [35]. This model describes the whole robot by a linear inverted pendulum, reducing the complex walking dynamics to a linear differential equation.

Later, some extensions of this model have also been studied, such as the Three-Dimensional Inverted Pendulum Model and the Reaction Mass Pendulum. The Three-Dimensional Linear Inverted Pendulum (3D-LIPM) is derived from a general three-dimensional inverted pendulum whose motion is constrained to move along and arbitrarily defined plane. It allows a separate controller design for the sagittal and the lateral motion, simplifying a walking pattern generation [36]. For a robot supporting its body on one leg, its dominant dynamics can be represent by a single inverted pendulum which connects the supporting foot and the center of mass of the whole robot. Equations describing the dynamics of the center of mass along the plane can be derived from the motion equations of a 3D inverted pendulum by applying some constraints.

Although the original dynamics are nonlinear, the resulting equations from this model are independent linear equations. The only parameter which governs those dynamics is the z intersection of the constraint plane and the inclination of the walking plane never affects the horizontal motion.

Reaction Mass Pendulum

A significant number of conceptually simple but behavior-rich "inverted pendulum" humanoid models have greatly enhanced the understanding and analytical insight of humanoid dynamics. However, these models do not take into consideration the robot's momentum properties. The Reaction Mass Pendulum (RMP) incorporates the centroidal moment of inertia of the humanoid robot. The centroidal moment of inertia is a property of the distributed masses of the robot limbs (arms, leg, etc) away from the CoM, which directly contributes to the body's angular momentum and its rate of change, and consequently to balance maintenance.

As shown in Figure 4.3, an RMP consists of a "leg" that joins the CoP and the CoM, and an ellipsoidal "body" - the abstracted reaction mass - that characterizes the generalized inertia of the entire robot projected at the CoM [37].

As the robot moves, the shape, size, and orientation of the ellipsoid changes, in function of all limb movements, thus reflecting the entire system's behavior. In such conditions, the CoP and CoM are continuously in movement, and the 3D reaction mass has continuously variable inertia.

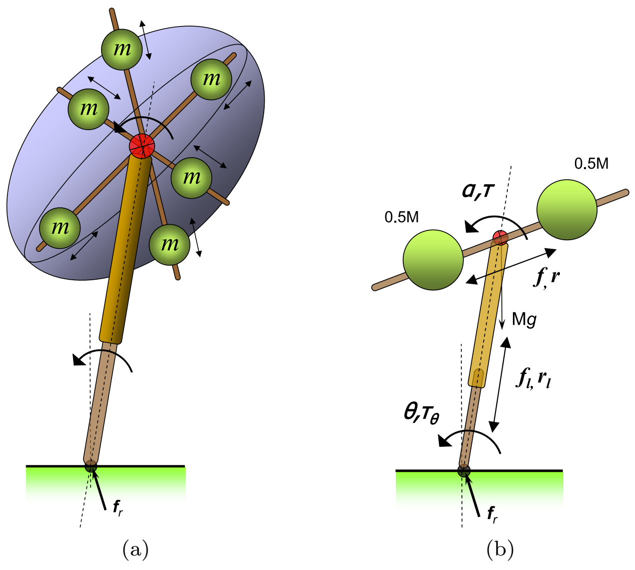


Figure 4.3: (a) Conceptual mechanical realization of the 3D RMP. The ellipsoid can be reduced to three pairs of equal point masses at different radial distances that are radially actuated to slide on their linear tracks. The overall frame consisting of the three pairs of linear tracks form the skeleton which can be actuated in three rotational degrees of freedom. (b) 2D Reaction wheel Pendulum Model. The distance between the two point masses is $2r$ [37].

Any given configuration of the robot can be modeled by three pairs of point masses linearly actuated along the three principal orthogonal directions of the ellipsoid. The masses of each pair are always equidistant from the ellipsoid center. The CoM of the ellipsoid is therefore always fixed at its center. It is the radial movement of the point masses, moving according system configuration, which affects the shape and size of the ellipsoid.

The RMP is mainly an analysis tool of the kinematics and dynamics of a general humanoid robot, and thus, control laws still need to be formulated and applied. Actually, and despite all these reduced models usefulness, at the point of implementation one has to find control schemes which map the strategy back into the full dynamic model. Hence, they generally have difficulties in dealing with external perturbations, since they represent an unexpected change in the dynamics of the robot.

4.1.2 PHUA Approach

Although the basic dynamics of balance are currently understood, robust and general controllers that can deal with non-leveled foot support, or unexpected and unknown external disturbances such as a moving support have not yet emerged, even though quite complex controllers and robot models have been studied.

Nature has developed methods for controlling the movements of organisms with many degrees of freedom which differ strongly from existing approaches for balance control in humanoid robots. Biological organisms employ kinematic synergies that simultaneously engage many joints, and which are apparently designed in such a way that superposition is approximately linear. They reduce the high dimensionality of their action space by generating movements there are combinations of simultaneous movements of many joints, simpler movements, referred as kinematic synergies by the above authors [38]. It is shown that this control strategy can in principle be applied to humanoid robots, too. And in contrast to existing approaches, this control strategy reduces the need to carry out complex computations in real time, and it does not require knowledge of a dynamic model of the robot. Therefore, it can handle unexpected changes in the dynamics of a humanoid, for example external forces.

This a particularly interesting solution that contrasts with the existing models for robot balance and motion and strategies based on pre-programmed routines. In a similar way, robot learning from demonstration algorithms gives the possibility to have a robotic system with learning capabilities that will allow it to adapt to new circumstances and environments.

The strategy here presented is an extension of the methods proposed by the PHUA project [14, 15], in which this work is inserted. This project proposes an approach for kinesthetic teaching in which the user interactively demonstrates a specific motion task, while feeling the dynamics of the system to be controlled via a haptic interface, hence the expression tele-kinesthetic [1]. Figure 4.4 shows an inexperienced operator interacting with a virtualized object while teleoperating the PHUA robot, in an attempt to draw simple lines on a transparent board with a color marker attached to the arm's end-effector. The board plane location was not known beforehand, therefore, the user was forced to determine it by inputting three points. This plane was then generated in the device's workspace as a planar force field.

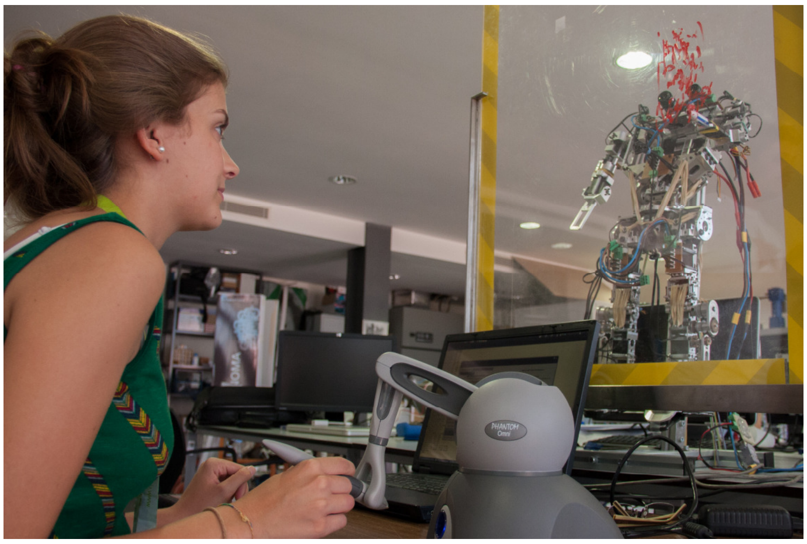


Figure 4.4: Teleoperation of the robot arm with force feedback [1].

In balance tasks, the force feedback on the operator can be generated by different sensory data (inertial, center of pressure, joint angles, etc). The robot's stability, or instability, is deduced from this data, and felt by the operator through this force feedback.

Recorded information about the robot's state (from joint state, CoP position, ground reaction forces to joystick kinematics and applied forces, etc) during many and widely varied experiences, can later be used by a learning from demonstration algorithm. Based on the outcome, the robot will know how to perform such balance and motion operations without the operator's control.

The compelling about this strategy, is that the user can train the robot for as many situations as he remembers, even expose it to non-recommended circumstances, thus teaching the robot to avoid them. Moreover, an understanding of the complex dynamic model of the robot can be discarded, since it is the operator who actually commands the robot in real-time. At the present stage of development, the joystick's position is translated into robot hip's movement. Consequently, the calculations involved are confined to inverse and differential kinematics, which describe the hip's position in terms of joint angles, sent to the robot. The control methodology is quite simple, and complex joint planning algorithms are not necessary.

Currently, the information received from the PHUA platform consists solely of the joint state, ground reaction force, CoP position, and some inertial data. At any given state, the feedback generation depends only on this little, however valuable, information.

Current Control Methodology

Most of the work previously done has consisted of teleoperating the real-robot using the haptic device, while using the general CoP to generate a force feedback for user interaction. This force is calculated based on the difference between the CoP in the robot home position (lower limbs "stretched") and the current CoP. In these demonstrations, the robot feet are considered to be kept in constant contact with the ground, and at a constant distance from each other. The movements of the robot are controlled by the haptic device state, more precisely, its position. Here, the hip position is driven by the location of the joystick tip in Cartesian space. Additionally, because only a single axis of the joystick is monitored at a time, the robot movements are constrained to one plane, despite some three-dimensional movement attempts. This is a very simplistic and unrealistic approach, however the only one possible, due to the shortage of degrees of freedom available on the joystick, when compared to those available on the robot.

Proposed Control Methodology

This work presents a renewed strategy to control the PHUA robot, based on the configuration already explained. The double joystick configuration allows the user to control each leg individually, greatly enlarging the sphere of teleoperation possibilities. Different scenarios can be experienced, where the user explores alternative configurations for the robot.

Apart from its normal working position, each joystick can be easily viewed as a robotic leg, containing the primary degrees-of-freedom of a human limb. As an alternative to the hip's inverse kinematics, a Joint space control can thus be implemented. With this strategy, the robot joints are individually controlled according to PHANToM's configuration. For a given joystick configuration, there is an equal leg configuration.

Besides promoting a wide variety of robot positions, this methodology is impressively intuitive. Moving the joysticks as if they were real robot legs is the ultimate teaching process.

To implement this strategy, new command methodologies have to be developed. A position command for the presented approach is defined in Section 4.2. Generalized force feedback equations are also developed, which consider this type of control for several different robot configurations.

4.2 Haptic Control

4.2.1 Kinematic Correspondence

After defining the V-REP communication strategy, in Chapter 3, methodologies for teleoperation are defined for inverse kinematics, and torque/force modes. Despite being tested in the simulator, the results of applying such methodologies reveal an underlying consistency with the reality. This shows, in a first stage, that it is possible and relevant the development of an infrastructure to operate the real robot, based on this approach. Currently, the application of such methodologies is restricted to the V-REP realm. Thus, the term "robot" is denoting the V-REP model of PHUA, not the real robotic platform.

Two actuation modes are defined: *inverse kinematics mode* (in Cartesian space) and *torque mode* (in Joint space). If in IK mode, the user will control the hip position in the Cartesian space of V-REP. With this mode there is the intention of reproducing the experiments conducted so far, in order to validate the model and test its accuracy. Good results can prove the relevance of the model, and the project itself. When operating in torque mode, each joint is individually actuated as function of its joystick pair. This is the methodology purposed to throw PHUA one step beyond.

Both joint operation types implement a position control. When a joint motor is enabled and the position control is enabled, there exists a control loop where the PID parameters will try to drive the joint to the desired target position.

The control is done by adjusting the joint velocity (regular position control), as described by Equations (4.1) and (4.2).

$$velocityToApply = (K_p e_i + K_i \sum e + K_d(e_i - e_{i-1}))/dynamicsTimeStep \tag{4.1}$$

$$maxTorqueToApply = maxTorque \tag{4.2}$$

Joints define their own velocity planning, according to the PID parameters, the dynamics calculation step, the cumulative error for the controller parameters e, and the error associated with each control iteration e_i. The error value is given by the difference between the target position and the current position, of each joint (with revolute cyclic joints, the shortest cyclic distance is taken). The maximum torque/force that a joint will be able to exert is defined by the user, as well as its velocity upper limit.

When a joint is in inverse kinematics mode, it can be operated in a hybrid fashion: hybrid operation allows the joint to operate in a regular way, but additionally, just before dynamic calculations, the current joint position will be copied to the target position, and then, during dynamics calculations, the joint will be handled as a motor in position control. By selecting this option - hybrid operation - the control loop is automatically enabled. This feature allows for instance to control the leg of a humanoid robot by simply specifying the desired foot position. The corresponding calculated joint positions will then be applied as position control values for the leg for the leg dynamic motion. This is the principle behind the inverse kinematics mode of teleoperation. The torque/force mode uses the same control methodology but the joints have to be motor enabled and the PID control parameters have to be set.

The position command defined in this work is a closed loop between the V-REP and the PHANToM device, serving as the intermediary between the user and the virtual world. At the same rate the simulation receives the information from the joystick and updates its state, it will also send information about the robot current state to other ROS modules. These will generate a force vector to be reproduced by the PHANToM device, interactively, as the user controls the robot. Thus closing the control loop, represented in Figure 4.5

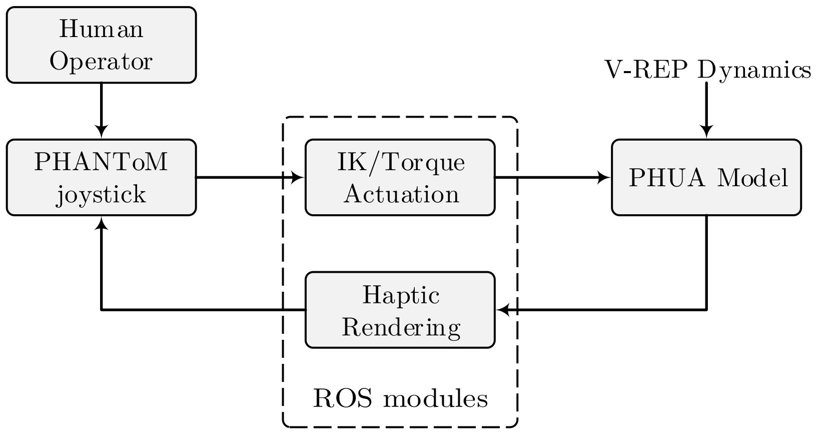


Figure 4.5: Position command diagram, from the user to V-REP interaction [13].

Only position commands are sent to the model, characterizing a direct relationship between the joystick position and the robot position.

Inverse kinematics mode

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 user control loop is always initiated relatively to the current end-effector position (hip position).

When the simulation is initiated, the current robot state and the joystick absolute position are known. 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 current hip position. 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. The user can refine the control, but increasing the sensitivity. Large position increments may lead to unexpected *jumps* during the simulation when the robot tries to reach the target position. This is particularly critical if the user moves the joystick too fast, or if simulation delays tend to appear. Delays on simulation might appear depending on the calculations weight. During real-time simulations, if the simulation time is not able to follow the real-time (e.g. because of some momentarily heavy calculations), the simulation time will try catching up the lost time, which results in an apparent speed-up, and instability. The simulation will jump some iterations, causing discontinuities that can be maximized by large position increments or rapid movements of the operator. Without compromising the robot workspace dimensions, the joystick workspace can be reduced to increase the resolution of the end-effector's action space, resulting in a fine-grained control.

The inverse kinematics calculations are entirely carried out by the module existing on V-REP. If the communication is successful, and if the simulation runs without significant delays, the robot's hip will follow the joystick movement accordingly. However, if a configuration is singular or not reachable the inverse kinematics task will *break*. The model will start to behave strangely, and the simulation loses its interest. Figure 4.6 illustrates the Cartesian correspondence between the joystick's end-effector and robot's hip, during the teleoperation.

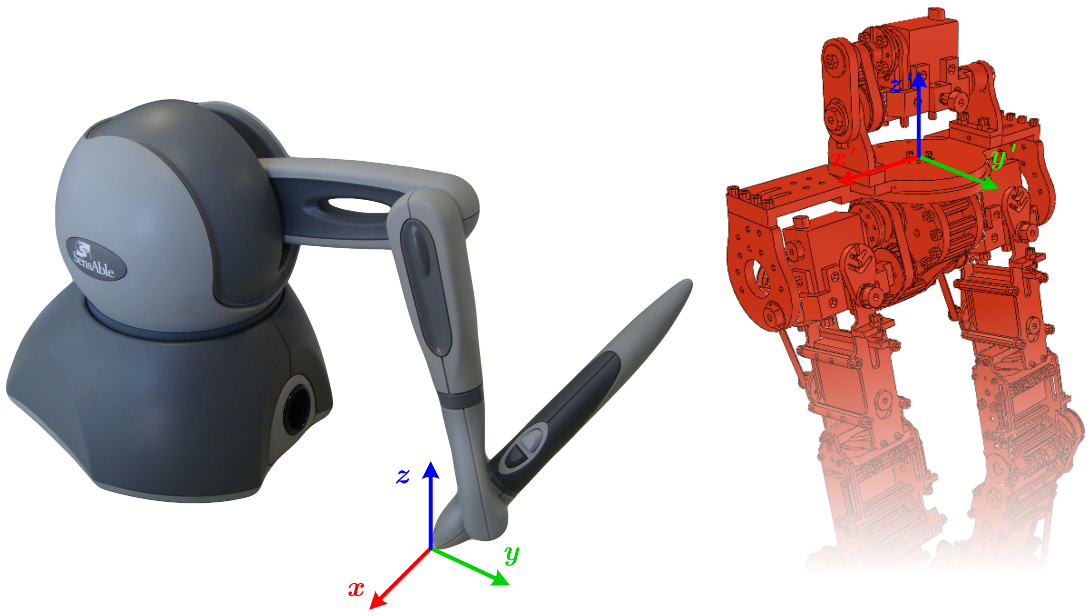


Figure 4.6: Inverse kinematics correspondence, for model teleoperation.

Since the default coordinate frame of the PHANToM device does not coincide with the V-REP model frame, a conversion should be done in order to obtain the axis orientation presented in the Figure 4.6.

$$\begin{bmatrix} x_{vrep} \\ y_{vrep} \\ z_{vrep} \end{bmatrix} = \begin{bmatrix} -x_{phantom} \\ z_{phantom} \\ y_{phantom} \end{bmatrix} \tag{4.3}$$

It should be noted that Equation (4.3) represents a conversion from the PHANToM original coordinate system to V-REP coordinate system, and not the existence of two different frames.

This command strategy using inverse kinematics has the advantage of direct coordinate mapping, but it does not present explicit singularity and non-reachable point avoidance mechanisms. One way of avoiding redundancies at the fourth joint (knee joints), is to ensure that both knees are in slight flexion. If the leg configuration does not get too close to a "stretched" configuration - singularity - the inverse kinematics solutions will keep "positive flexion" postures, for both kinematic chains. Additionally, if the system is defined as redundant, then the joint limits corrections will be applied during IK resolution. Limiting the knee joints to positive values will avoid singularity situations, since critical configurations cannot be reached. Enabling this feature will reduce the joint space dimensions, and consequently the range of positions reachable by the robot. Yet, its incorporation greatly contributes to adjust the model to real robot conditions.

Torque mode

The torque mode presented in this work implements the same position control of the inverse kinematics methodology, but the information retrieved from the joystick is a vector of joint positions. Thanks to its particular geometry, the PHANToM Omni provides a very interesting match with the degrees-of-freedom of the robot. Figure 4.7 provides a comparison between the joystick joints, and the right leg joints of the robot, demonstrating the accurate correspondence existing between these two. The joint numbering is done in accordance with the V-REP model (since the 3^{rd} joint, corresponding to thigh rotation (yaw), is currently blocked in the real platform, its representation can be ignored).

Even though the joystick has six degrees-of-freedom or and a very unique morphology that makes possible a perfect correspondence with all the robot joints, some of these degrees-of-freedom may not be used to control the mechanism. The strategy here formulated ignores the principle initially stated, which declares an absolute correspondence between the joystick joints and the robot joints. In fact, for the time being, only the first three joints are used to control the robot. However, future approaches based on joint-by-joint robot control can implement the full correspondence illustrated in Figure 4.7.

Controlling all the active joints of the model is not an easy task for the operator. The joystick state is directly sent to the robot, so the operator must be cautious about some aspects, in particular, the foot rotation. Foot contact with the ground is one of the most important, if not the most important aspect of the simulation. The dynamics in this interaction involves collision response algorithms that will establish how the shapes comprising the foot will react to the contact. This is of particular importance, since the control methodology relies on values measured by the force sensors located on the foot to evaluate the system dynamics, and compute the force feedback. Besides serving as dynamic links during the simulation, the forces sensors are used to measure forces acting at contact of the ground and the mechanism, and thus obtain information about the CoP position. However, correct measurements can be performed only if all force sensors are in contact with the ground. If some of the sensors deploys from the ground surface, the mechanism as a whole will rotate about the foot edge and overturn, the system is unstable. Both in single or double-support configuration, the robot must keep their feet parallel to the ground, in order to ensure the foot's whole area, and not only the edge, is in contact with the ground.

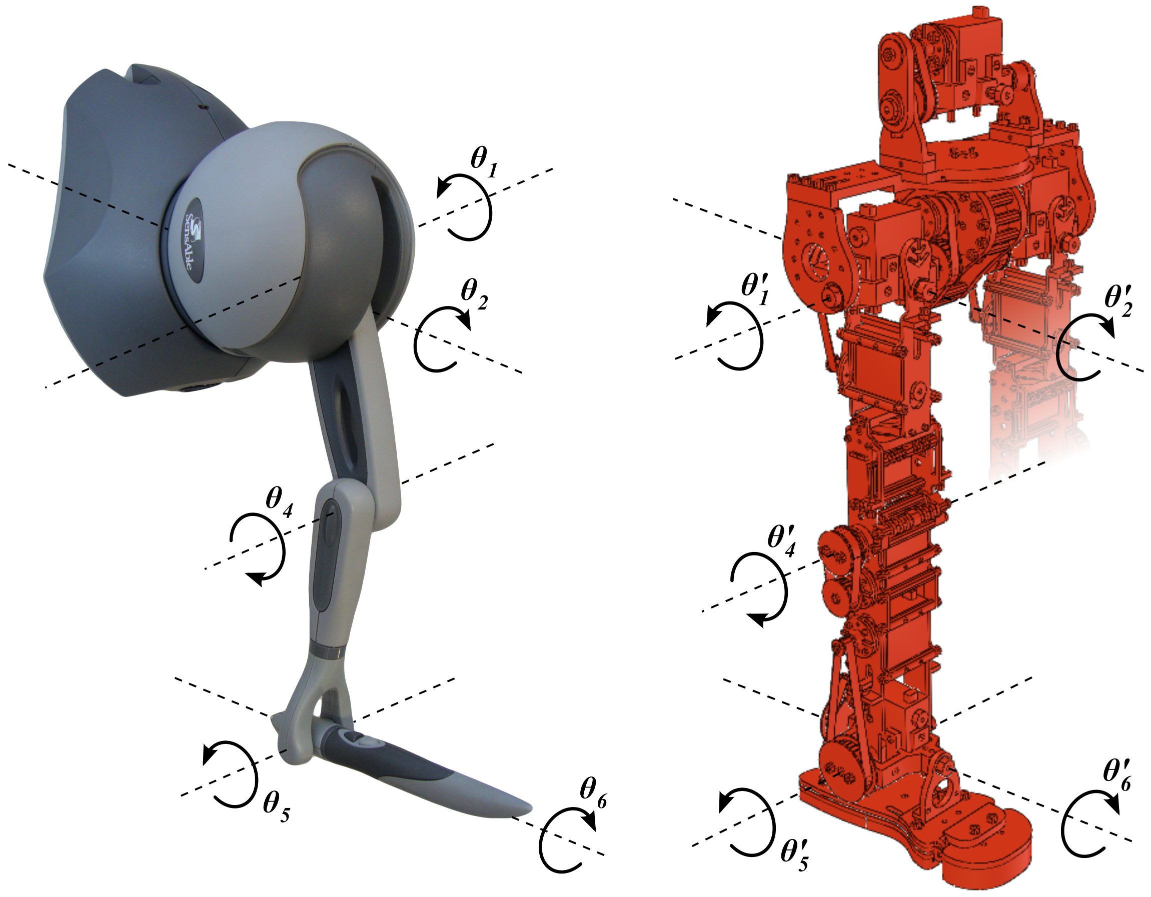


Figure 4.7: Joint correspondence between the PHANToM Omni and the V-REP model.

To ensure the foot is parallel to the ground, the joint vector applied to model's right leg can be written as follows in Equation (4.4). An analogous association can be applied to the left leg.

$$\begin{bmatrix}\theta_{1,vrep}\\ \theta_{2,vrep}\\ \theta_{4,vrep}\\ \theta_{5,vrep}\\ \theta_{6,vrep}\end{bmatrix} = \begin{bmatrix}\theta_{1,phantom}\\ \theta_{2,phantom}\\ \theta_{4,phantom}\\ \theta_{4,phantom}-\theta_{1,phantom}\\ -\theta_{2,phantom}\end{bmatrix} \tag{4.4}$$

Again, since the joystick joints do not coincide with the adopted convention, represented in Figure 4.7, the device's default joint space have to be converted to the model's joint space. Equation (4.5) describes this transformation, where $\theta_{default}$ represent the position of the first three joints, according to joystick's default reference, shown in Figure 3.2.

$$\begin{bmatrix}\theta_{1,phantom}\\ \theta_{2,phantom}\\ \theta_{4,phantom}\end{bmatrix} = \begin{bmatrix}\theta_{2,default}\\ \theta_{1,default}\\ -\pi/2+\theta_{2,default}-\theta_{3,default}\end{bmatrix}(rad) \tag{4.5}$$

While $\theta_{1,default}$ is determined by the base joint, the $\theta_{2,default}$ and $\theta_{3,default}$ are determined from the pieces "connecting-rod" and "crank". Due to the particular construction of the device, minimum and maximum values of $\theta_{3,default}$ are not constant and depend on value of $\theta_{2,default}$, and the angle inter-arms. The value in the 3rd element of the *default* matrix, describes this dependence.

The coordinate mapping in this loop is absolute, despite the morphological differences between the robot's kinematic chains workspaces and the joystick's own workspace. This assumption is based on the philosophy adopted with the project. Teaching the robot how to balance and move, thinking on a joystick as a robot leg, means that the joystick's configuration must fit plausible configurations for a humanoid leg.

4.2.2 Force Feedback

This work implements a general strategy for generating the force feedback based on the ZMP method, which can be applied to both single and double-support phases. Based on the ZMP location, the joystick renders a force on the user that allows him to feel the system dynamics. If balance maintenance is compromised, the user takes actions to restore the mechanism to a stable configuration.

Apart from the realization of the relative motion of the mechanism's links, the most important task of a locomotion mechanism during the gait is to preserve its dynamic balance, or stability. Concerning the ZMP method basic definition, the system is in dynamic equilibrium if the ZMP position is within the support polygon [32]. However, in reality, the ZMP can exist only within the support polygon, and all the calculated positions of this point outside the support polygon represent fictitious locations. When the ZMP approaches the support polygon edges, it will remain an overall indicator of the system behavior if no additional moments are acting at this point. Otherwise, the mechanism would start to rotate about the foot/feet and the system would collapse. In this case, the acting point of the ground reaction force would be on the foot edge, but it would be no longer the ZMP, since conditions of zero moment about the foot would not be satisfied.

The ZMP location defines a practically unavoidable stability margin, relatively to support polygon limits. Approaching the foot edges will drastically reduce the stability, and the system is more likely to lose its balance. Quantifying this margin represents the basic principle involved in the force feedback generation.

The algorithm compares the ZMP position with the real support polygon size and establishes a metric of stability/instability, based on the distance from the support polygon edge to the computed position of ZMP. Considering operation in dynamically stable conditions, in which the simulation makes sense, the ZMP coincides with the CoP [32], so its position is used to predict the dynamic behavior of the system and define the force feedback. The force feedback signals synthesized by the joystick are determined by a function that expresses simultaneously the CoP approximation to the robot's feet edges, and its separation from the most stable position.

Originally, the force rendering was based exclusively on the instability criterion, which relates the CoP position and the feet edges. However, during the first IK experiments, it was found that in lateral movements the force feedback generation was not consistent with the simulation, given that the force took nearly empty values. This is because of the support-polygon geometry. In these cases, the CoP excursion was not long enough to reach a "renderable" force zone in the lateral direction. In other words, the distance calculated to the front or back limits of the polygon was always lower than the calculated to the lateral edges, and thus the force rendering could not express the actual motion of the robot. This led to the incorporation of a complementary force rendering approach based on the CoP distance to the origin, which can perfectly describe the system's behavior in situations like this. This describes the unifying theory of force feedback generation.

Then, when both situations are verified, the resultant force is determined as follows in Equation (4.6), as a function of the two components.

$$\begin{aligned} \boldsymbol{F_R} &= \alpha \cdot \boldsymbol{F_1} + \beta \cdot \boldsymbol{F_2} \\ &= \frac{1}{\boldsymbol{\eta}+1} \cdot \boldsymbol{F_1} + \frac{\boldsymbol{\eta}}{\boldsymbol{\eta}+1} \cdot \boldsymbol{F_2} \quad (N) \end{aligned} \tag{4.6}$$

This equation calculates the resulting force vector sent to the device, giving complementary weights to the two methodologies by defining a dimensionless ratio $\boldsymbol{\eta}$, which depends on the force magnitude.

The force magnitude $F(s)$ is determined for each case, using expressions (4.7) and (4.8). These equations, obtained experimentally, describe the force quantification as function of the CoP position relative to the robot's stability point (origin point of the feet coordinate system), and its distance to the robot's support base boundary, $F_1(s)$ and $F_2(s)$ respectively.

$$F_1(s) = \begin{cases} 0 & \text{if} \quad s \in [0 \quad d_{deadzone,1}[\\ m_1 \cdot s - b_1 & \text{if} \quad s \in [d_{deadzone,1} \quad d_{intersec}[\\ 1/(\gamma_1 \cdot (1 + e^{-\alpha_1 (s-\beta_1)})) + 0.5 & \text{if} \quad s \in [d_{intersec} \quad d_{limit,1}[\\ 3 & \text{if} \quad s \in [d_{limit,1} \quad +\infty[\end{cases} \tag{4.7}$$

$$F_2(s) = \begin{cases} 0 & \text{if} \quad s \in]d_{deadzone,2} \quad +\infty[\\ -m_2 \cdot s + b_2 & \text{if} \quad s \in]d_{intersec} \quad d_{deadzone,2}] \\ 1/(\gamma_2 \cdot (1 + e^{\alpha_2 (s-\beta_2)})) + 0.5 & \text{if} \quad s \in]d_{limit,2} \quad d_{intersec}] \\ 3 & \text{if} \quad s \in]0 \quad d_{limit,2}] \end{cases} \tag{4.8}$$

Graphically, this formulation is represented as Figure 4.8 shows, for a maximum force of 3N (approximately 10% below the hardware threshold, for safety reasons).

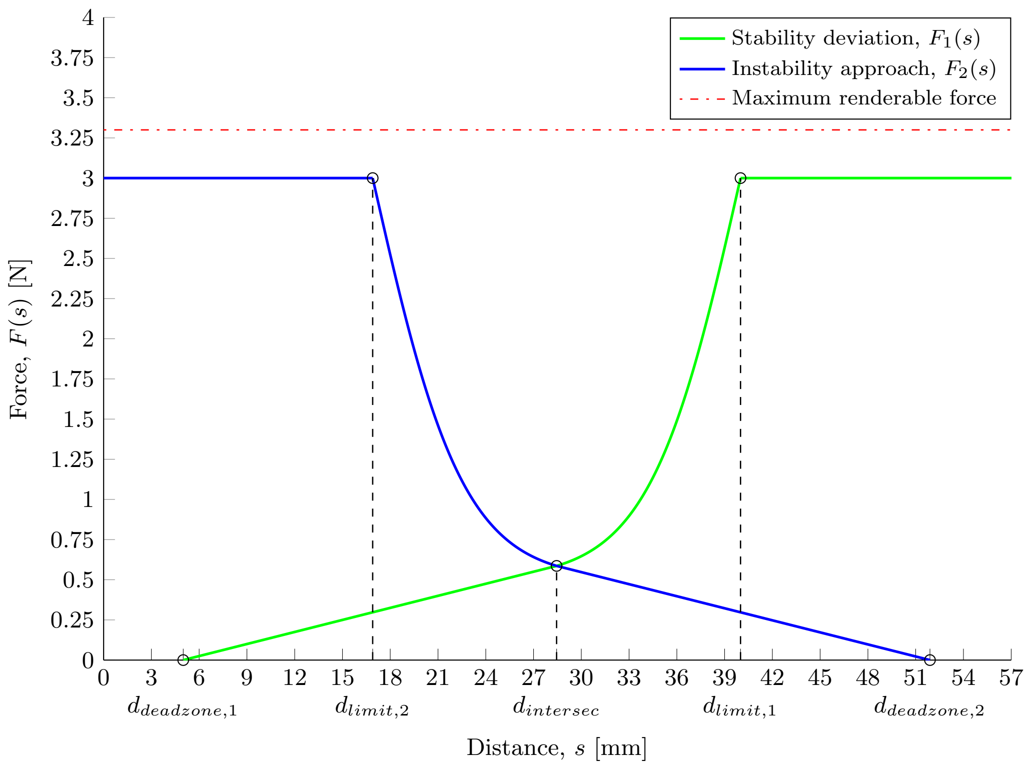


Figure 4.8: Two complementary formulations for the haptic force generation.

The force vectors $\boldsymbol{F_1}$ and $\boldsymbol{F_2}$ are decomposed into their x and y components according to the Equation (4.9). The vertical component, z, of each vector becomes null as a consequence of CoP and support base bi-dimensionality.

$$\boldsymbol{F} = F(s) \cdot \frac{\boldsymbol{COP}}{\|\boldsymbol{COP}\|} \quad (N) \tag{4.9}$$

To specify the weights given to each strategy, the rendering algorithm evaluates the growth rate of force magnitude for both methodologies, according to Equation (4.10), and finally defines the ratio

η, according to Equation (4.11).

$$\frac{\Delta \boldsymbol{F}}{\Delta t} = \frac{\boldsymbol{F}(t) - \boldsymbol{F}(t - \Delta t)}{\Delta t} \tag{4.10}$$

$$\boldsymbol{\eta} = \left| \frac{\Delta \boldsymbol{F_2}}{\Delta \boldsymbol{F_1}} \right| \tag{4.11}$$

With the ratio defined by the Equation 4.11, the algorithm balances the two strategies, or force components, but giving more importance to the predominant one.

This type of formulation, initially dedicated to inverse kinematics experiments in the Cartesian space, can be easily applied to torque mode experiments. However, due to the "leg-like" configuration of the joystick, the y component of the resultant force must be rendered in the z direction (remember Figure 4.6).

Chapter 5

Experiments and Results

This chapter describes the haptic demonstrations undertaken to implement and test the methodologies described previously, as well as the software applications developed to support them. The haptic demonstrations have been divided into two major parts: *inverse kinematics control*, and *joint-by-joint* or *torque control*, thus following the same operation mode separation. Both consist basically of teleoperating the robot model using the haptic device, and the general CoP position to generate a force feedback for user interaction. The simulation loop functioning, including simulation speed and ROS frequency, and other essential features are also explained.

5.1 Simulation Loop

This section describes the first order concepts related to simulation control, for real-time simulations definition and characterization. The simulator operates by advancing the simulation time at constant time steps. Figure 5.1 illustrates the main simulation loop, for non real-time simulations.

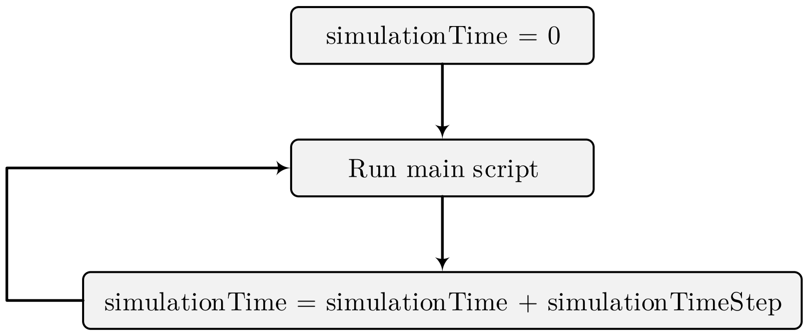


Figure 5.1: Main simulation loop (adapted [17]).

Real-time simulation is supported by trying to keep the simulation time synchronized with the real time (Figure 5.2). Depending on the simulation complexity, performance of the computer and simulation settings, real-time simulation might not always be possible. Inverse kinematics computation, in particular, is a highly resource-intensive process. When simulating IK models in real-time, there are typically some periods when the simulation time is not able to follow the real-time, resulting in unsatisfactory time delays. Usually, these delays are not substantial and can be neglected. However, in really heavy simulations these delays can be large enough to implicate the simulation's crash.

Due to the nature of this project, simulating the PHUA model in real-time is an important condition that must be fulfilled. The basis of the teleoperation concept, and of this project in particular, requires the real-time control of the robotic platform. The simulation provides excellent means to

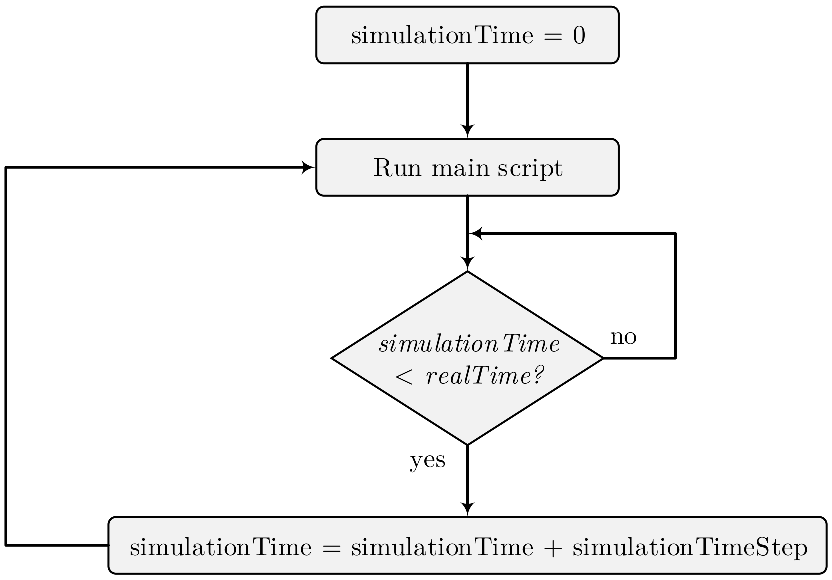

simulationTime = 0
Run main script
simulationTime < realTime?
no
yes
simulationTime = simulationTime + simulationTimeStep

Figure 5.2: Real-time simulation loop (adapted [17]).

overcome some of the obstacles present on the real robot, test different scenarios, and even speed up or slow down the robot's response. Though this would bring numerous advantages, the simulation's main purpose is to simulate the real-world conditions to which the robot is subject, and build a scenario that can later be reproduced in reality.

For teleoperation, there must exist a time reference common to both the operator and the simulation scene. This is the only possible way to compare the user's commands with the robot's response, and with the feedback generation. The simulation time must, to the extent possible, equal the clock time.

Simulation Speed

In non real-time simulations, the simulation speed (i.e. the perceived speed) is mainly dependent on two factors: the simulation time step and the number of simulation passes for one rendering pass [17]. Each time the main script is executed, the simulation time is incremented by the simulation time step. This parameter can be re-configured before the simulation starts, even though the default value was kept during all the simulations. Using large time steps results in fast but unstable simulations. Small time steps, on the other hand, will generally lead to more precise simulations, but will take more time. The number of simulation passes per frame (ppf), or the number of simulation passes for one rendering pass, indicates the number of main script executions before the screen refresh. The default unitary value is also kept.

In the case of a real-time simulation, the simulation speed mainly depends (ignoring hardware restrictions), on the real-time multiplication coefficient, but also on the simulation time step. A too small simulation time step might not be compatible with the real-time character of a simulation, because of the limited calculation power of the computer [17]. A multiplication factor of X would try to run the simulation X times faster than real-time. Obviously, this value is kept unitary too. The experiments have shown that a time step of $dt = 50\ ms$ (default value) indeed guarantees an appropriate balance between speed and accuracy.

Threaded rendering

To ensure the simulation time follows the real-time without significant delays the *threaded rendering* option is enabled. The rendering operation will always increase the simulation cycle duration, and therefore also slowing down simulation speed. This is an acute problem when simulations have real-time requirements. When the *threaded rendering* mode is enabled, the simulation cycle will only consist of the execution of the main script, thus, simulations will run at maximum speed. Rendering will run in a separate thread during the simulation, effectively accelerating the simulation cycle [17].

This measure was the solution found to avoid time delays in real-time simulations. However, some drawbacks of this implementation have to be considered. For instance, the lack of synchronism between the rendering and the simulation, which may cause visual glitches.

ROS Frequency

The ROS communication between V-REP and other ROS nodes is mostly made via a pre-defined series of ROS messages. The diagram of Figure 5.3 illustrates how ROS messages are handled on the server side (i.e. on the V-REP ROS plugin side).

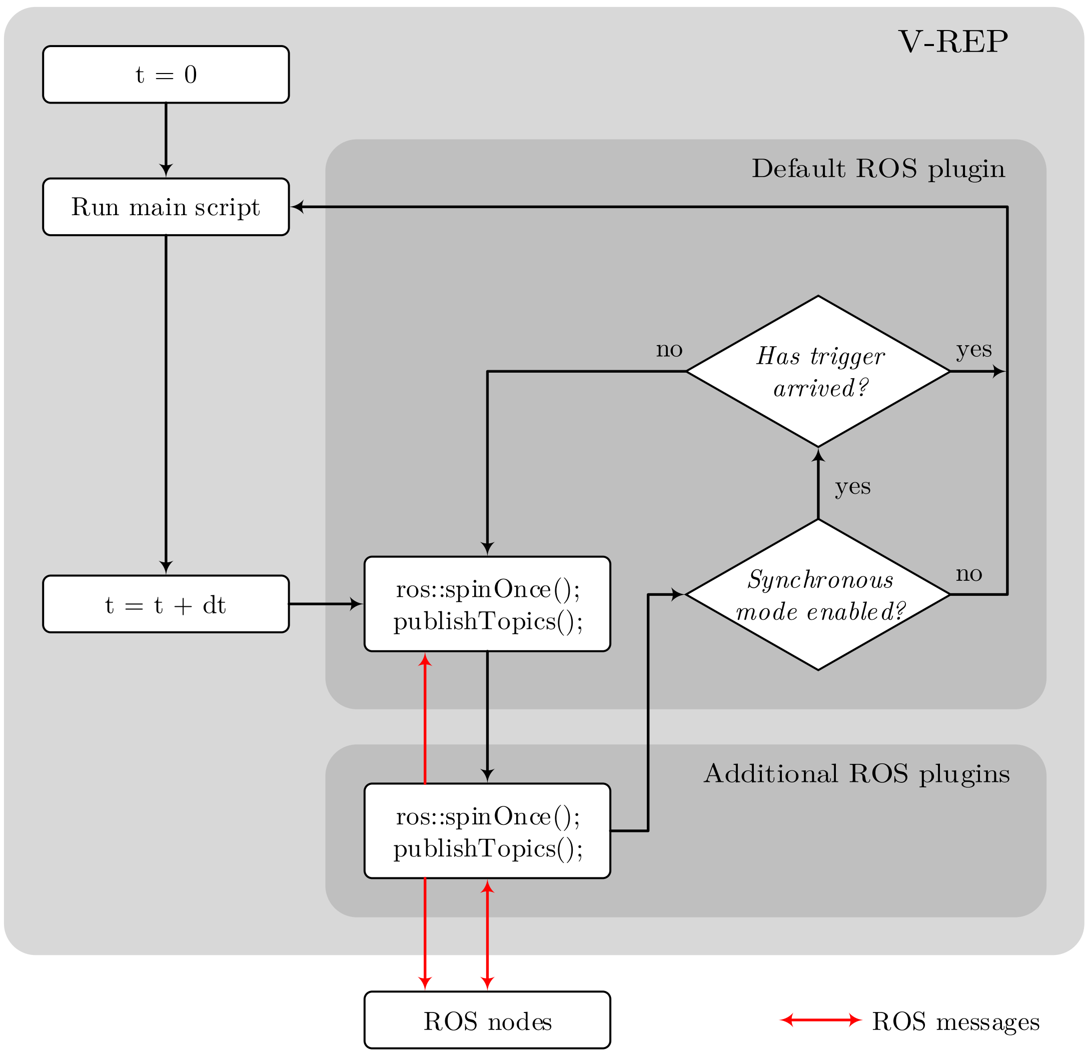


Figure 5.3: ROS message handling, server side (adapted [17]).

As can be seen in this diagram, the ROS messages are processed one time in each simulation step, as the main script is executed. The simulation loop, which is basically the main script program, consists of two phases: the actuation phase, and the sensing phase [17]. The regular part of a default main script is composed of these two sections. The first section (the actuation section) allows actuating or modifying the scene content, handling joints, dynamics, or inverse kinematics groups, while the second section (the sensing or probing section) allows sensing and probing the scene content, handling collisions, distance calculations, proximity sensors, force sensors, etc. Mixing up actuation and sensing commands will lead to an unpredictable simulation run.

A non-thread child script, which is used to control the model, also has this phase separation implemented. This script handles the ROS communication, subscribing and publishing to a wide variety of topics. Incoming messages are handled and their data is processed - actuation phase. Then, the joint state, sensor values, object poses, etc, are read in the sensing phase, and the information is published by V-REP. The display is processed next.

From previous statements, it is immediately recognizable that the ROS frequency is directly associated with the simulation speed. The rate at which data is subscribed and processed/published/applied is influenced by the simulation time step, and the other parameters mentioned above; these dictate the main script calling rate. A new message is basically emitted at the end of each simulation loop. Posting messages more frequently than that wouldn't make sense, since the simulation state wouldn't change. Considering the simulation loop runs once every 50 *ms* (given that the simulation content is not too heavy), there will be 20 simulation passes per second. Thus, V-REP will send out data (and also read and evaluate it) 20 times per second. This is the ROS frequency established for all the framework. Since V-REP limits the frequency value, the other nodes on the loop cannot update their state at higher rates, and consequently cannot send information faster, too. Despite the high frequency at which the *phantom_control* node publishes the joystick data, these effects will not be felt in the simulation. The frequency of the designed setup system is approximately 20 Hz.

Conducted Experiments

To ease the reading, a basic scheme listing the simulation experiments presented in this work is shown in Figure 5.4.

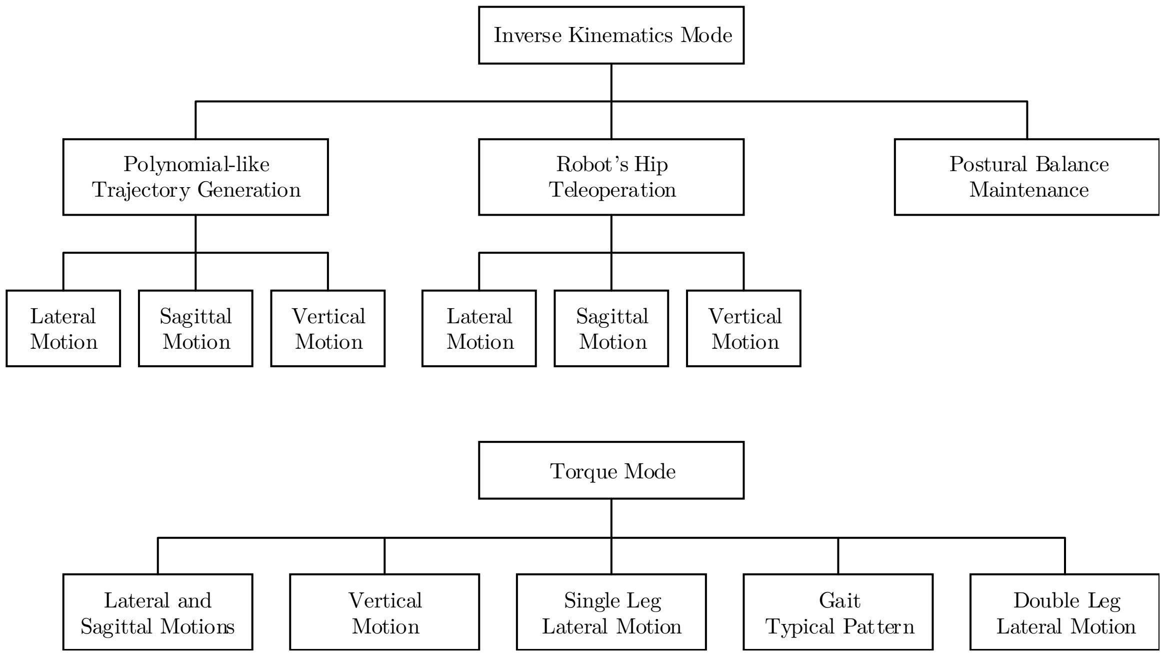

Inverse Kinematics Mode
Polynomial-like Trajectory Generation
Robot's Hip Teleoperation
Postural Balance Maintenance
Lateral Motion
Sagittal Motion
Vertical Motion
Lateral Motion
Sagittal Motion
Vertical Motion
Torque Mode
Lateral and Sagittal Motions
Vertical Motion
Single Leg Lateral Motion
Gait Typical Pattern
Double Leg Lateral Motion

Figure 5.4: Summary of simulation experiments.

5.2 Inverse Kinematics Mode

In the inverse kinematics demonstrations, the operator typically uses the joystick to control the robot's hip position, while the system is monitoring the CoP location. It then renders a force vector, whose magnitude depends on the CoP digression into unstable regions.

During the demonstrations, the robot's feet are kept at a constant distance from each other, and they are considered to be in continuous and direct contact with the ground. Nevertheless, due to heavy IK calculations, simulation glitches may appear and lead to undesired quivering behaviors that will avoid the feet of keep touching the floor. Something similar occurs when the user operates the joystick too fast. Inverse kinematics calculations cannot resolve the joint state fast enough to fit the hip position, which results in a apparent rigid body behavior. In such cases, the force values measured on the feet are unexpectedly large. As a result, the CoP position alternates consecutively between the two feet, and the feedback generation is totally compromised. Synthesized forces on the joystick are continuously alternating their direction, causing a exponential positive feedback that can be hardly controlled by the user. This means that the operator loses control over the teleoperation scene, and a falling scenario is very likely to happen. Such simulations are considered irrelevant, and their results are ignored. These are not presented in the document.

The conducted experiments test different types of actuation in order to evaluate the system's natural behavior, and how the user's operation interferes with it. Results from automatically generated position are presented as a reference to the subsequent experiments.

In most of the experiments, the robot performs simple movements in one of the fundamental directions. So that, they can be described by the standard anatomical planes represented in Figure 5.5. The XYZ coordinate frame represents the sign convention adopted for the IK experiments. It does not coincide with the robot frame, which is defined on the model's feet.

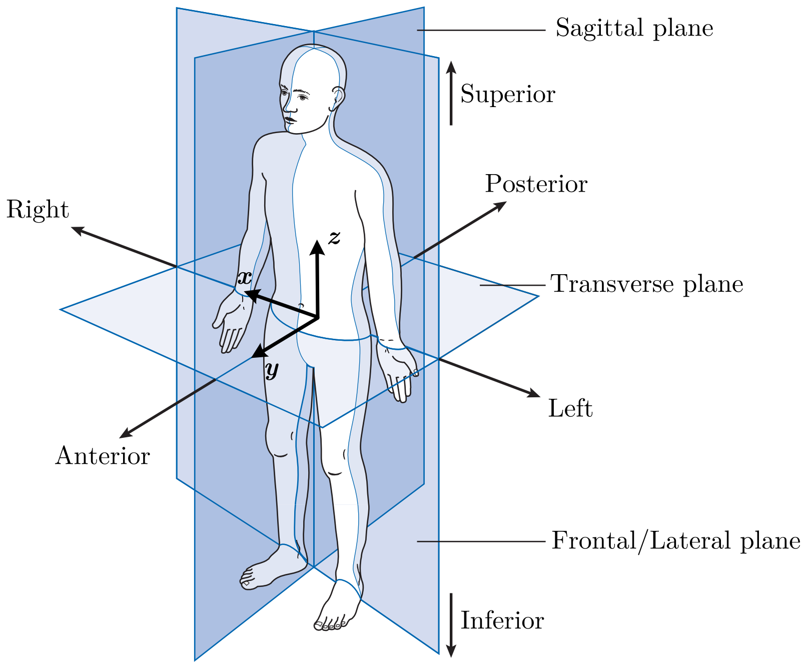


Figure 5.5: The anatomical position, with three reference planes and six fundamental directions (adapted [16]).

For all the experiments, only the lower limbs are simulated. Figure 5.6 shows an example of a teleoperation task in IK mode, where the model used for simulation can be seen. The decision of excluding the torso and the upper limbs from the simulations is due to three main factors: lack of methodologies to control the joints and consequently the towering mass above the pelvis, which greatly influences the robot's balance; increase of the simulation's weight, which could compromise the real-time simulation conditions; absence of the upper body in the current experiments on the real platform.

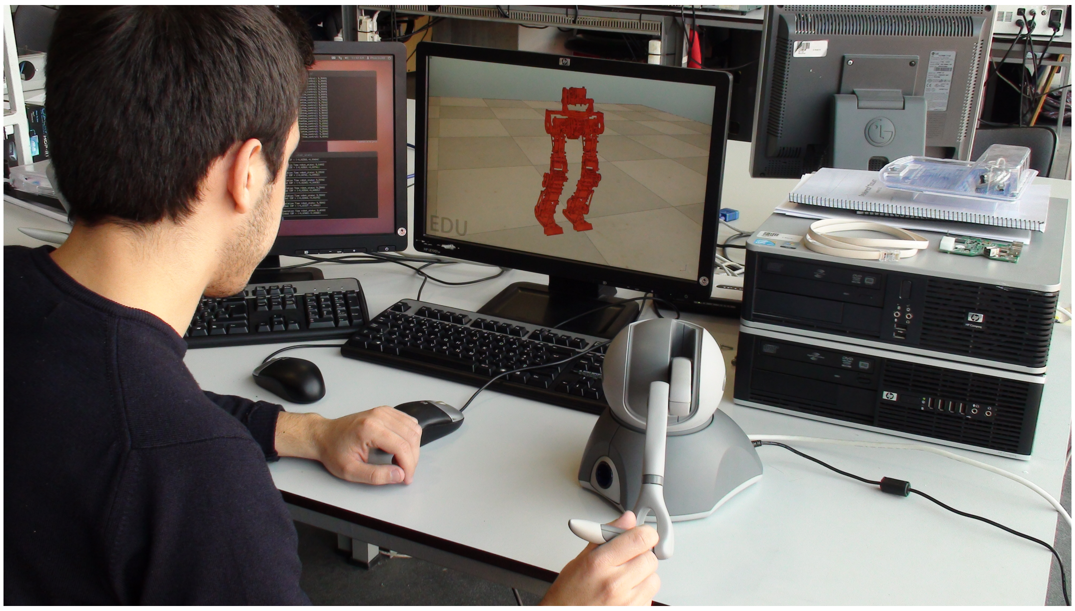


Figure 5.6: Example of a teleoperation task in IK mode.

5.2.1 Polynomial-like Trajectory Generation

In this first experiment, the hip's motion is automatically controlled by the *phantom_control* module, which uses a polynomial function to calculate the robot's end-effector excursion. The user has no role in teleoperating the system, since a synthetic motion order is applied. On its turn, the force feedback calculation uses the methods already described to apply the computed forces to the haptic device. Hence, although the user is not directly controlling the robot, he is able to feel the system's dynamics during the simulation. Disposing of the user's control, allows for the assessment of the mechanism's most natural reaction, for instance, joint angles and velocity states, and CoP trajectory evaluation.

Hip's lateral movement

For this demonstration, the hip moves sideways in the XOZ plane, with the ankle's inversion/eversion and the hip's abduction/adduction joints performing the relevant movements. The hip is held at a fixed height of $z = 0.4032\ m$, as the polynomial generates the lateral component of the end-effector's position within a specified interval of values, $x = [-0.05, 0.05]$. These values express the range of all possible positions for the robot's motion, beyond which the joint limits are exceeded and the model breaks. These are based on experimentally obtained data. Figure 5.7 describes the hip's motion during this demonstration. In Figure 5.12 the extreme positions occupied by the hip are illustrated.

From the observation of this figure (5.7), a slight delay between the expected and the real robot position can be perceived. This delay in position measurements is constant over the simulation time, and corresponds to approximately 0.1 s. The indicated value reveals that the possible cause for this difference may be associated with the simulation actuation and sensing phases. Since the *theoretical* position is measured at the time of its publishing by the *phantom_control* module, and the real PHUA

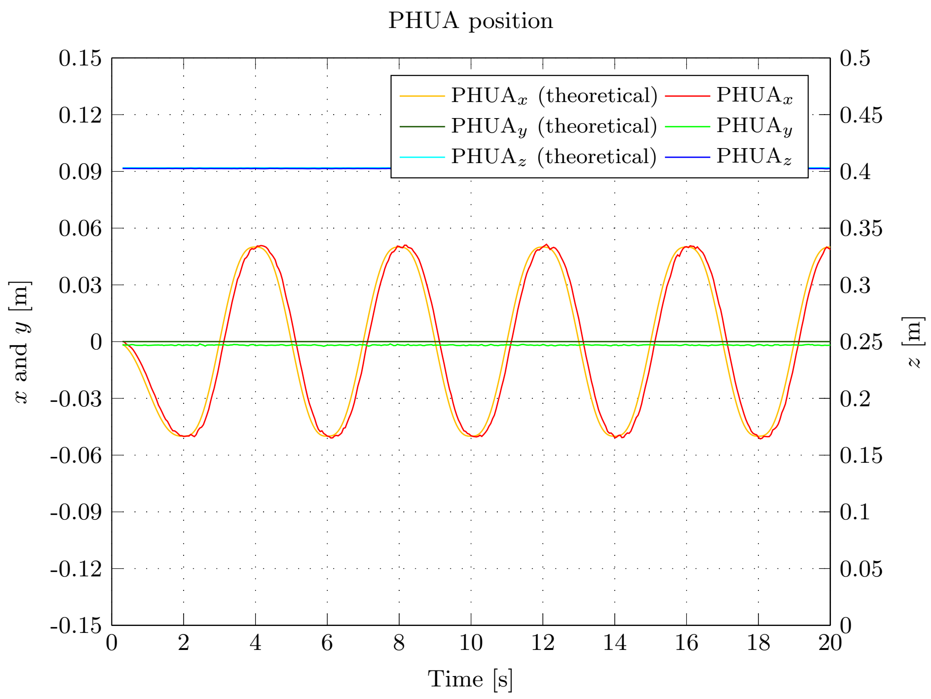


Figure 5.7: Model's hip position during the lateral free-movement simulation.

position is recorded by the *robot_state* when V-REP publishes it, considering the main script executes in 50 ms increments, this assumption makes absolute sense.

The 5th degree polynomial function generates the consecutive values for hip position depending on the previously mentioned limits and the simulation time, driving the robot from $x = -0.05$ to $x = 0.05$ in about 2 seconds (though this value is always dependent on the simulation response in real-time loop). When any of the limits is reached, the same function generates the robot motion in the opposite direction.

The joint state evolution, illustrated in Figure 5.8, represents the results from the inverse kinematics calculations carried out by V-REP, concerning the desired end-effector's position. These confirm the predominant role of the hip and ankle joints in the y direction (hip and ankle pitches), but also shows the importance of the knee joints motion in reaching the programmed position. Both legs have to be almost stretched in order to achieve the farthest position.

Figure 5.9 shows the path of the CoP during the demonstration, where it is clearly recognizable the hip's side movement. In Figure 5.10 the x and y components of CoP during the experiment are depicted. The y has very little variation when compared to the x component, which has a wide range of variations. In fact, the CoP location closely follows the hip position, asserting this reference point as an excellent indicator of the system's dynamics.

Using the proposed force feedback methodology, the user feels a force in the direction of the robot's unbalance. Although the joystick is not directly controlling the simulated model, by pressing the first button on the device, the user can feel the robot's dynamics reflected by the CoP's motion throughout the feet area. Figure 5.11 represents the feedback force generated by the joystick, on the user. As expected, the y component of the force vector is virtually nonexistent, and cannot certainly be felt by the user. On the other hand, the x component presents high peaks at the exactly same time steps where the CoP_x reaches its maximum value, thus efficiently recreating the system's behavior.

As stated, the generated force follows successfully the CoP oscillations, however, due to the exponential part of the force rendering algorithm, tiny variations in the CoP location greatly influence the calculated force. This is particularly unfortunate for the operator's teleoperation, since the feedback can affect the control itself. To avoid critical situations, a float-point filter is implemented "on the fly", computing the final force value based on the current and the previous calculated values, according to Equation (5.1), where $\beta = 0.4$, $F_R^{'}(t)$ is the filtered force, and F_R is the actual force before filtering. This filter is applied to every simulated experiments.

$$\boldsymbol{F_R^{'}}(t) = (1-\beta)\boldsymbol{F_R^{'}}(t-\Delta t) + \beta \boldsymbol{F_R}(t) \tag{5.1}$$

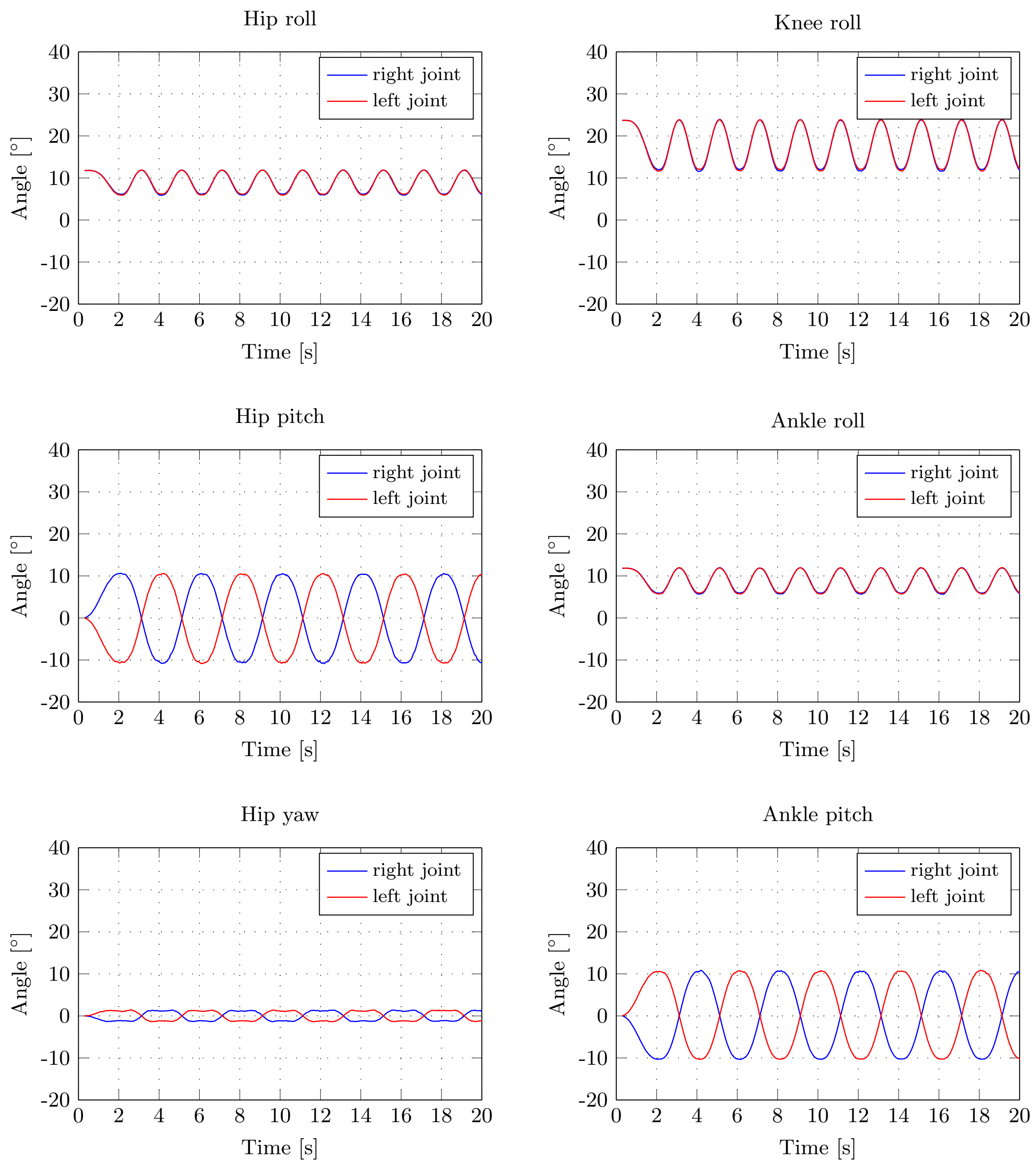


Figure 5.8: Robot's joint state evolution during the lateral free-movement simulation.

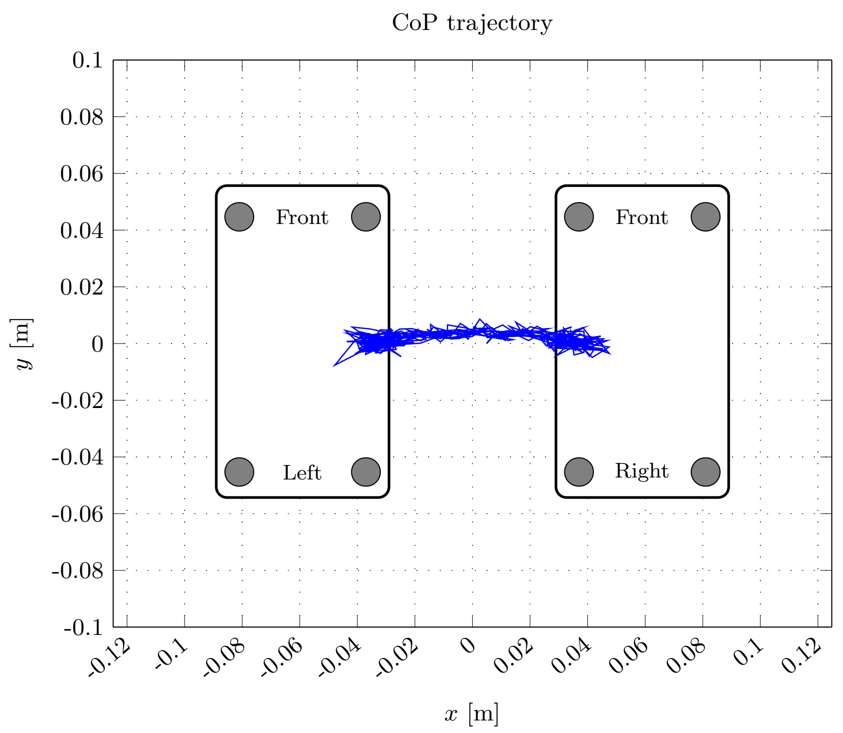


Figure 5.9: Complete trajectory of the CoP point during the lateral free-movement simulation.

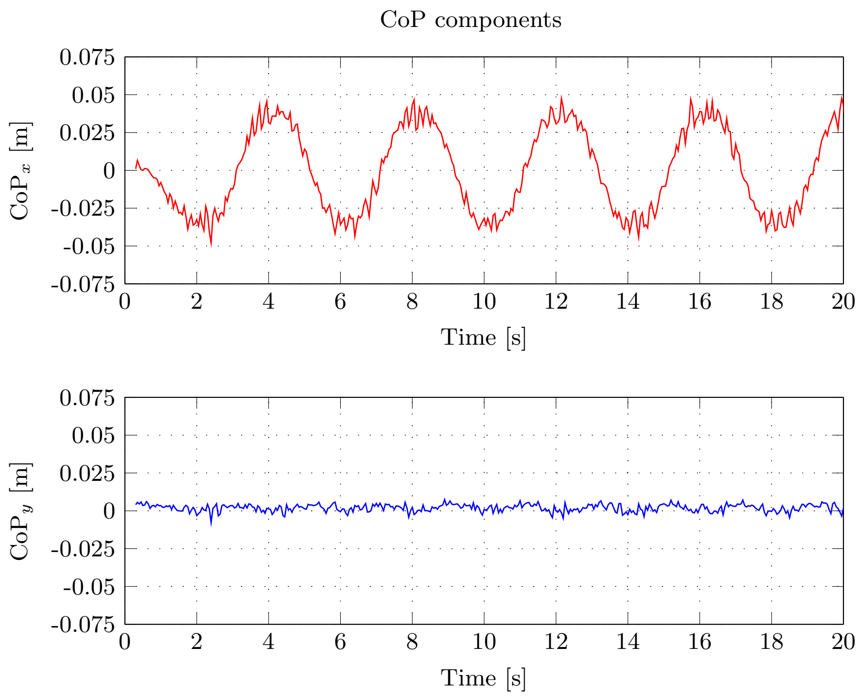


Figure 5.10: Bi-dimentional CoP components variation during the lateral free-movement simulation.

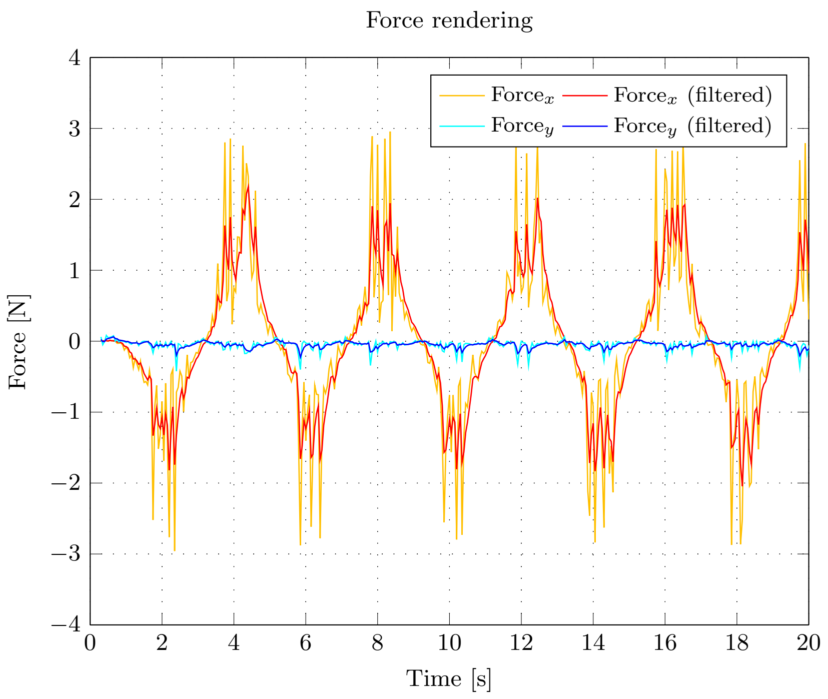
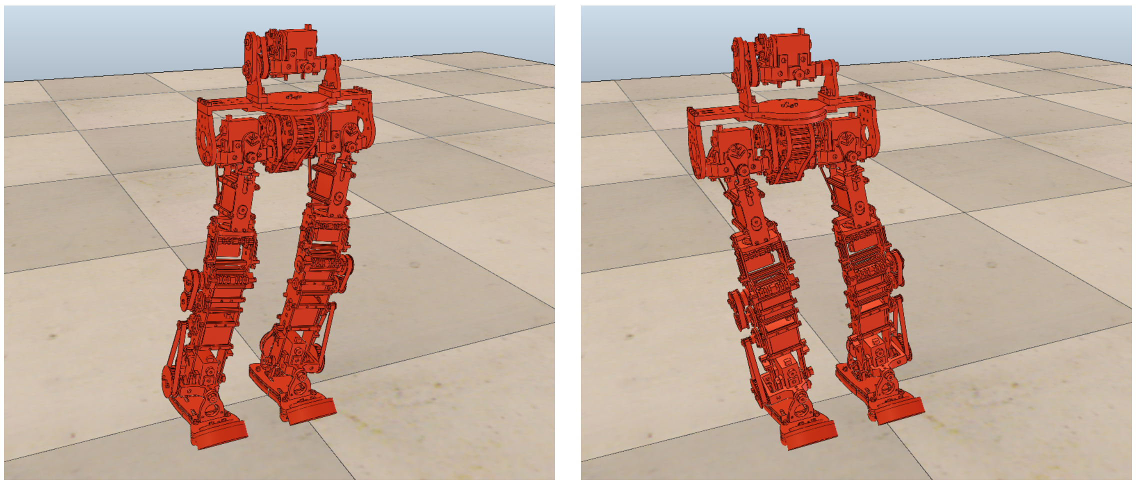


Figure 5.11: Force rendered by the haptic device during the lateral free-movement simulation.



(a) PHUA$_x$ = -50 mm.

(b) PHUA$_x$ = 50 mm.

Figure 5.12: Extreme positions ocuppied by the robot during the lateral free-movement simulation.

Hip's sagittal movement

In this simulation, the robot's hip moves back and forth along the YOZ plane. As in the lateral motion, the robot maintains a constant height of $z = 0.4032\ m$ during the simulation. However, in this demonstration, the polynomial function returns the values to be applied to the y coordinate of the hip, as the x coordinate equals zero. Figure 5.13 shows the movement performed by the robot's hip during the simulation.

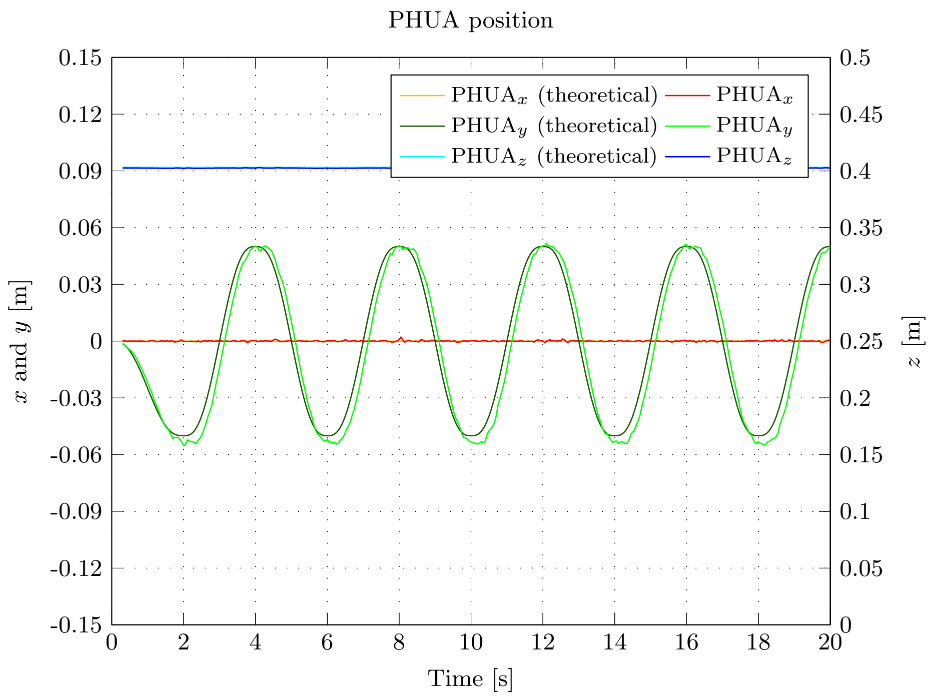


Figure 5.13: Model's hip position during the sagittal free-movement simulation.

Again, the motion is limited to a range of positions, which is in this case $y = [-50, 50]$ (Figure 5.18). This interval allows the movement performance without compromising the joints limits of the robot. The action here simulated is of particular interest for the walking gait generation, as the forward balancing of the body greatly contributes to this activity.

This simulation reveals a strong dependence on the knee, and hip and ankle roll joints (x direction), as expected. The remaining joints seem not to be actuated is this movement. Besides, given the symmetry of the model along the movement direction, the joint values for both left and right legs are quite similar (see Figure 5.14).

Figures 5.15 and 5.16 represent the CoP variation during this experiment, which distinctly denotes the back and forth motion of the robot. As with the previously described movement (in the perpendicular direction), the predominant component closely tracks the hip position in a qualitative and quantitative manner.

The generated forces in this experiment are shown in Figure 5.17. In the y axis, the higher forces are created when the robot goes forward. In fact, by observing the Figure 5.16, a higher deviation of the CoP into the positive side is noticeable. At distant positions, this has a considerable influence on the feedback force generation.

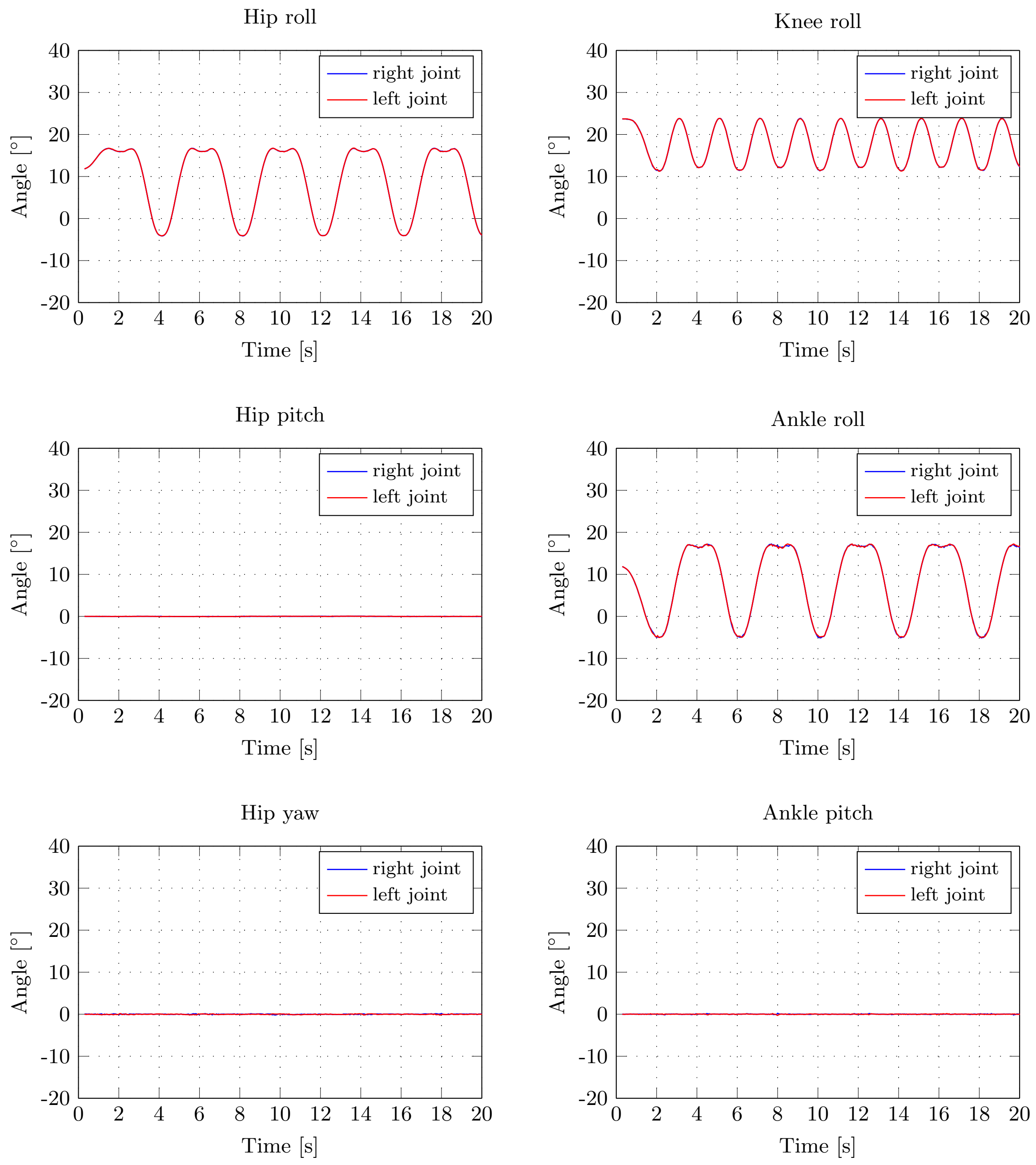


Figure 5.14: Robot's joint state evolution during the sagittal free-movement simulation.

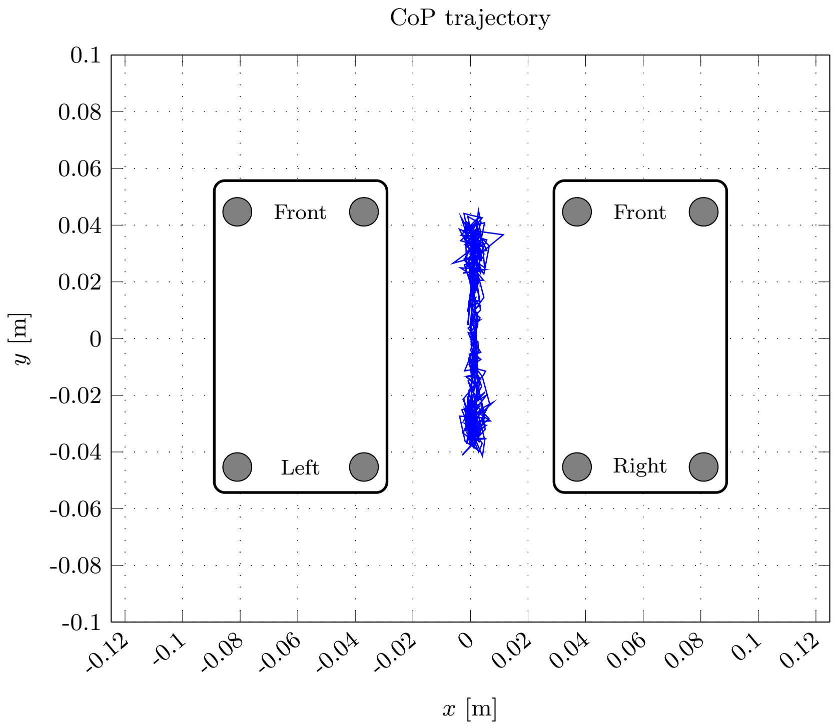


Figure 5.15: Complete trajectory of the CoP point during the sagittal free-movement simulation.

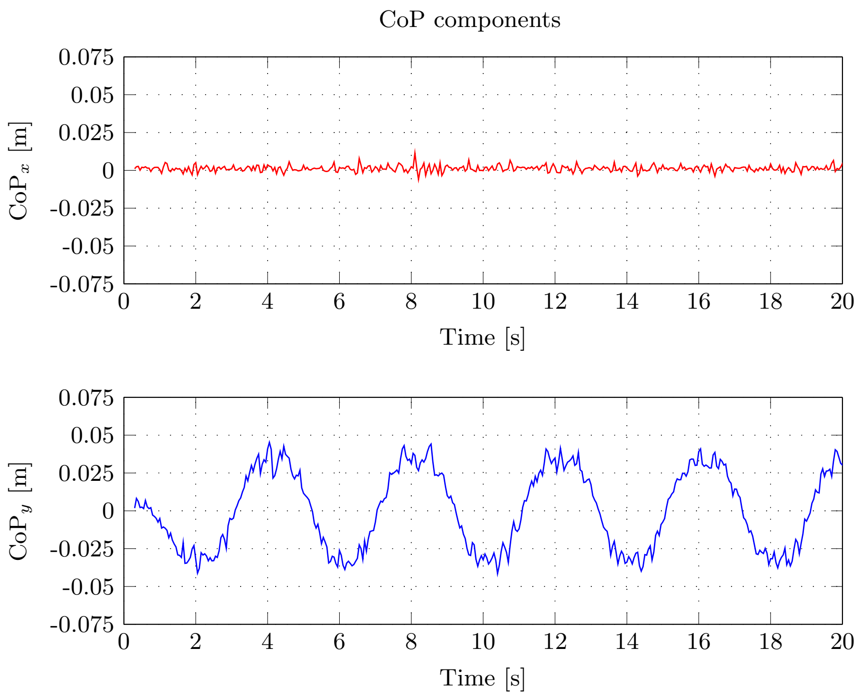


Figure 5.16: Bi-dimentional CoP components variation during the sagittal free-movement simulation.

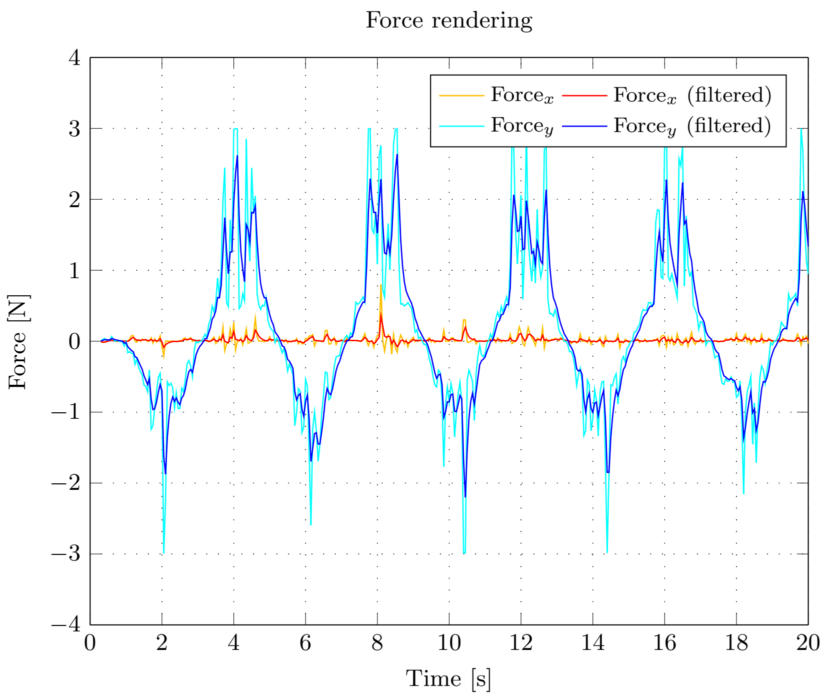
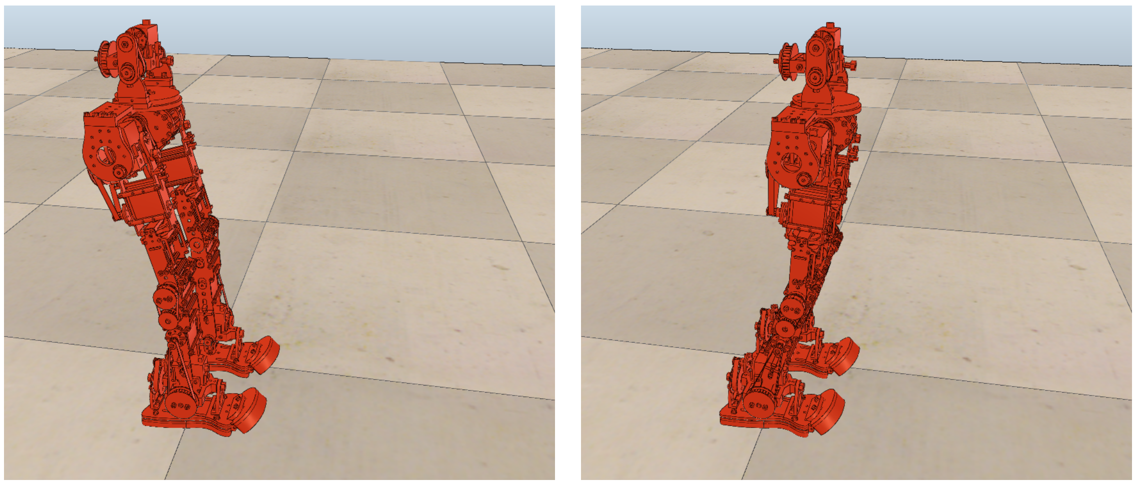


Figure 5.17: Force rendered by the haptic device during the sagittal free-movement simulation.



(a) PHUA$_y = -50\ mm$.

(b) PHUA$_y = 50\ mm$.

Figure 5.18: Extreme positions ocuppied by the robot during the sagittal free-movement simulation.

Hip's vertical movement

This simulation consists of the vertical movement of the hip. Once again, one of its position coordinates is automatically generated by the 5^{th} degree polynomial function. In this case, the robot does not maintain a fixed height, and varies within $z = [0.3572, 0.4072]$, positions represented in Figure 5.24. The x, and y coordinates are null, as Figure 5.19 shows.

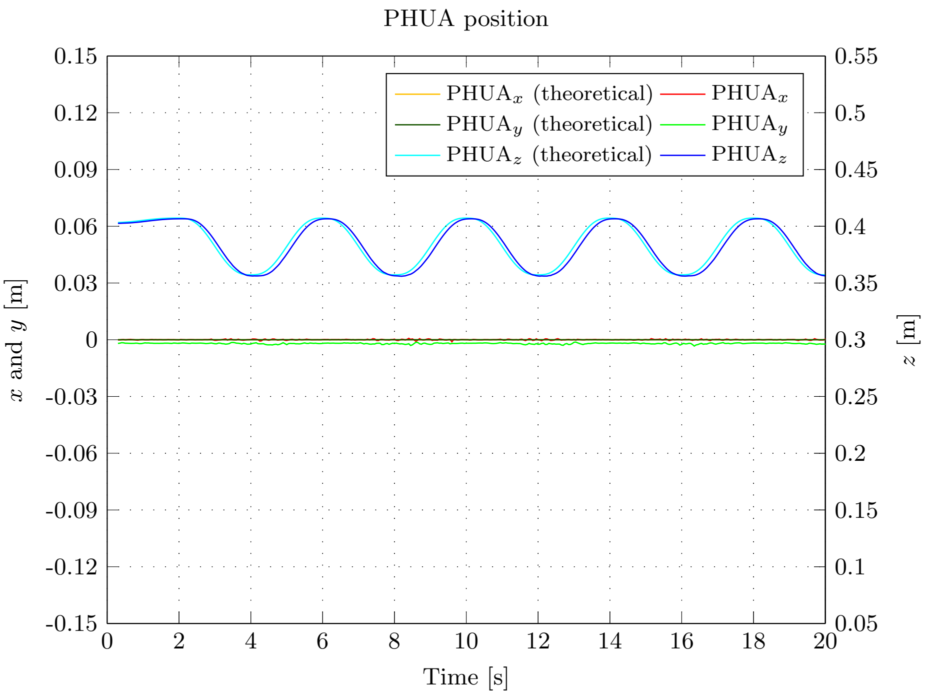


Figure 5.19: Model's hip position during the vertical free-movement simulation.

The inverse kinematics module actuates three main joints: flexion/extension of the ankle, knee and hip. As expected, these joints show a huge variation on their values during the simulation, particularly the knee joints. In contrast, the joints in the y direction (pitch) keep their values on zero (Figure 5.20).

Since the movement amplitude is constrained to the joint limits of the ankle, the hip's excursion is not too long, and the CoP variations were minimal, as depicted in Figures 5.21 and 5.22. Nonetheless, there is a observed trend towards the migration of CoP to positive y regions when the robot is on its lowest position.

As a consequence of this small CoP deviation, the rendered force is also quite small. In fact, the force values represented in Figure 5.23 were affected of a scale factor ($\times 3$), introduced to the force magnitude calculations during the simulation. Otherwise, the feedback force would be so insignificant that would become useless.

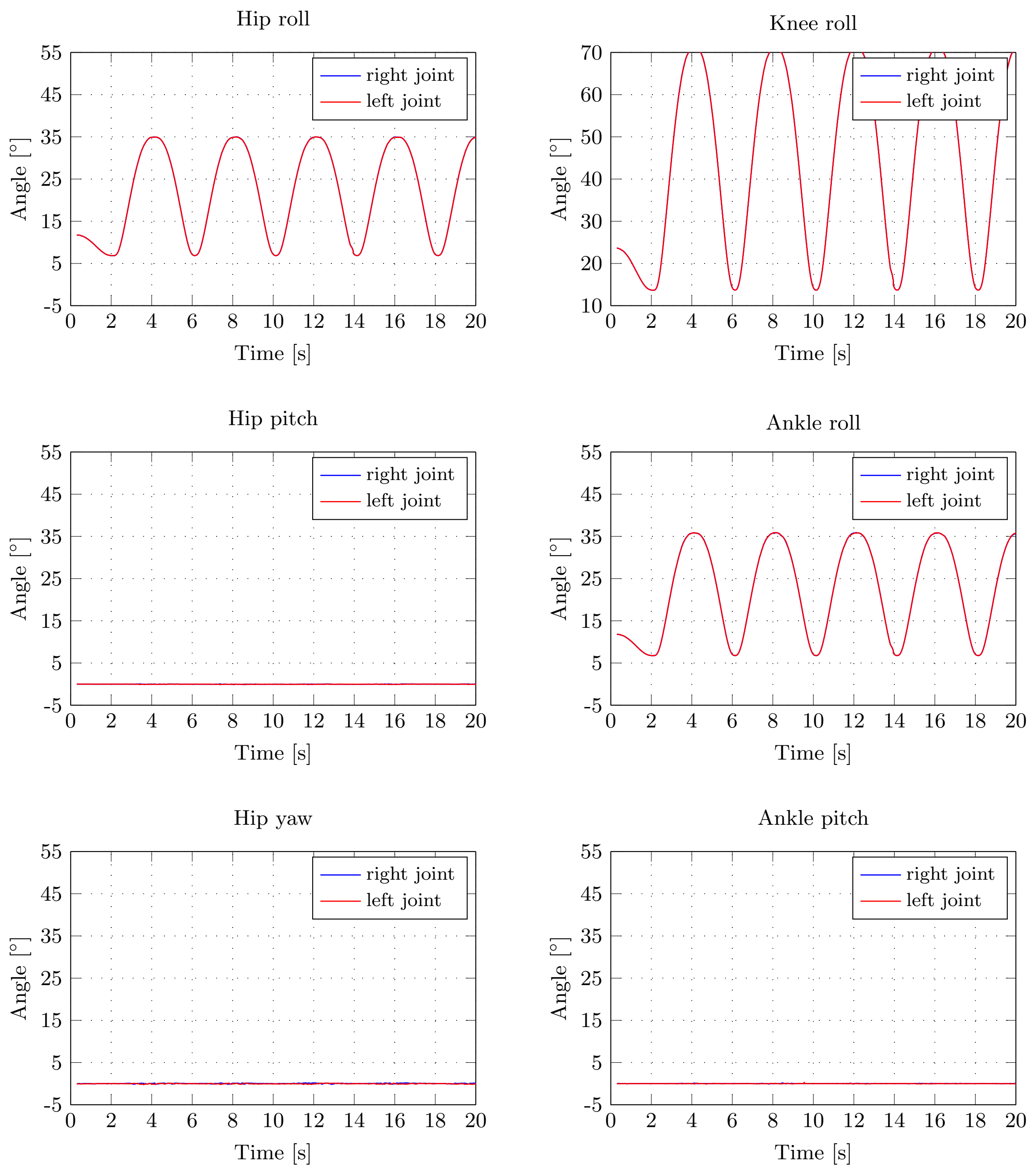


Figure 5.20: Robot's joint state evolution during the vertical free-movement simulation.

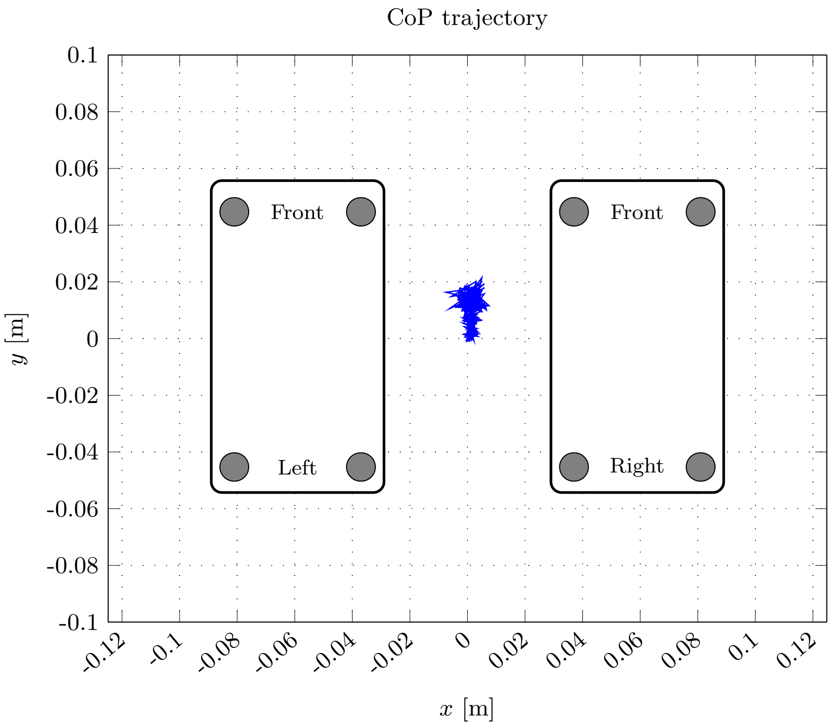


Figure 5.21: Complete trajectory of the CoP point during the vertical free-movement simulation.

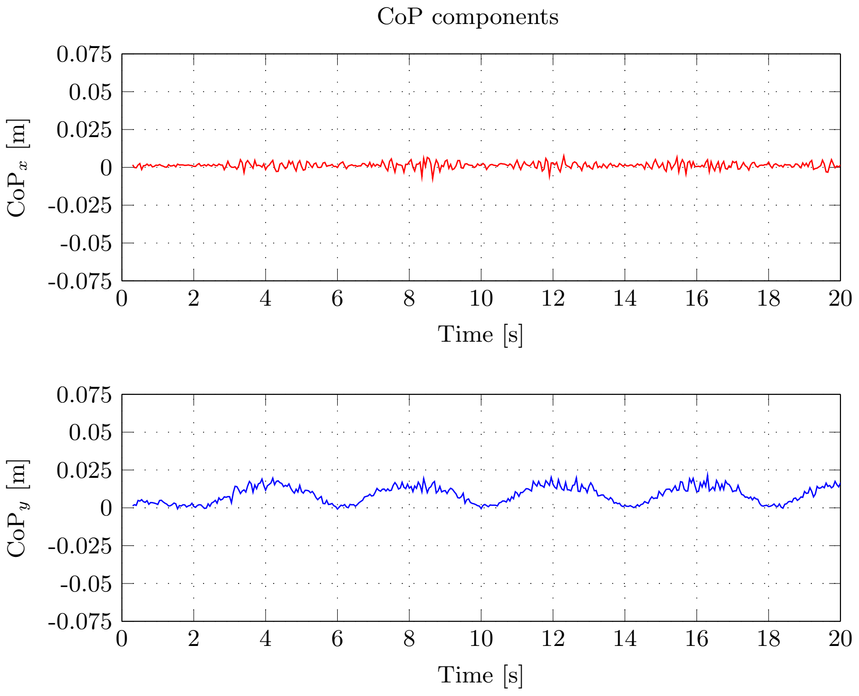


Figure 5.22: Bi-dimentional CoP components variation during the vertical free-movement simulation.

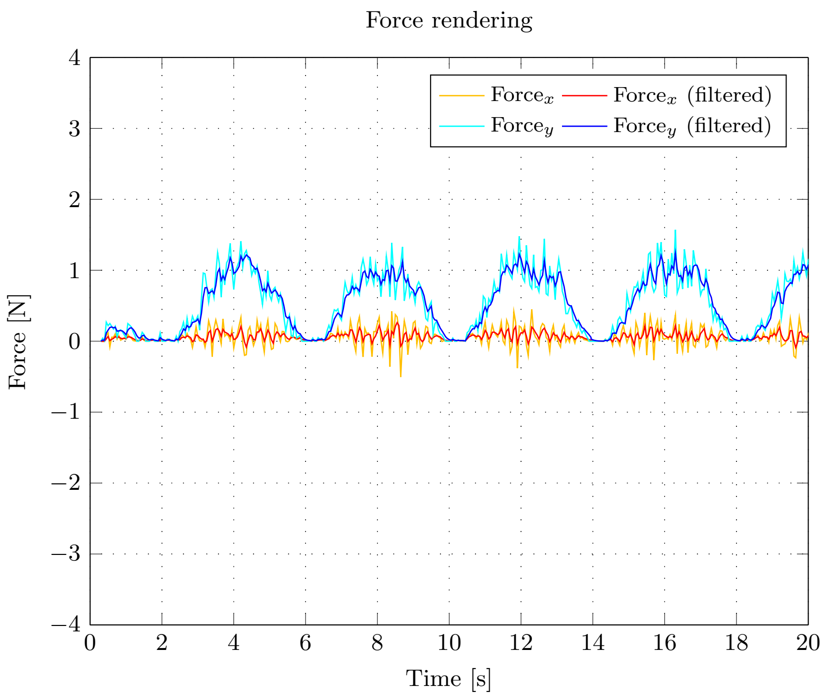


Figure 5.23: Force rendered by the haptic device during the vertical free-movement simulation.

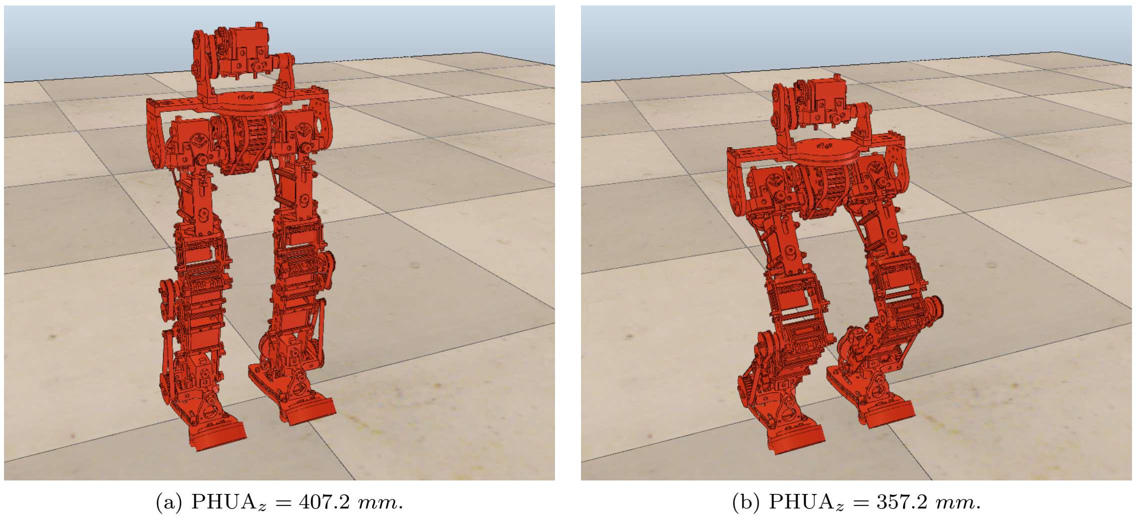


Figure 5.24: Extreme positions ocuppied by the robot during the vertical free-movement simulation.

5.2.2 Robot's Hip Teleoperation

This set of simulations involves the actual teleoperation of the model. The hip's motion is directly controlled by the haptic interface, which reproduces the operator movements onto the simulation scene. Initially, the joystick commands are restricted to one single axis, with concern for results comparison with the non-teleoperated experience. Yet again, the feedback generation uses the early described method.

Hip's lateral movement

The background results of this experiment are very similar to those exposed on the previous test. Despite the inconstancy of the movement itself, different from the clean periodic motion, the system response to the operator's command is surprisingly fast and effective. Figure 5.25 shows a continuous alternation on the movement's direction caused by a intentional irregular teleoperation. The main goal was to prove the reliability of the model when under real teleoperation conditions, characterized by fast and unexpected movements to keep the robot's balance.

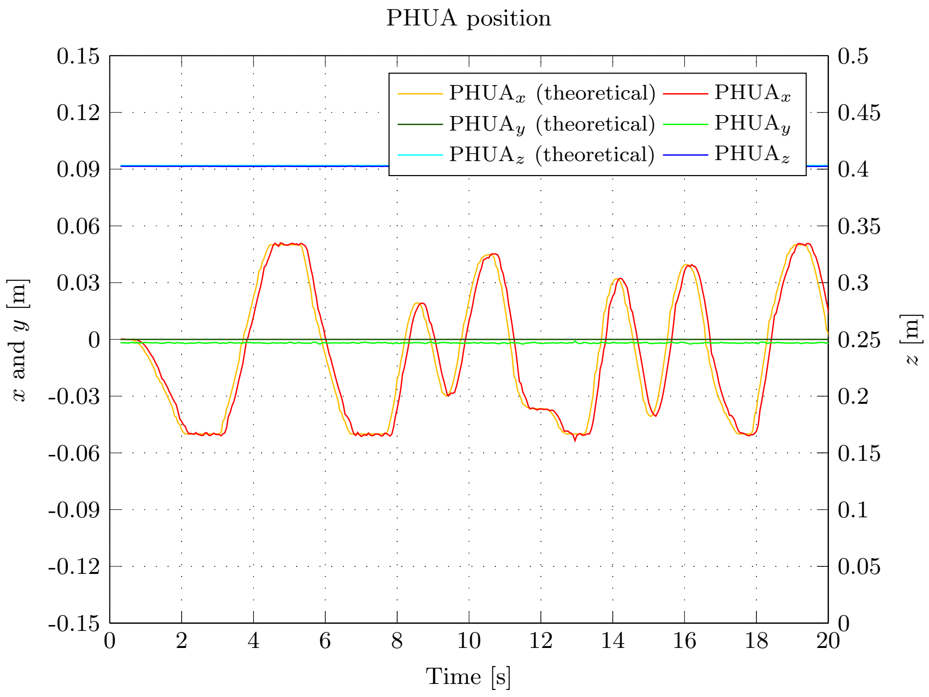


Figure 5.25: Model's hip position during the teleoperation in the lateral plane.

Again, the motion is characterized by the ankle's and hip's pitch joint variations, precisely described in Figure 5.26, and by a strong fluctuation of x component of CoP. The inverse kinematics limits imposed previously are enforced to the hip's motion once more, limiting the CoP excursion. However, as in the preceding experiment, when the hip reaches the extreme positions (Figure 5.30), the CoP follows accordingly (Figures 5.27 and 5.28).

The feedback force generated, illustrated in Figure 5.29, truly reflects this abrupt variation in the CoP position during the simulation.

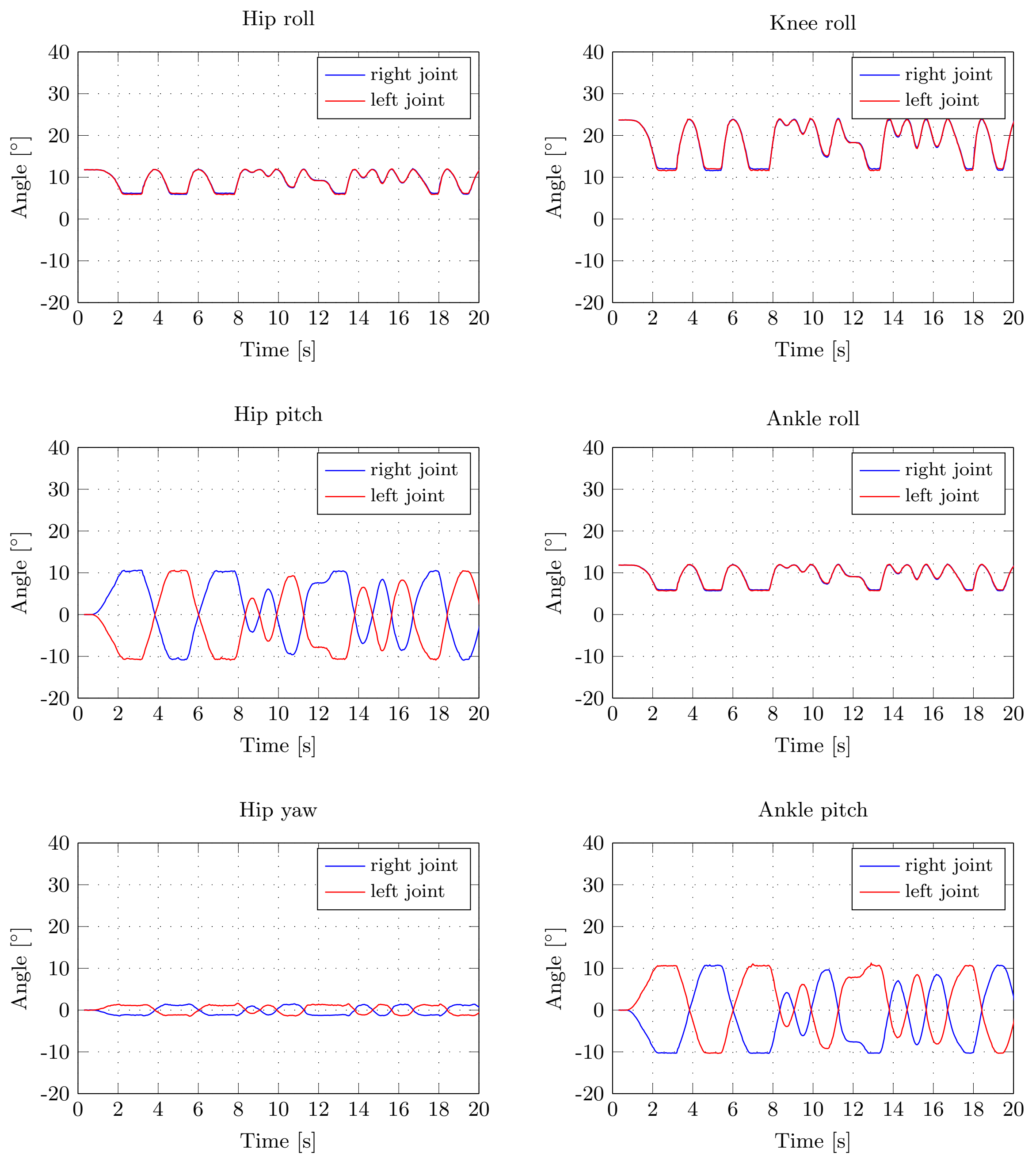


Figure 5.26: Robot's joint position evolution during the teleoperation in the lateral plane.

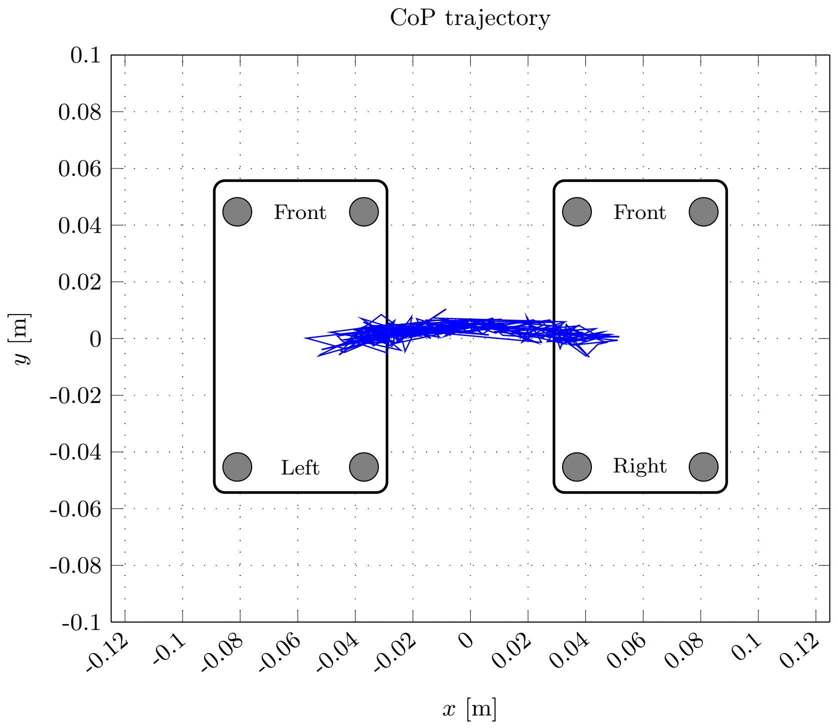


Figure 5.27: Complete trajectory of the CoP point during the teleoperation in the lateral plane.

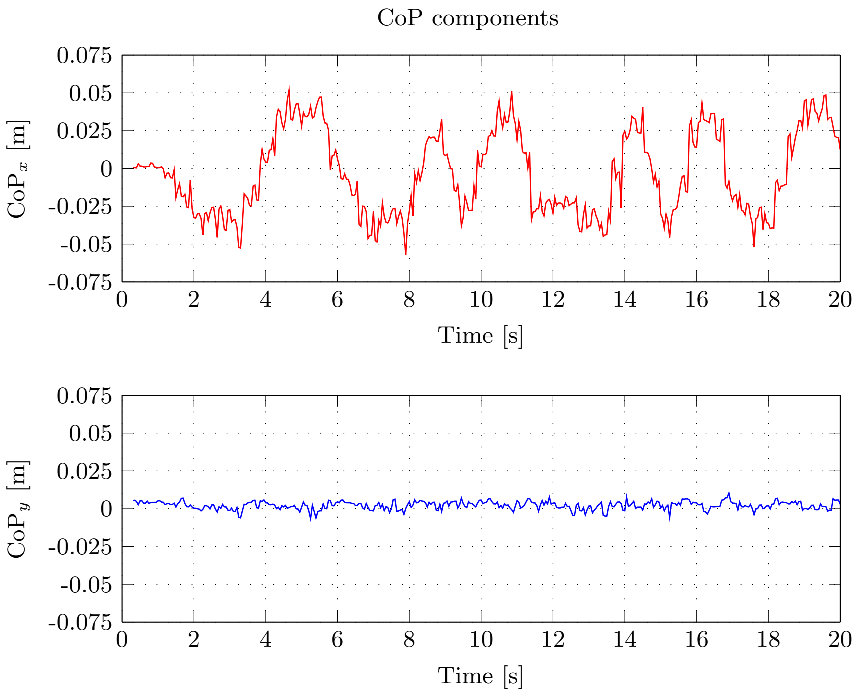


Figure 5.28: Bi-dimentional CoP components variation during the teleoperation in the lateral plane.

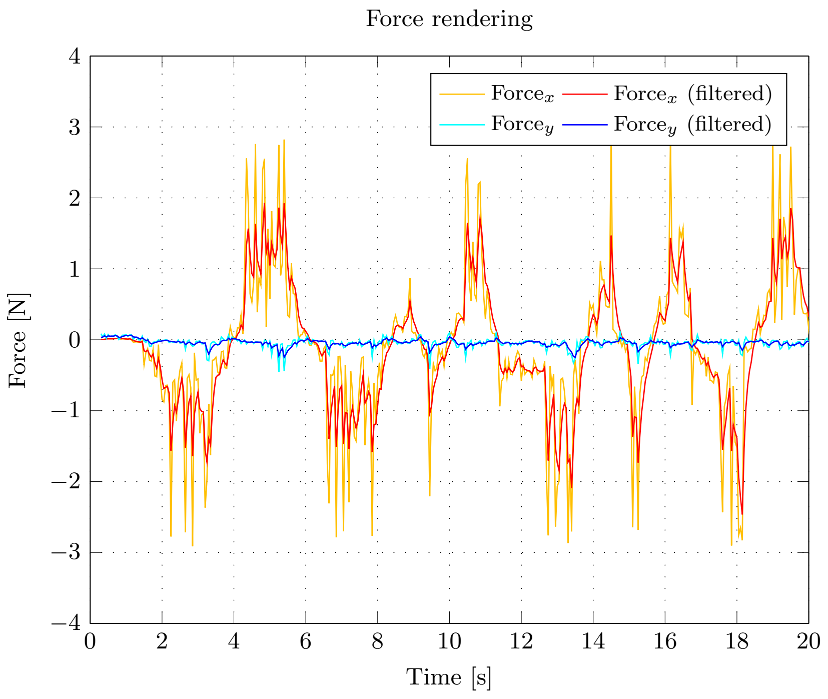
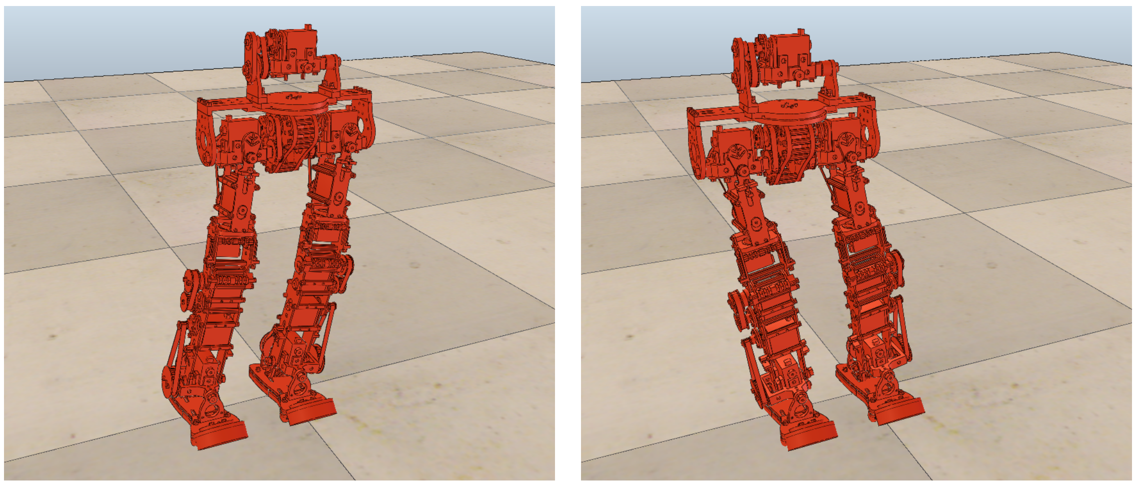


Figure 5.29: Force rendered by the haptic device during the teleoperation in the lateral plane.



(a) PHUA$_x$ = -50 mm. (b) PHUA$_x$ = 50 mm.

Figure 5.30: Extreme positions ocuppied by the robot during during the teleoperation in the lateral plane.

Hip's sagittal movement

In this demonstration the user teleoperates the model moving the hip along the sagittal plane, YOZ, represented in Figure 5.38. Only the joystick movements along this direction are applied to the robot. The motion described in Figure 5.31 reflects again the randomness of the teleoperation, to which the simulation responds positively.

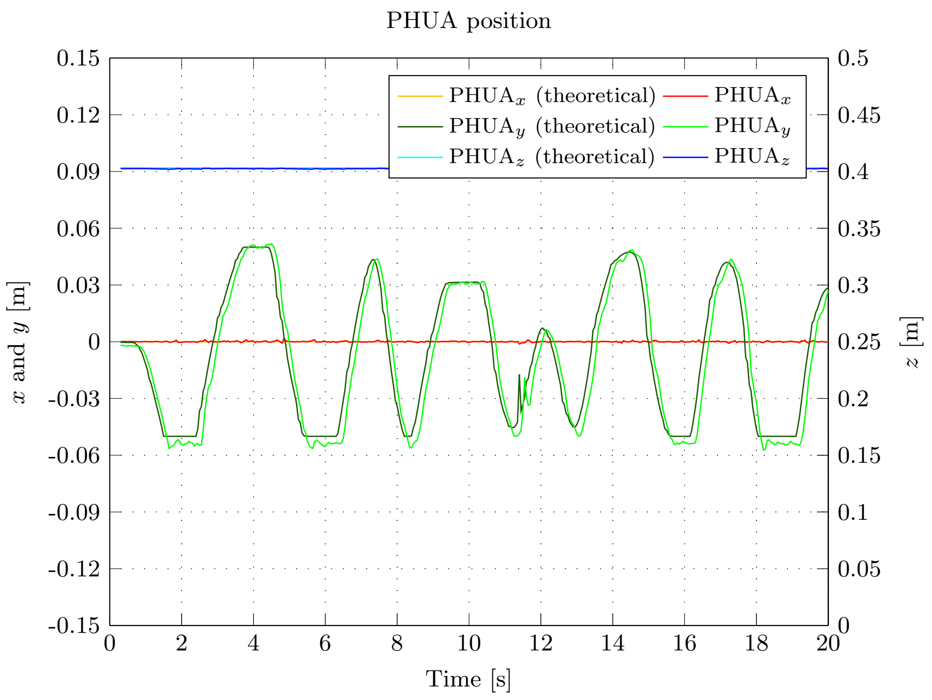


Figure 5.31: Model's hip position during the teleoperation in the sagittal plane.

During some time intervals the hip was kept still and then quickly actuated in the opposite direction, in order to study the force generation and the model's reaction. Around second 12, the user performed some quick and small movements in the negative zone (referring to feet coordinate system), pushing the robot back and forth relatively fast. This led to a noisier variation of the y component of CoP, as shown in Figures 5.35 and 5.36, which directly influenced the feedback force, as can be seen in Figure 5.37. As a matter of fact, the whole simulation is characterized by significant peaks of force, produced by the continuous and fast variation of CoP. Although the model's response to the user's input is fairly acceptable, the force peaks developed during these movements can truly harm the teleoperation process.

For this experiment, graphical representations of joint torque and velocity are also included. Besides the actuation basics, joint torques and velocity are commonly used to evaluate robotic motion and study the energy consumption of dedicated tasks. With V-REP it is also possible to have access to this information. Figures 5.33 and 5.34 illustrate the torque and velocity state over the simulation time, substantiating the behavior described by the joint positions in Figure 5.32. However, the torque and velocity charts show a particular detail. Contrary to expectations, the torque values of the hip roll joints are clearly close to zero, despite the huge variation on the velocity of the same joints. This is due to the absence of the upper body parts, during the simulations. Thus, considering the only mass above the hips is connected to the pelvis, given its parallel movement to the ground and the symmetrical distribution of the part's mass, the torque implicated in sagittal motion should equal zero, since the lever-arm distance vector is null.

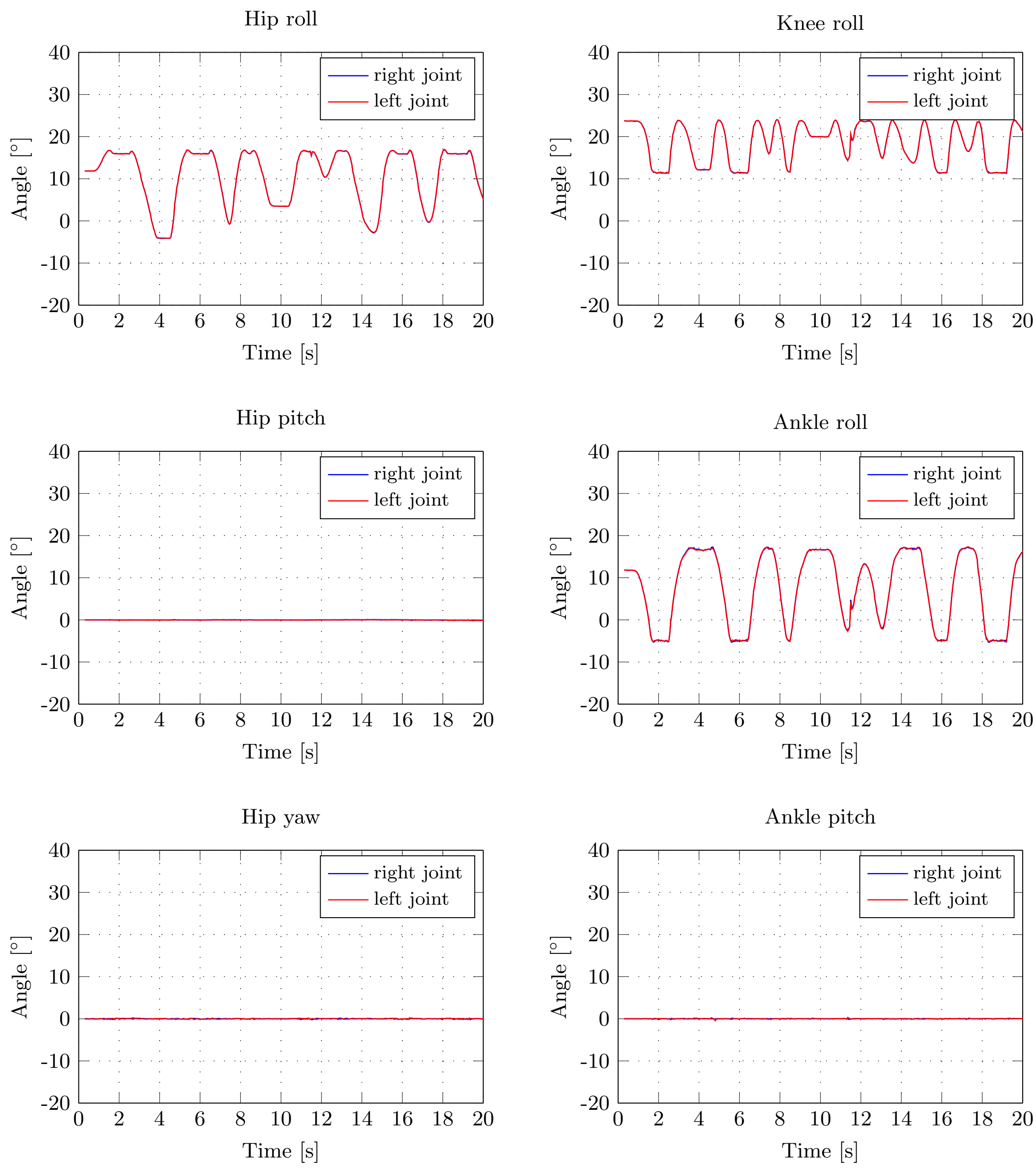


Figure 5.32: Robot's joint state evolution during the teleoperation in the sagittal plane.

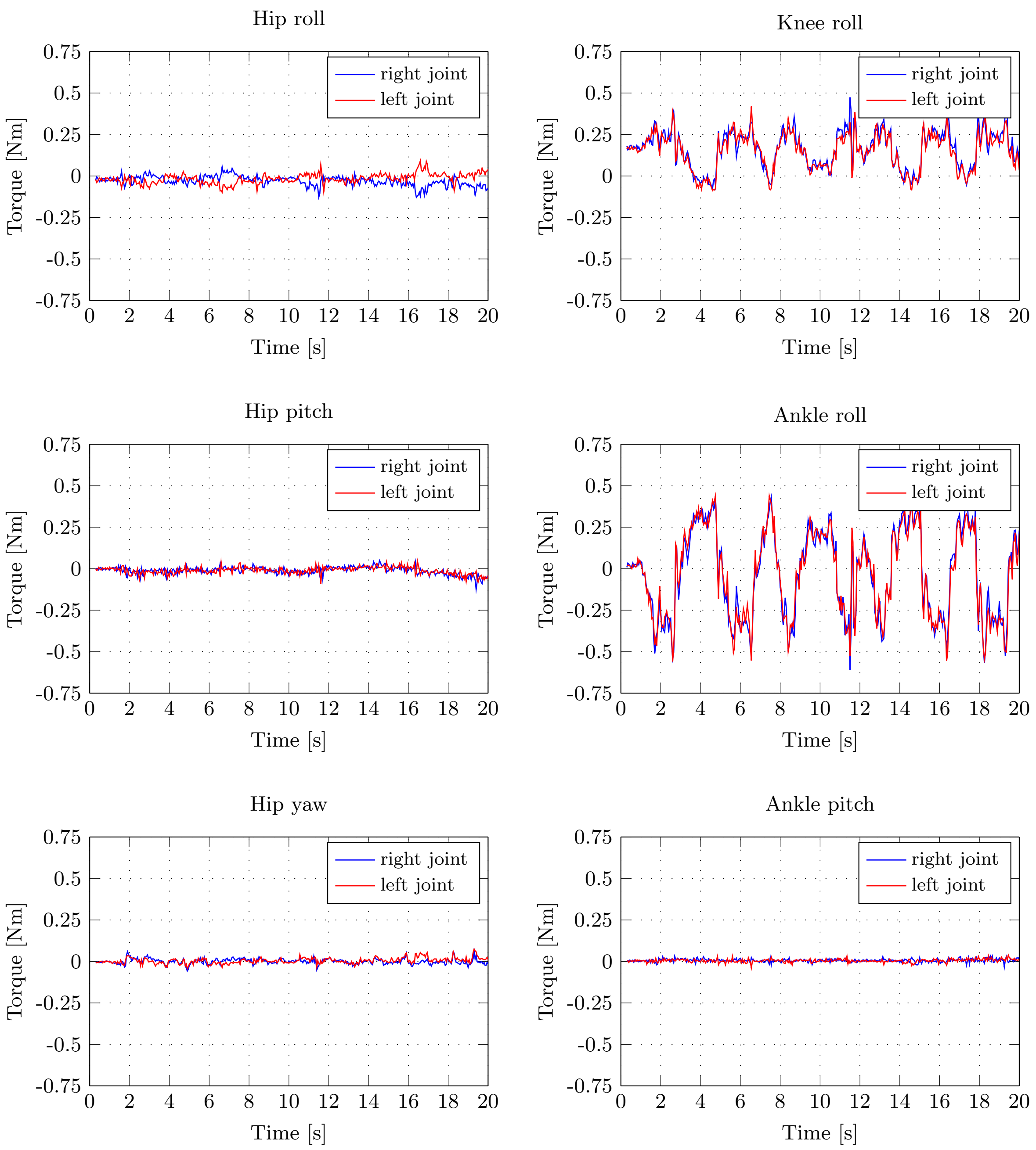


Figure 5.33: Robot's joint torque evolution during the teleoperation in the sagittal plane.

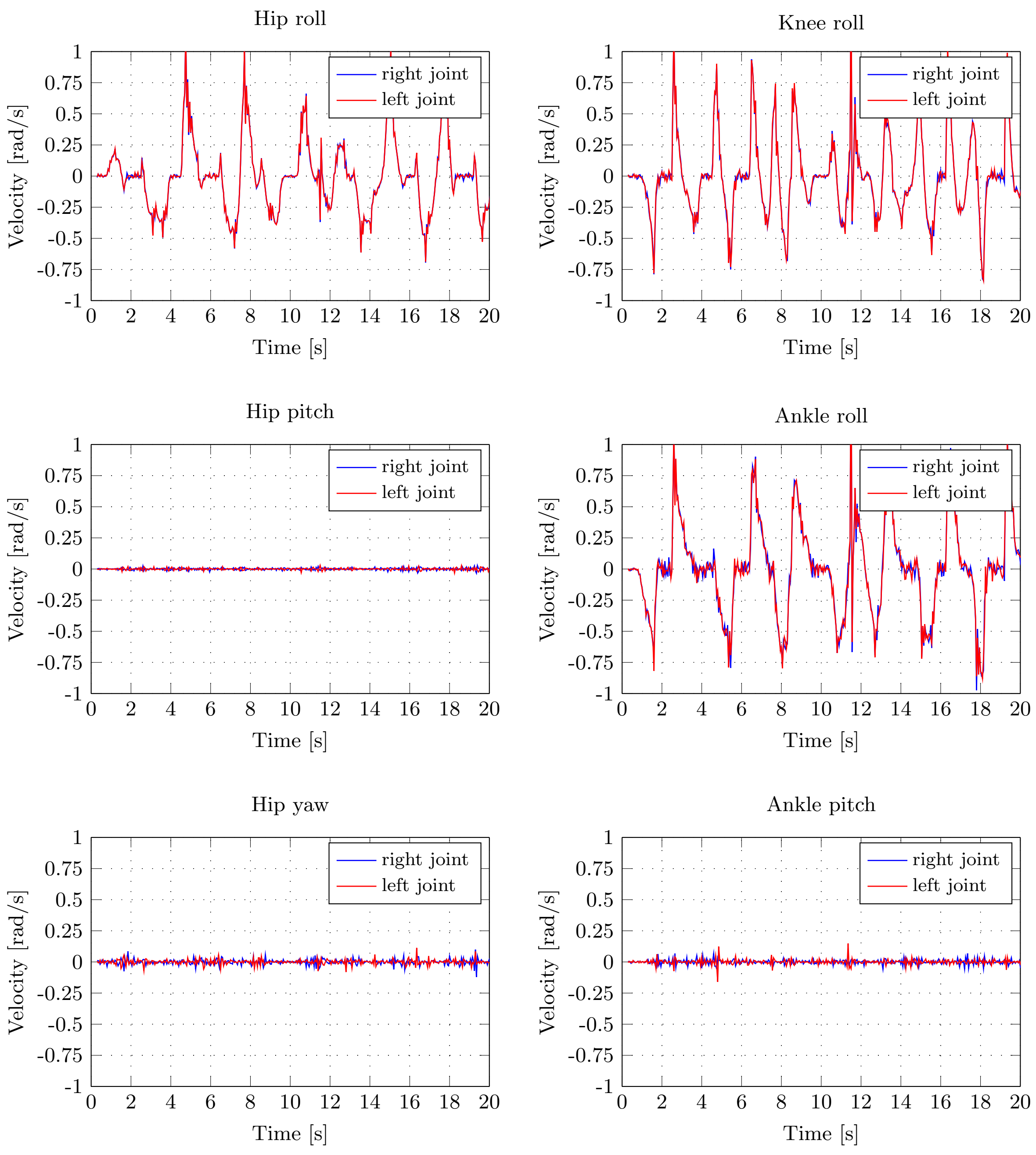


Figure 5.34: Robot's joint velocity evolution during the teleoperation in the sagittal plane.

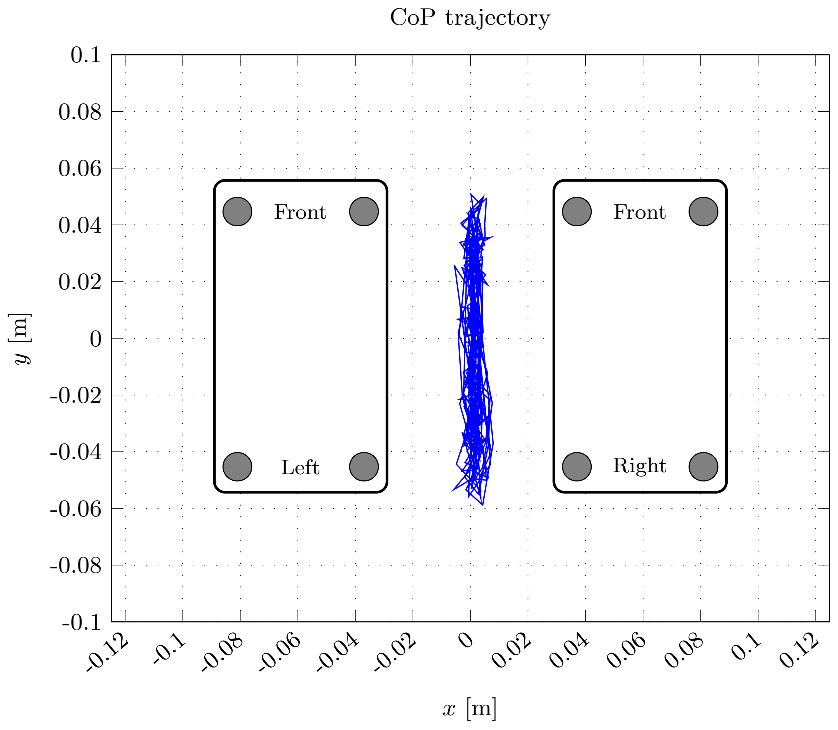


Figure 5.35: Complete trajectory of the CoP point during the teleoperation in the sagittal plane.

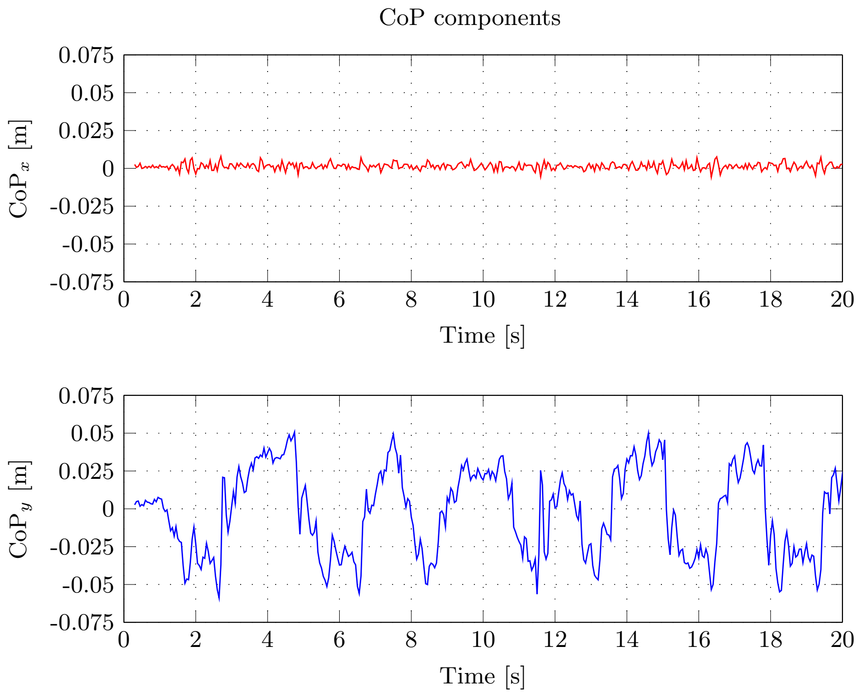


Figure 5.36: Bi-dimentional CoP components variation during the teleoperation in the sagittal plane.

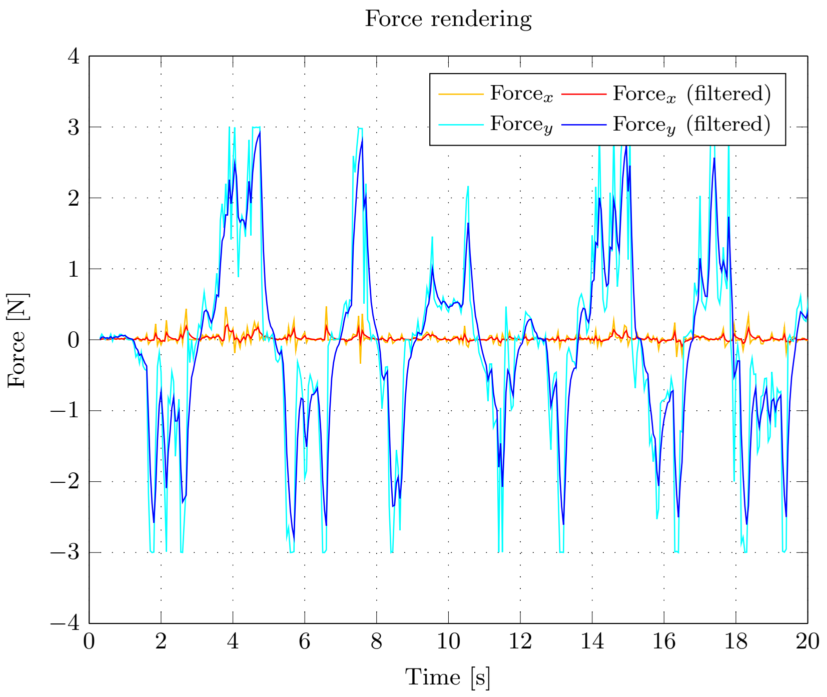
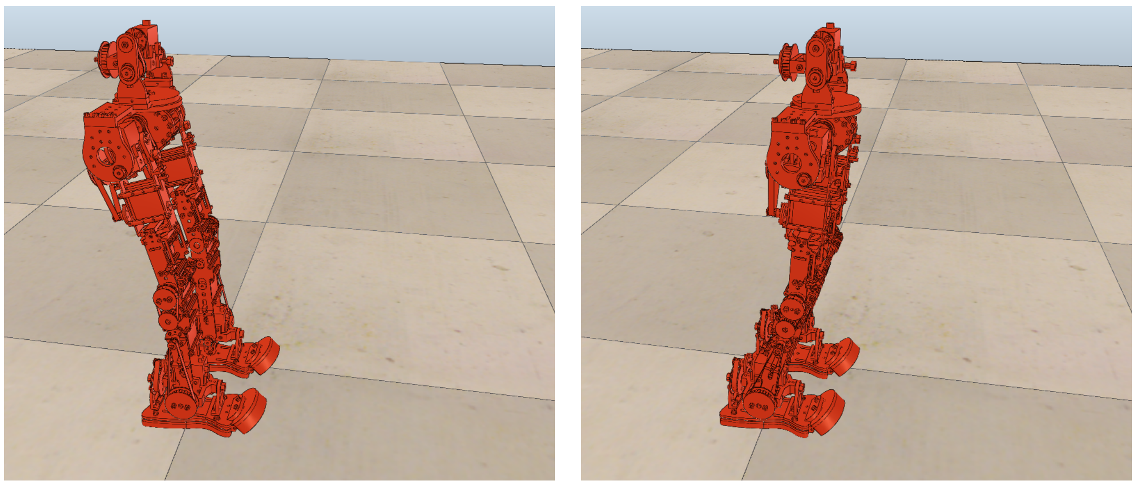


Figure 5.37: Force rendered by the haptic device during the teleoperation in the sagittal plane.



(a) PHUA$_y$ = -50 mm.

(b) PHUA$_y$ = 50 mm.

Figure 5.38: Extreme positions ocuppied by the robot during during the teleoperation in the sagittal plane.

Hip's vertical movement

As anticipated, this motion reveals a very similar behavior to the polynomial trajectory, even considering the irregular position variations introduced in the simulation, represented in Figure 5.39. The up and down movements verified are easily confirmed by the ankle, hip, and mainly knee joint values presented in Figure 5.40. Torque values are also represented in Figure 5.41. The results presented for the torque measures have the same explanation, since this movement involves hip roll joints actuation.

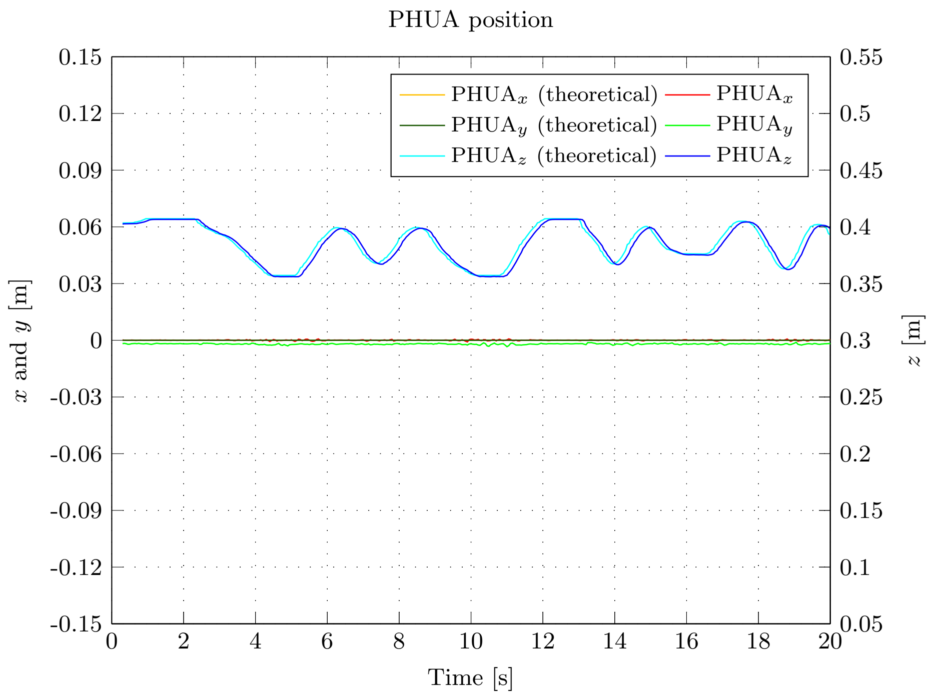


Figure 5.39: Model's hip position during the teleoperation in the vertical plane.

Giving the small amplitude of the hip's movement, $z = [0.3572, 0.4072]$ (representation available in Figure 5.45), the CoP deviation from the origin is also small, about 2 centimeters (Figures 5.42 and 5.43). This had an almost insignificant impact on the force rendering, which had a maximum value of approximately 1.5 N, even affected by a scale factor ($\times 3$), as shown in Figure 5.44. However, these tiny fluctuations on the CoP location are perfectly expected. The robot moves up and down keeping the pelvis parallel to the ground, while no lateral or sagittal motion is predicted. In the absence of the upper part of the body, which could lead to an additional moment about the feet, the robot remains virtually stable.

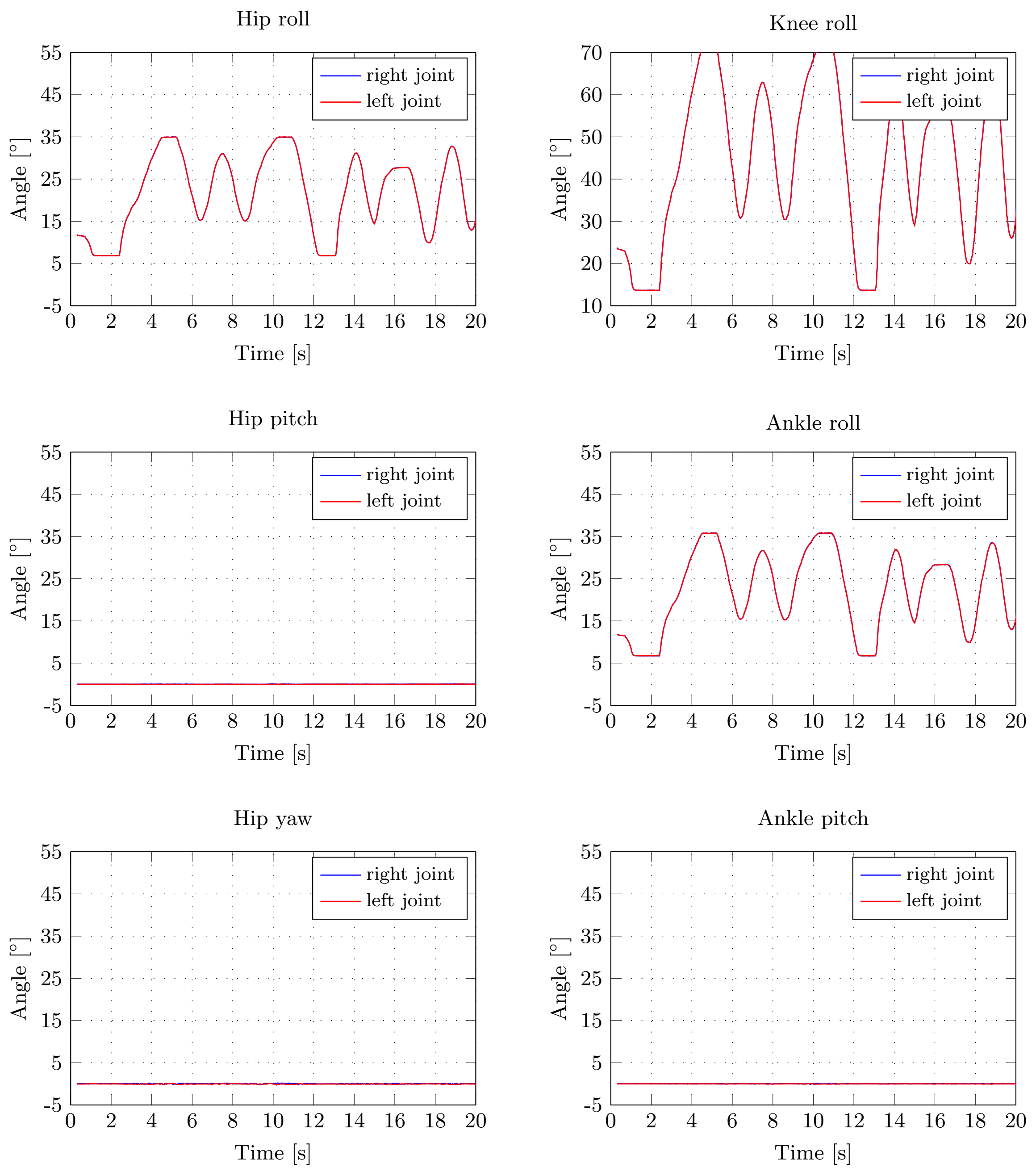


Figure 5.40: Robot's joint state evolution during the teleoperation in the vertical plane.

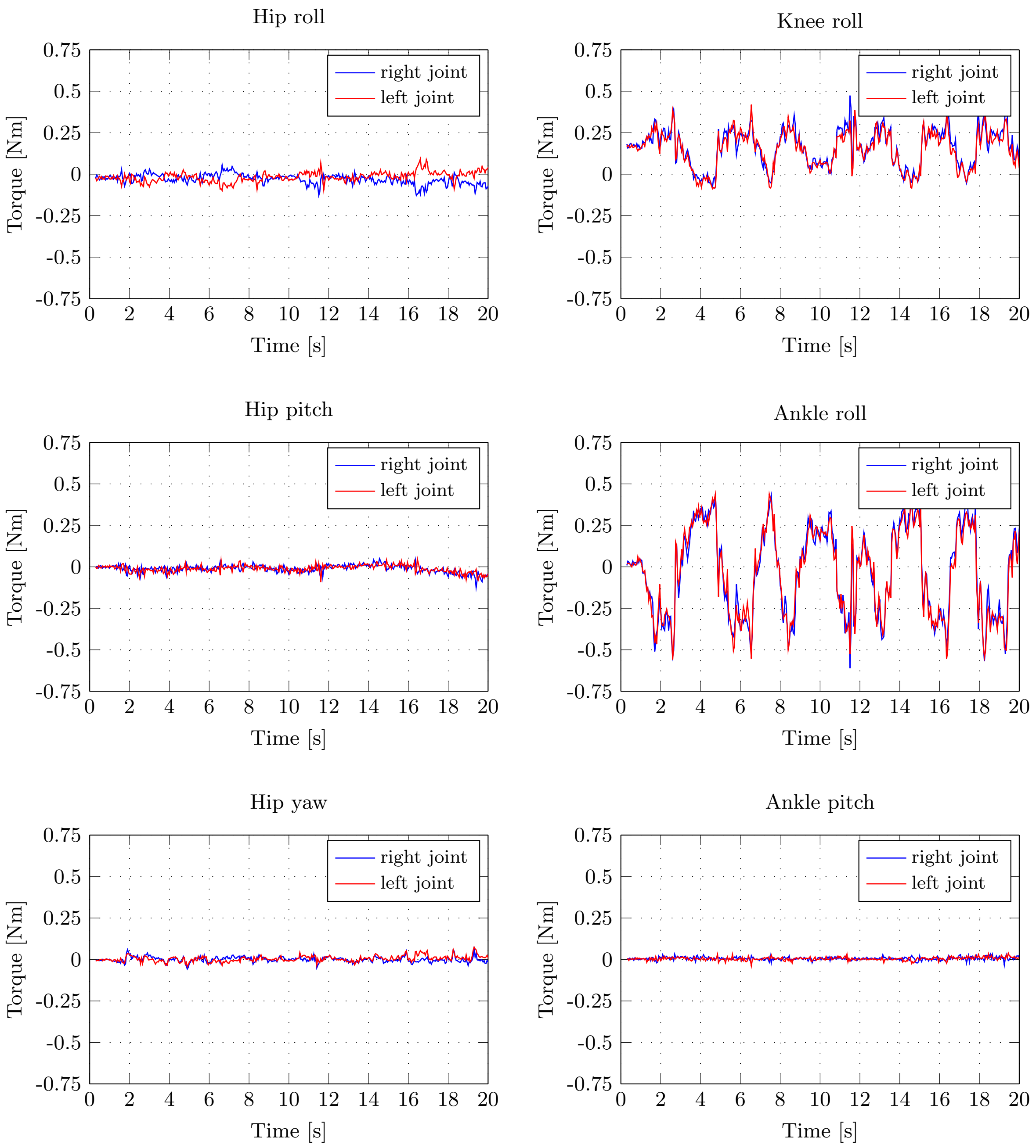


Figure 5.41: Robot's joint torque evolution during the teleoperation in the vertical plane.

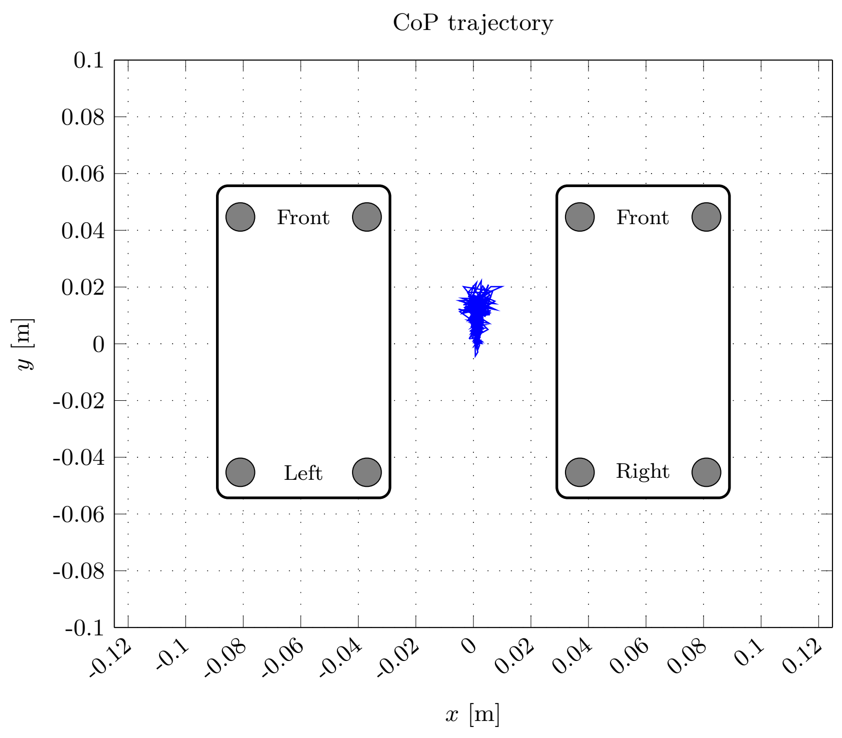


Figure 5.42: Complete trajectory of the CoP point during the teleoperation in the vertical plane.

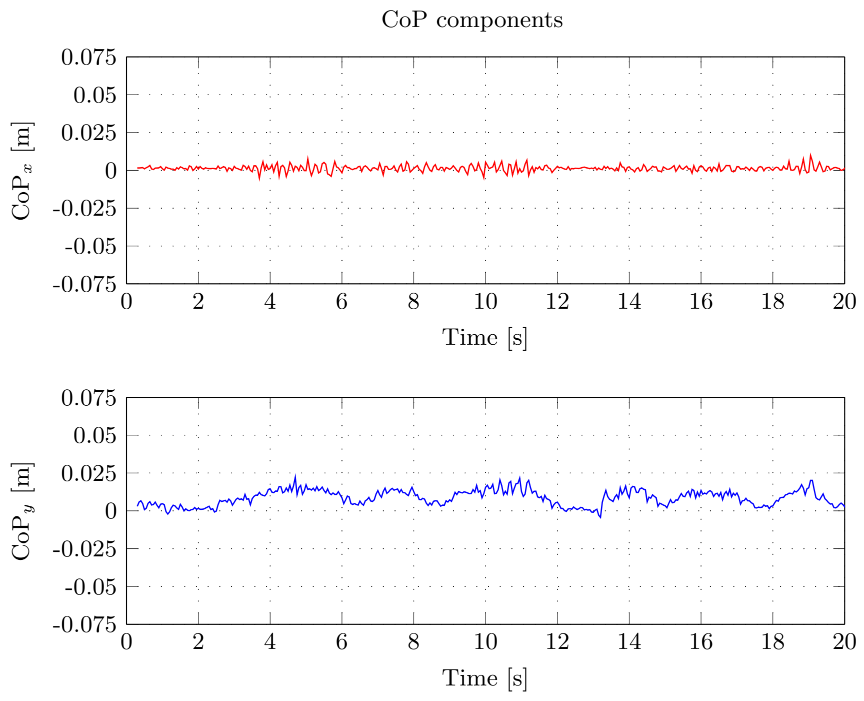


Figure 5.43: Bi-dimentional CoP components variation during the teleoperation in the vertical plane.

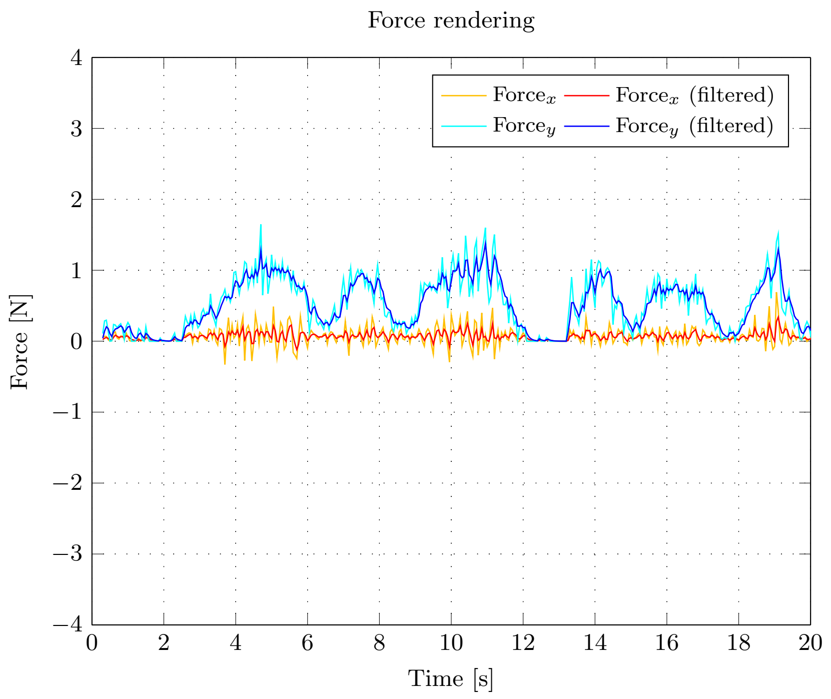
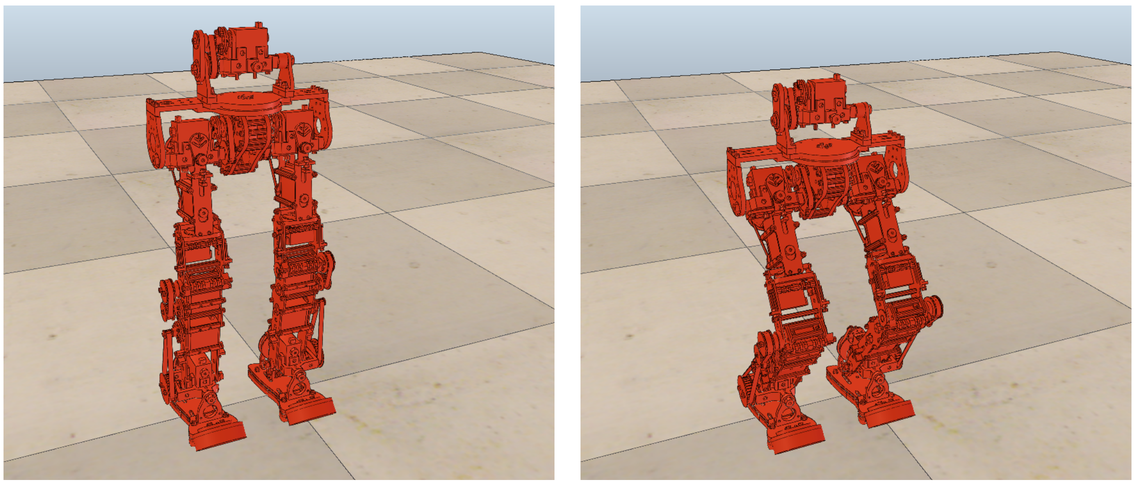


Figure 5.44: Force rendered by the haptic device during the teleoperation in the vertical plane.



(a) PHUA$_z$ = 407.2 mm.

(b) PHUA$_z$ = 357.2 mm.

Figure 5.45: Extreme positions ocuppied by the robot during the teleoperation in the vertical plane.

5.2.3 Haptic Feedback in Postural Balance Maintenance

The last IK experiment demonstrates the usefulness of the haptic feedback approach as a postural balance strategy. During the hip's vertical movement, automatically reproduced by the polynomial function, again between 0.3572 and 0.4072 m (Figure 5.51), the user tries to restore the robot balance countering the feedback force generated by the joystick. The aim is to ensure the CoP position remains as close as possible to the most stable position: the feet central point.

A clear conclusion arises from the previous experiments where the vertical motion of the model was studied: during its descendant movement, the CoP tends to go slightly forward on the feet referential frame. To counter that, the operator has to push the robot in the opposite direction, meaning in the negative y direction, and thereby try to balance the dynamics created during the vertical development. This action is perfectly described by the green curves in Figure 5.46, which represent the user commands sent through the joystick to the simulation. The blue curves represent the z variation of the hip's motion generated by the polynomial function.

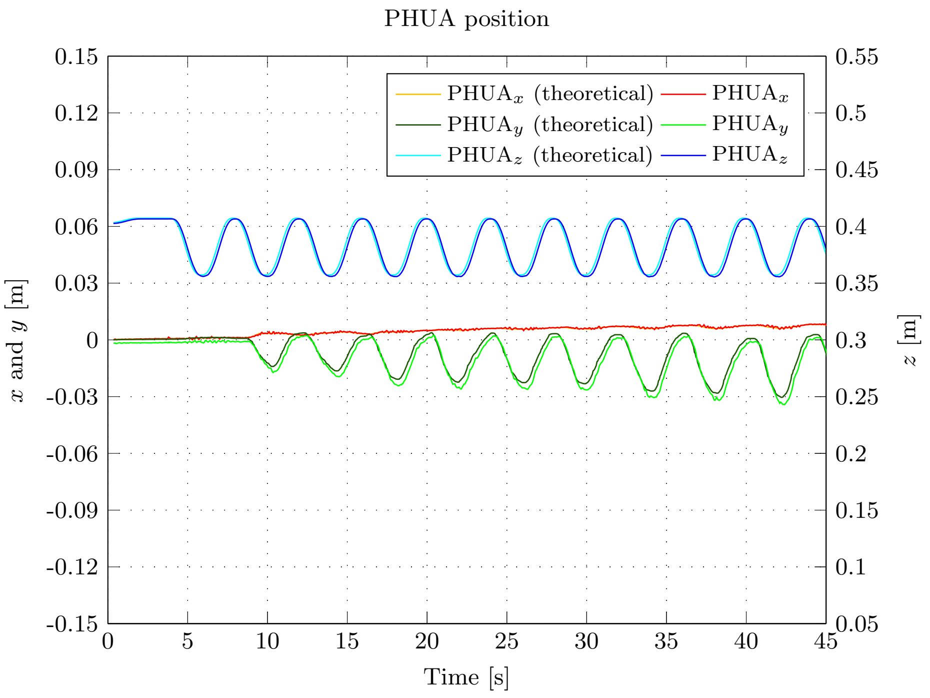


Figure 5.46: Model's hip position during the teleoperation task to compensate the CoP variation.

Ignoring the first seconds of simulation, where the operator is not actually performing any motion, the progress made over the simulation time puts the y component of CoP very close to zero, as can be observed in Figure 5.49, despite the little variation in the x direction, resulting from the user's interference, as can be perceived in the CoP trajectory in Figure 5.48. This proves not only the reliability of the simulator, whose dynamics module is capable of simulating real world conditions, but also the operator skills in the teleoperation. The "quality" of the teleoperation is clearly demonstrated by the low force values verified during the simulation, showing the robot's balance was always ensured, with the exception of the teleoperation start, at $t = 5$ s (Figure 5.50).

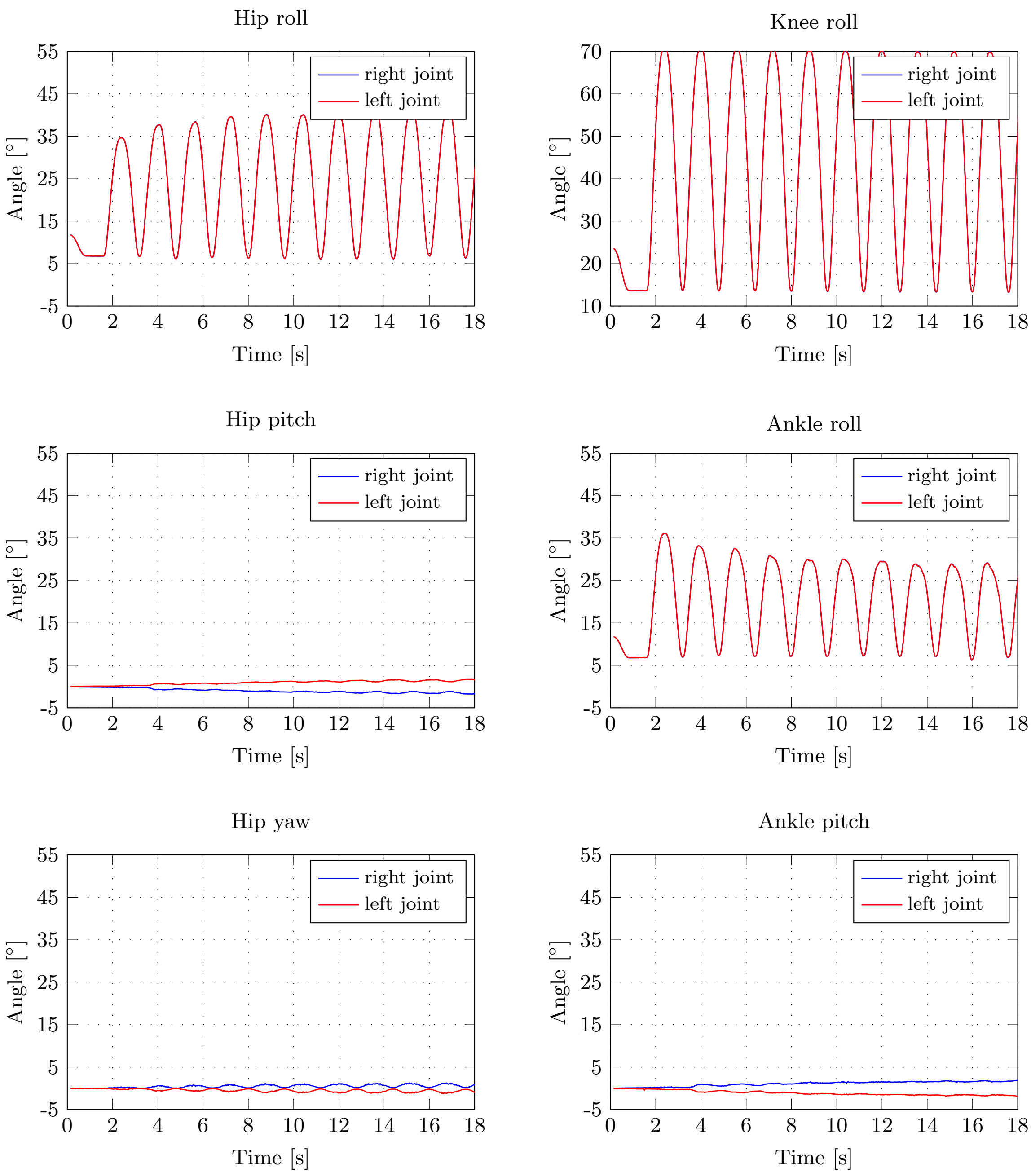


Figure 5.47: Robot's joint state evolution during the teleoperation task to compensate the CoP variation.

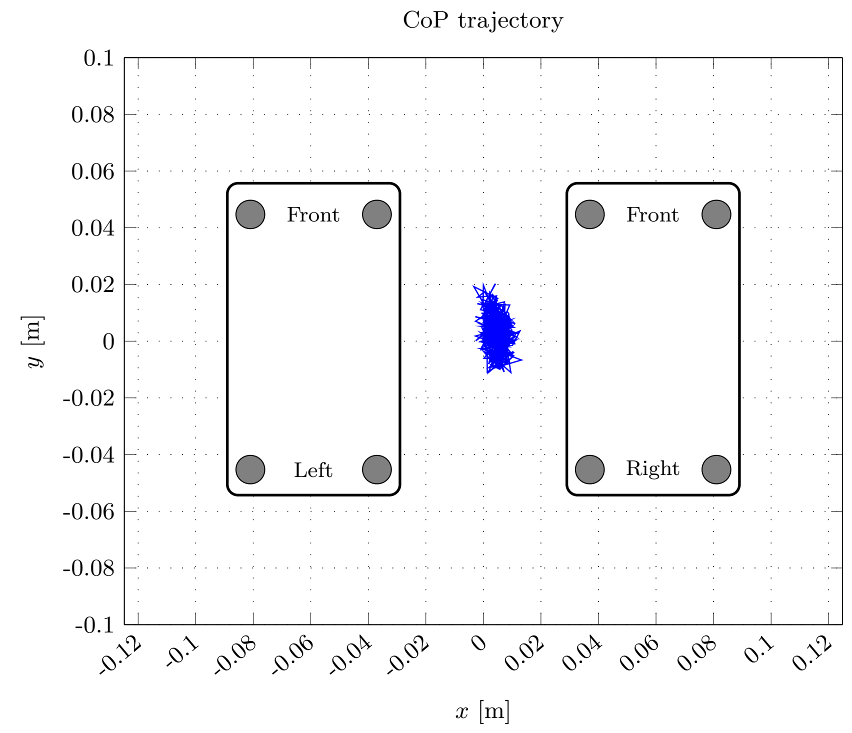


Figure 5.48: Complete trajectory of the CoP point during the teleoperation task to compensate the CoP variation.

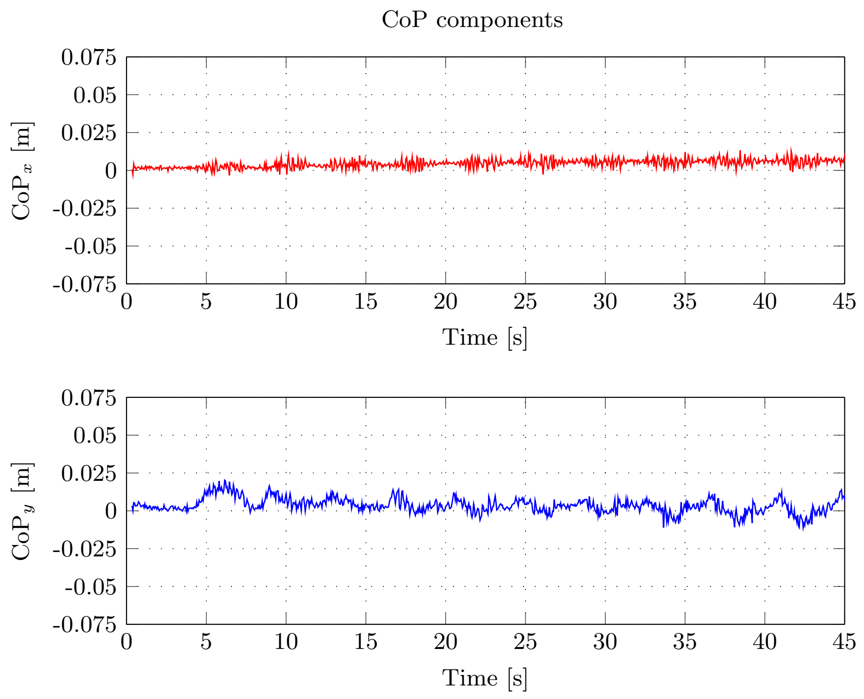


Figure 5.49: Bi-dimentional CoP components variation during the teleoperation task to compensate the CoP variation.

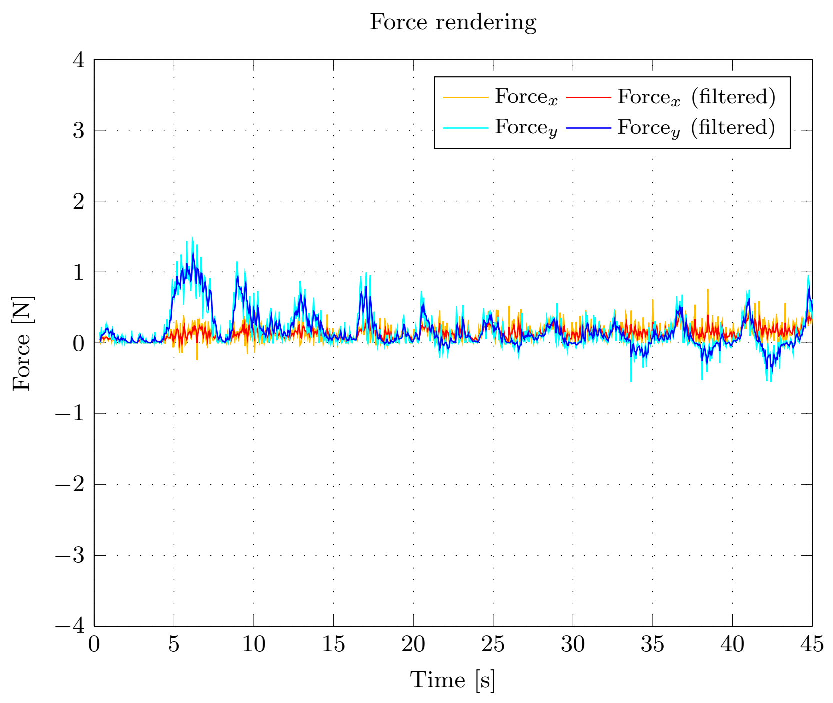
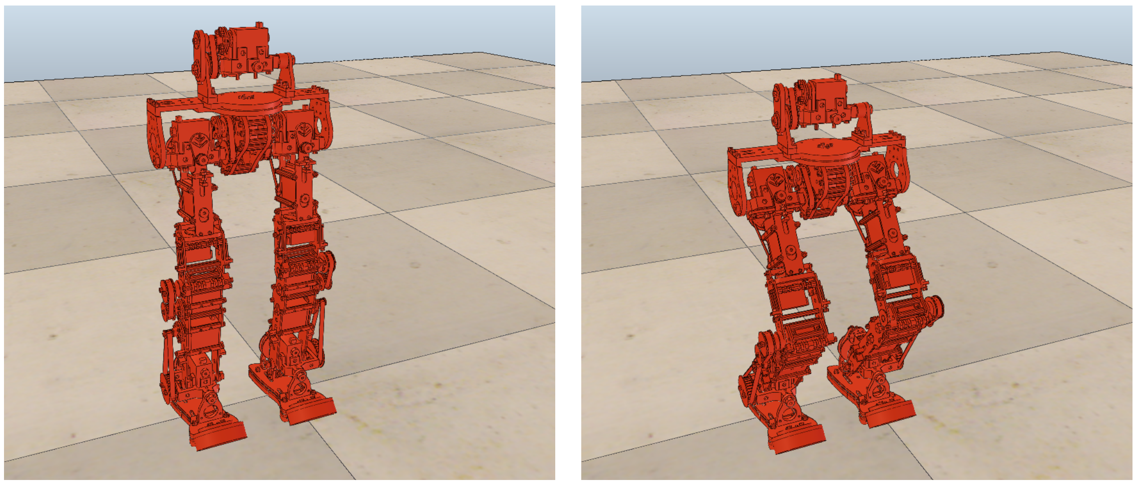


Figure 5.50: Force rendered by the haptic device during the teleoperation task to compensate the CoP variation.



(a) PHUA$_z$ = 407.2 mm.

(b) PHUA$_z$ = 357.2 mm.

Figure 5.51: Extreme positions ocuppied by the robot during the teleoperation task to compensate the CoP variation.

5.3 Torque Mode

The joint-by-joint control, designation commonly assigned to torque/force control mode, describes the joint actuation at an individual level. In some grassroots experiments, the operator applies this strategy to control the two legs separately, but at the same time. This is done using the dual haptic joystick configuration (Figure 5.52).

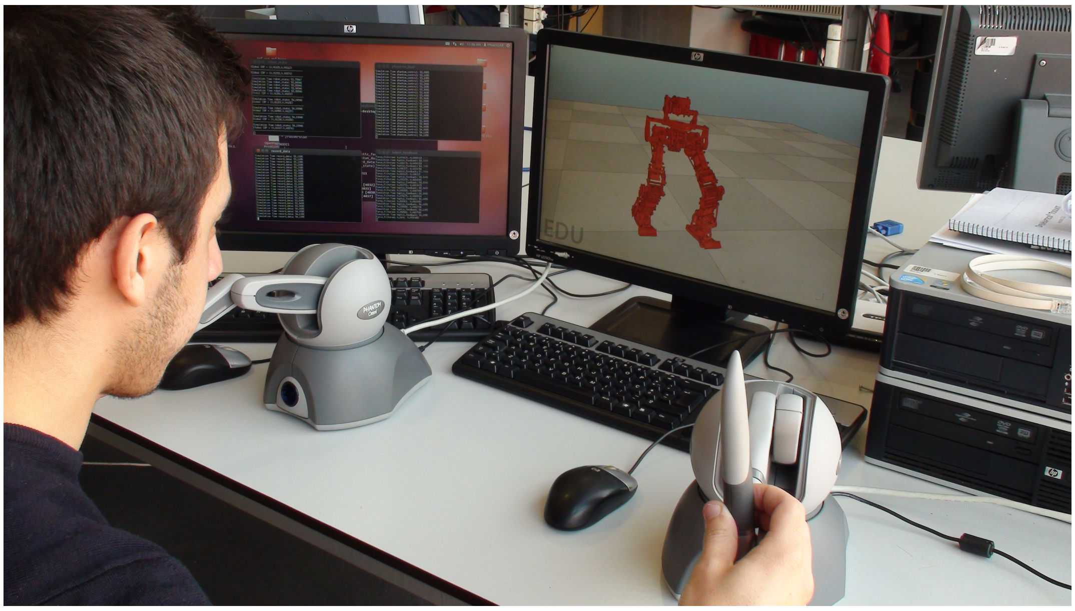


Figure 5.52: Example of a teleoperation task in torque mode.

To reach this objective, each joystick defines the joint state to be sent to each robot leg, serving as an interface to model control. The force feedback is generated concurrently. While the user is controlling the model using the torque control method, the system is monitoring its respective state, and a external force is synthesized to signal the operator when the robot is losing its balance. This is accomplished in much the same way as the IK mode, through the force rendering algorithm already described.

These experiments are an exploratory first proposal in which, above all, it is intended to enhance the user's integration with this methodology. The first attempts aim at familiarizing the user with the robot and the control strategy itself, by accomplishing simple equilibrium tasks and basic movements.

The results here presented are part of a single simulation in which the user tries to perform different teleoperation tasks. This demonstration is subdivided into five sections, representing the five different stages of the simulation:

- Lateral and sagittal motions of the entire body.
- Vertical motion of the entire body.
- Single leg lateral movement.
- Gait typical pattern.
- Double leg lateral movement.

During the first simulation steps, the robot moves along the lateral and the sagittal planes. However, instead of controlling the hip, the user controls the leg joints moving the two joysticks in parallel, that is, in the same direction. Keeping the two feet close to each other, the results are very much similar to those obtained previously. This can be confirmed by looking at the joint state evolution, in Figure 5.53, which first illustrates the lateral oscillation, and after the 20^{th} second, the sagittal movement. It should be noted that, since the robot legs are operated separately, the joint values presented are not equal in magnitude. This is, in turn, a characteristic feature of the dual/cooperative teleoperation.

An interesting fact about this simulation is that, despite the fact that the joystick base is fixed (the joystick is placed on the working table), when the user works with it, he's not controlling its tip, but its joints. Moving the two devices simultaneously, and in the same direction, will take the robot's hip to move instead of its feet. A behavior caused by the model's own weight.

In the next simulation stage the roll joints are actuated, and as a result, the robot bends its knees (Figure 5.54). Contrarily to the inverse kinematics experiments, all the robot joints are being controlled at the same time. Therefore, the user can have a hard time attempting to control and monitor the consequences of all his actions. As a result, undesired joints may be actuated during certain movements, as demonstrated in the graphical representations in Figure 5.55. This is a common issue of the dual teleoperation here implemented.

In the 3^{rd} phase of the experiment, the operator guides the robot to open one leg at a time in the lateral plane, while in the 5^{rd}, both legs move sideways. Due to robot's weight, and since the movement is not performed fast enough, the feet slide over the floor. In the pictures of the V-REP environment associated with these states, Figures 5.56 and 5.60, it can be perceived that the ankle joint depends on the hip joint values. As mentioned in the previous chapter, only the first three joints (hip's roll and pitch, and knee roll) are directly actuated. To the remaining joints, symmetrical values are applied. This ensures a more compliant contact with the ground, and eases the operator task, decreasing the degrees-of-freedom available to control.

In Figure 5.57 a large variation on the hip's and ankle's pitch can be observed which is characteristic of the lateral movement. From this data, one can identify two different stages: an early phase in which the right leg performs its motion, and a second phase where the right leg stands still and the left moves. In contrast, the information obtained for the simultaneous leg motion stage and represented in Figure 5.61, shows the hip's and ankle's curves closely following each other, as expected.

The 4^{th} stage of this simulation provides useful data on the biped typical gait pattern. Although none of the feet leaves the ground during the experiment (Figure 5.58), the alternate movement the user performs with the two joysticks (one leg forward and the other leg backwards), constitutes the first step in the introduction of more flexible ways of teleoperation. This alternation can be easily perceived by the roll joint values represented in Figure 5.59.

As a consequence of the unsatisfactory *jump* behavior verified during this simulation, the force rendering on the user was disabled. Anyhow, the resulting force was calculated, and the edited results are presented in Appendix A, for reference.

Lateral and sagittal movements

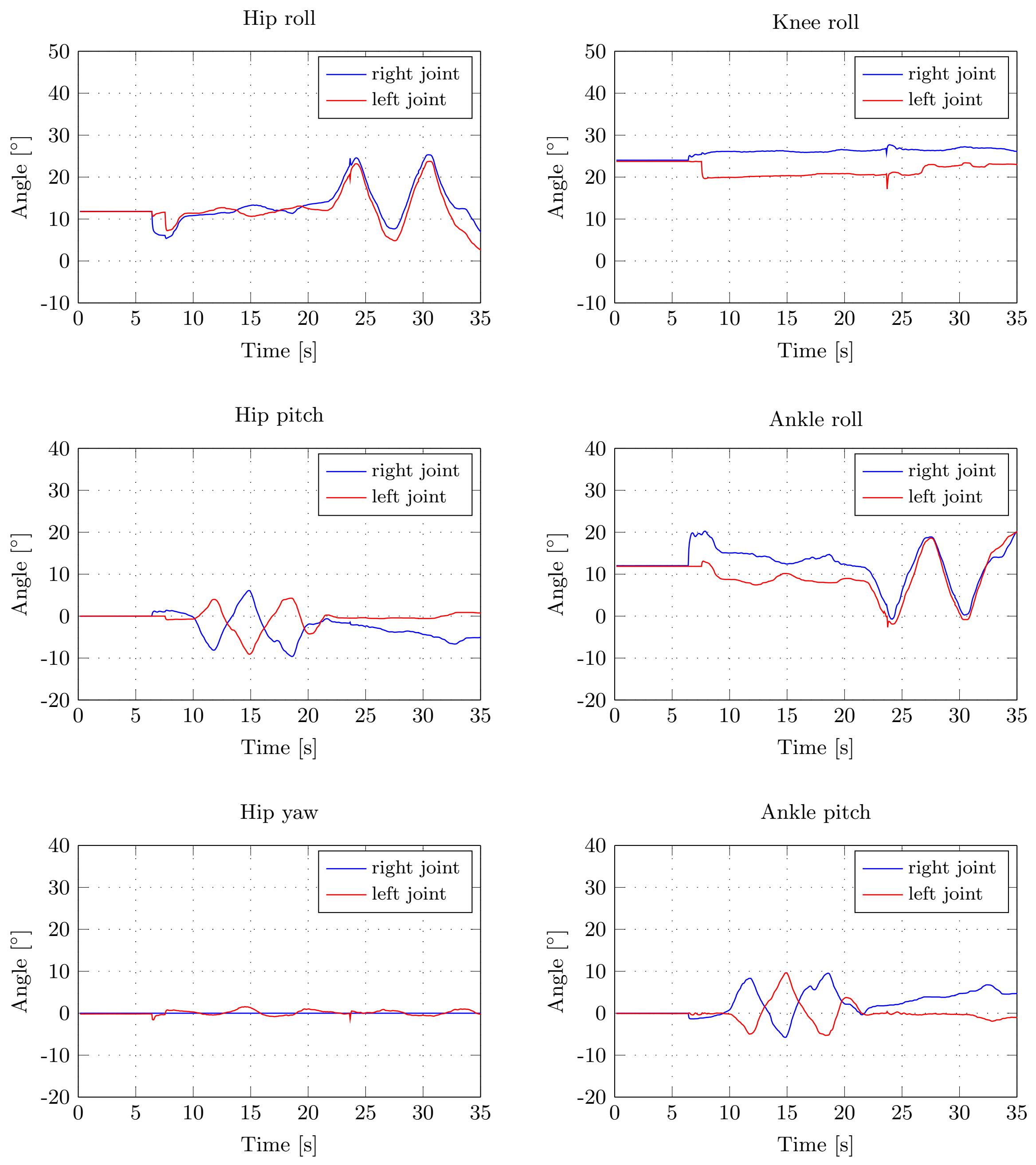
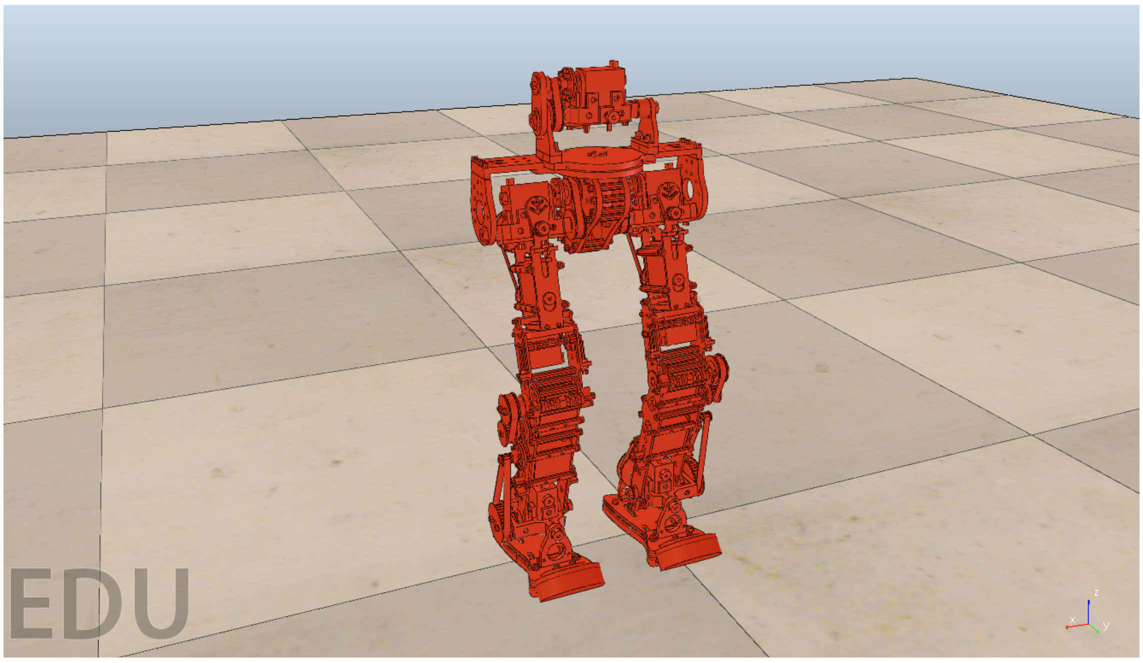
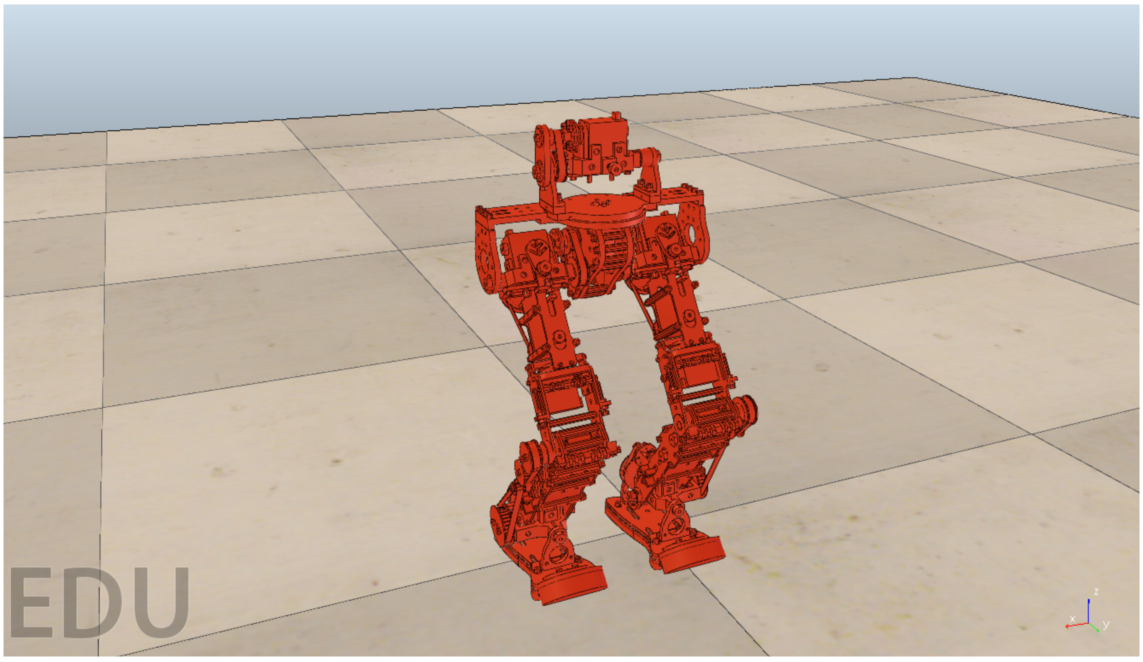


Figure 5.53: Robot's joint state evolution during the lateral and sagittal motion of the body.

Vertical movement



(a) Stable configuration.



(b) Lowest position.

Figure 5.54: Vertical movement.

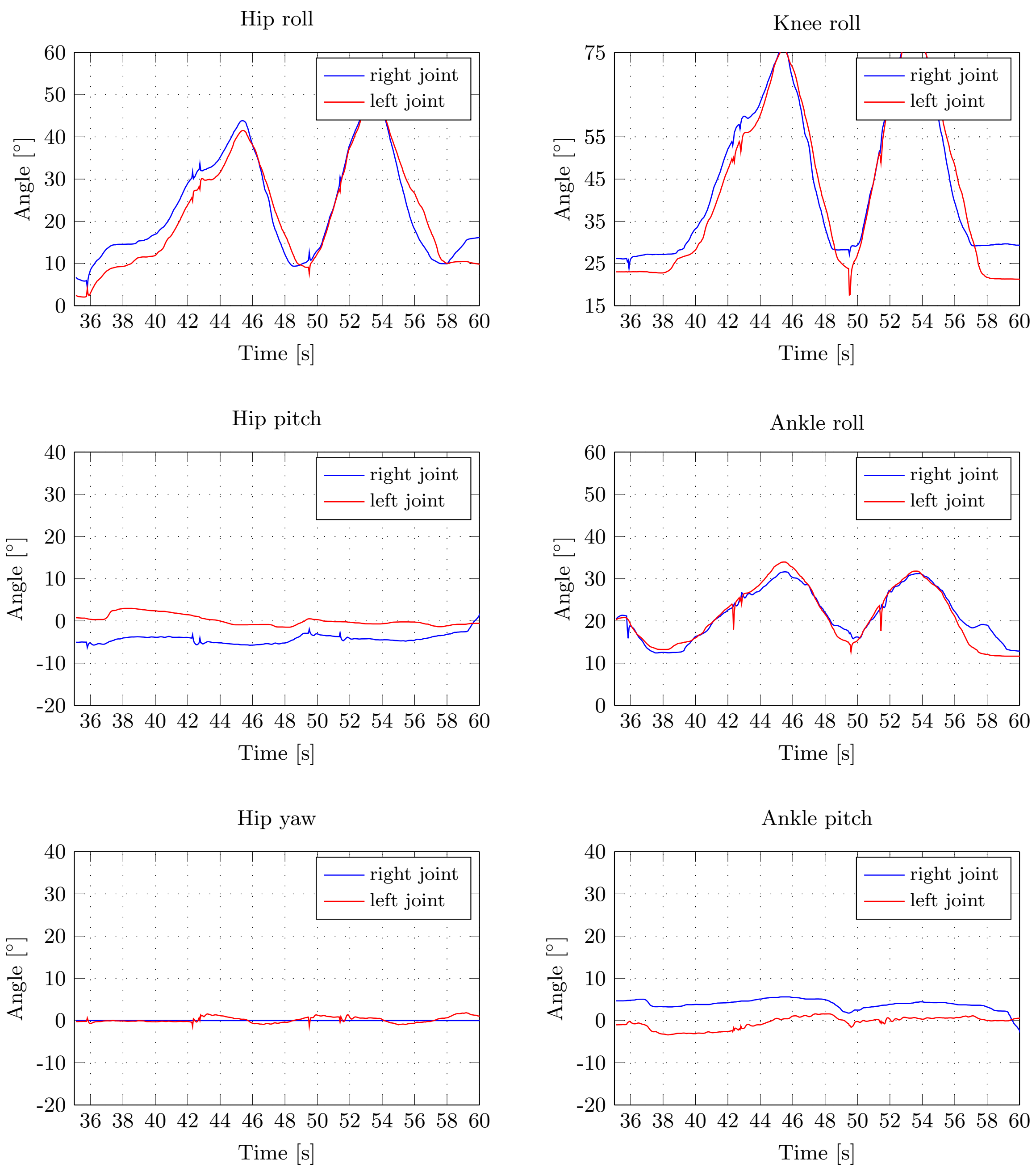
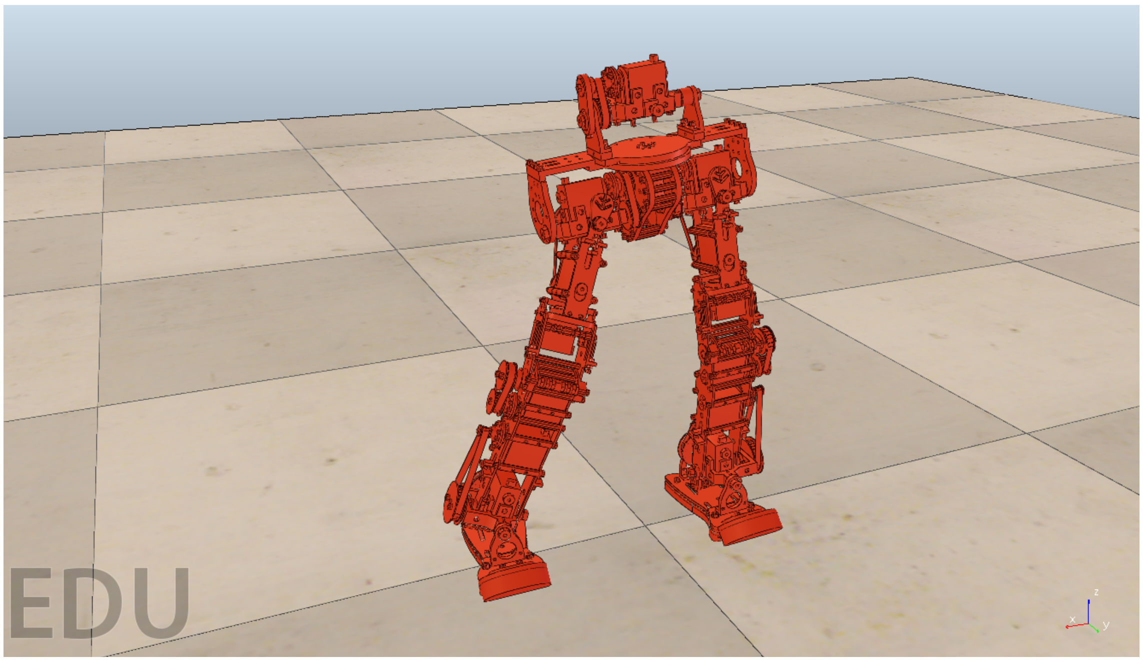
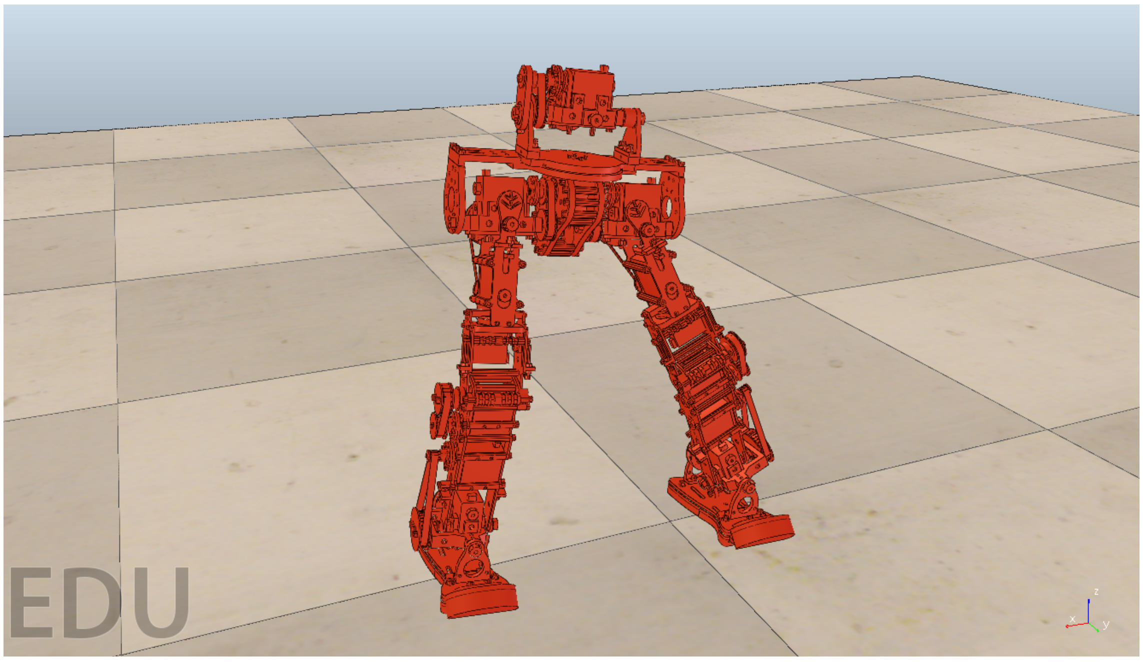


Figure 5.55: Robot's joint state evolution during the vertical motion of the body.

Single leg lateral movement



(a) Right leg sideway movement.



(b) Left leg sideway movement.

Figure 5.56: Single leg lateral movement.

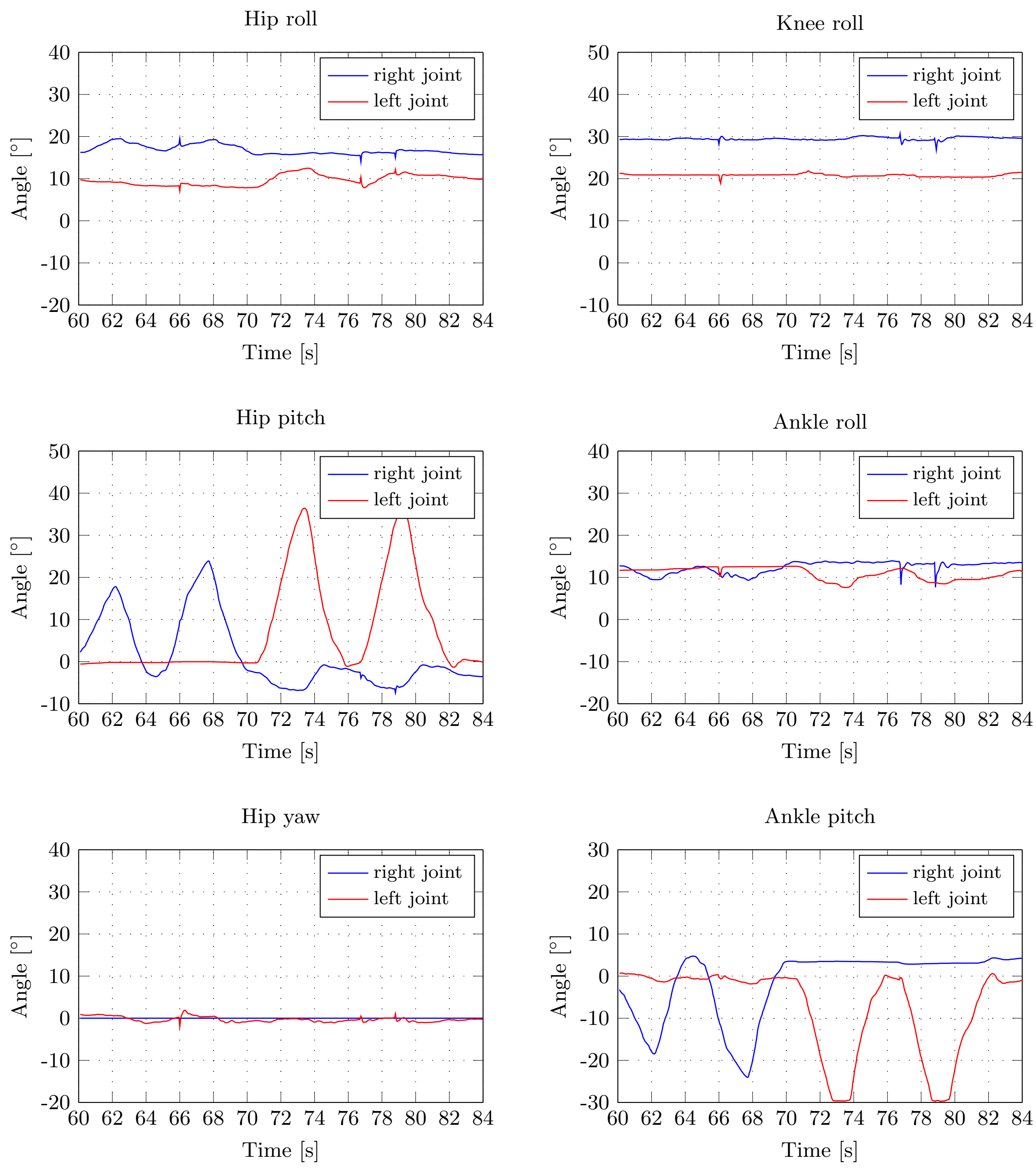
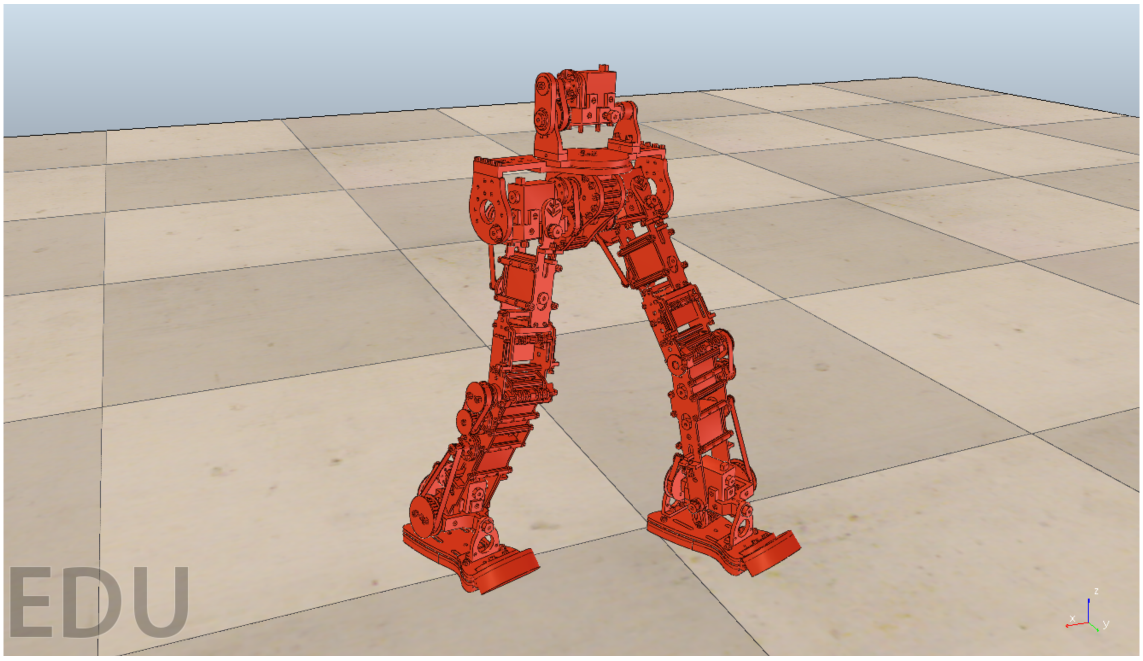
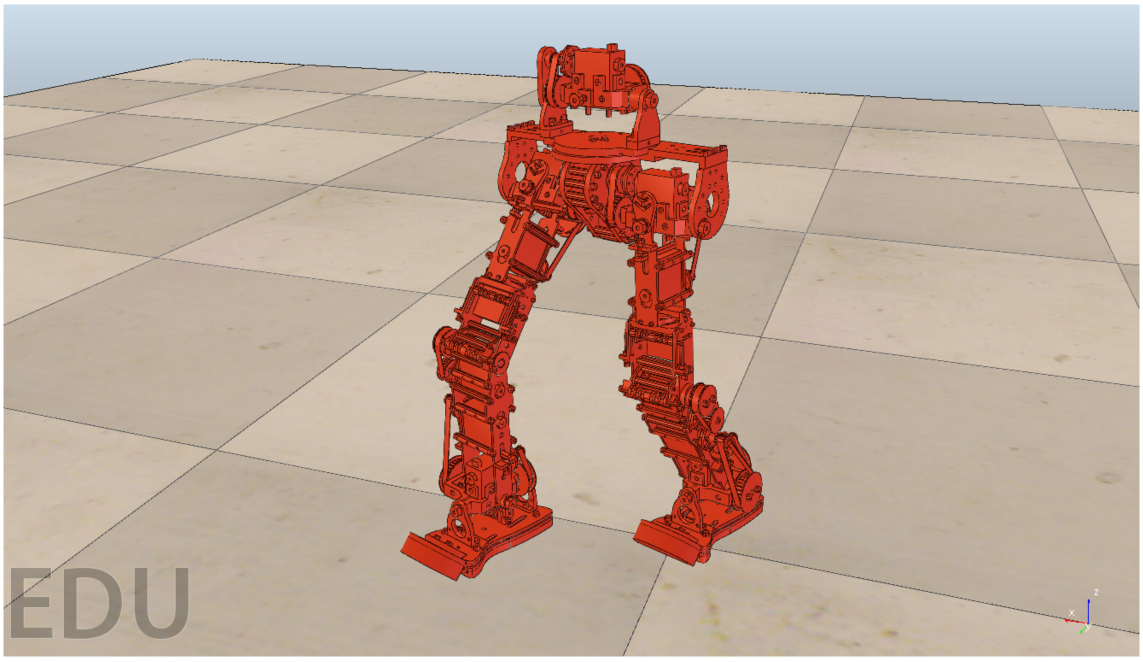


Figure 5.57: Robot's joint state evolution during the lateral motion of a single leg.

Gait typical pattern



(a) Left leg forward, right leg backward.



(b) Right leg forward, left leg backward.

Figure 5.58: Gait typical pattern.

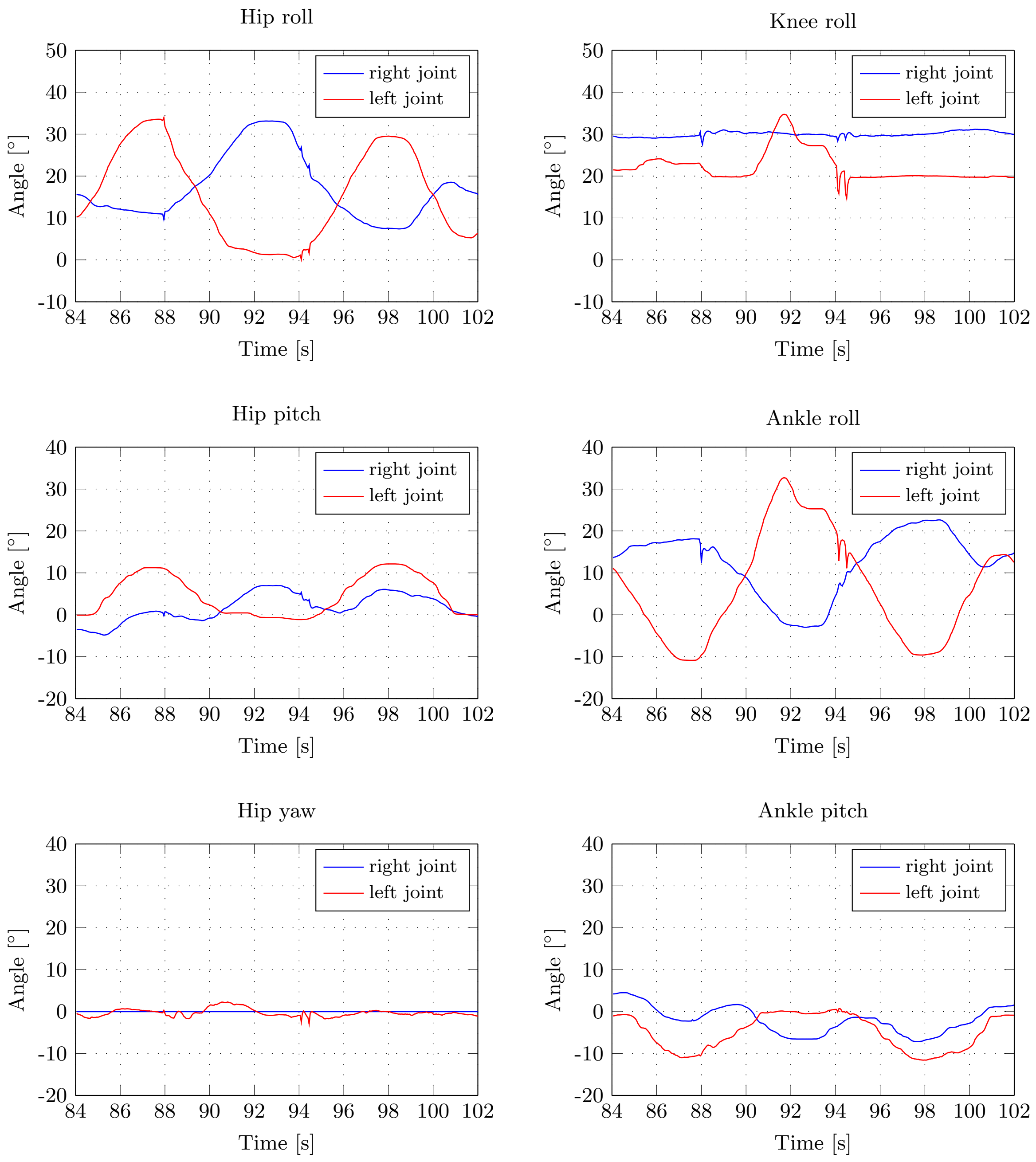
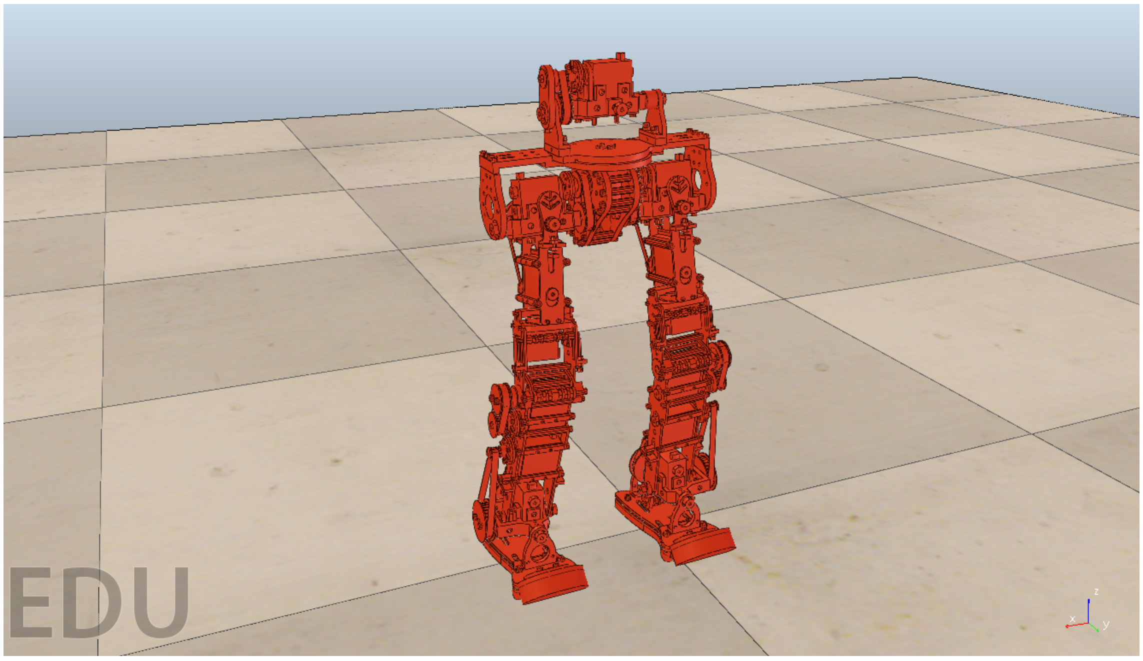
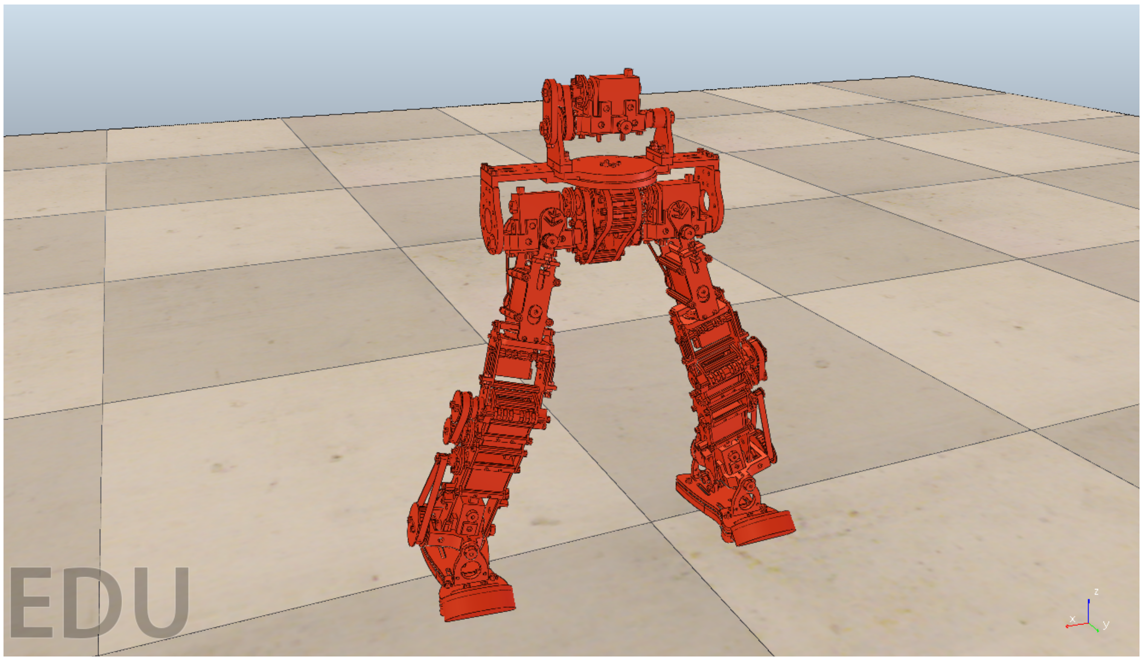


Figure 5.59: Robot's joint state evolution during a typical gait cycle.

Double leg lateral movement



(a) Stable configuration.



(b) Legs separated from each other.

Figure 5.60: Double leg lateral movement.

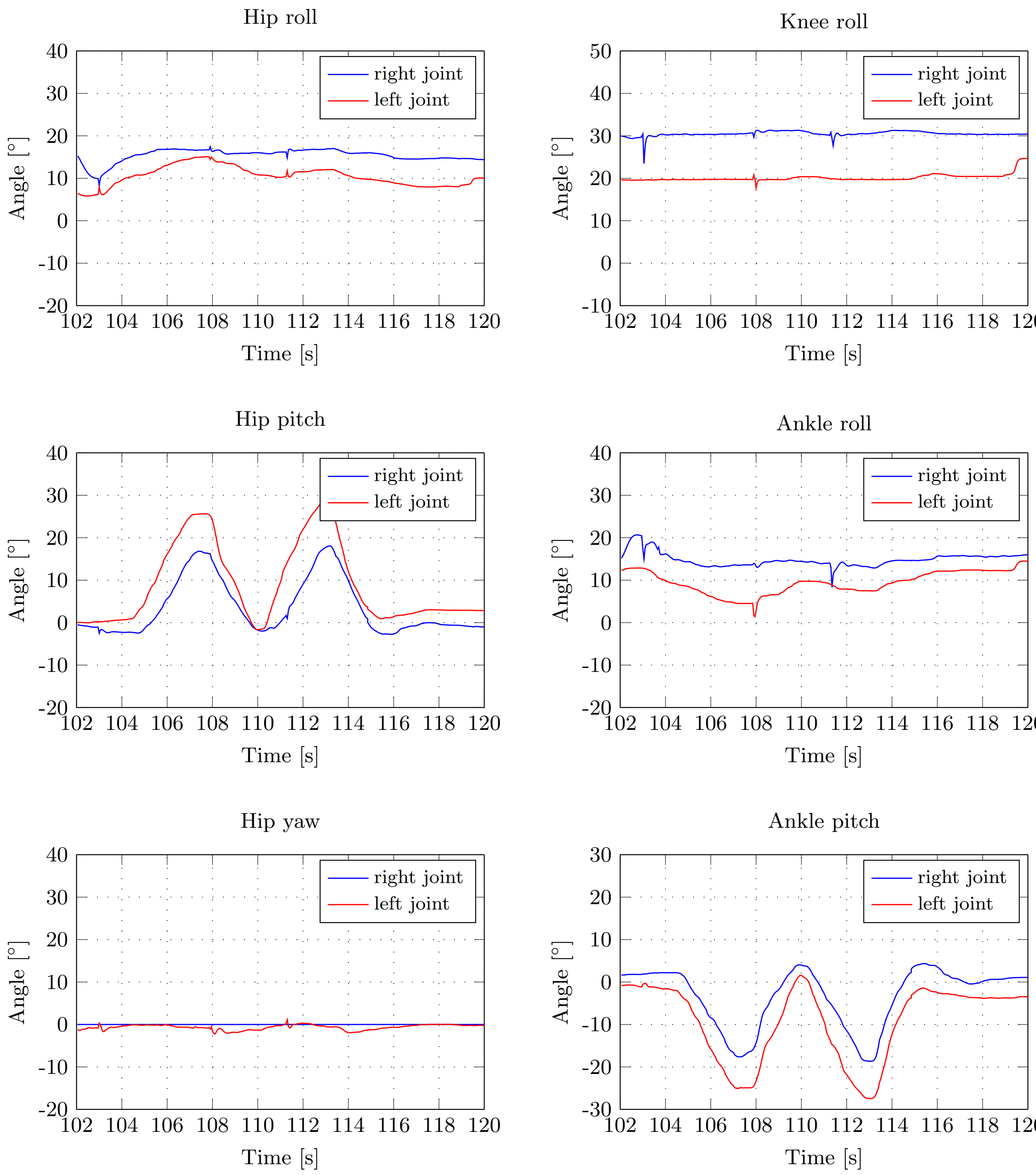


Figure 5.61: Robot's joint state evolution during the lateral motion of the two legs.

Chapter 6

Conclusions and Future Work

The final conclusions of this dissertation are detailed on this chapter, evaluating the success of the initially proposed objectives. For future works, some proposals are made, suggesting new approaches to the model's simulation, haptic rendering, and bimanual teleoperation in general.

6.1 Conclusions

The main objective of this dissertation work was to develop and test an experimental setup that would allow for cooperative haptic teleoperation of a simulated robot model, in V-REP.

Difficulties associated with the teleoperation of the PHUA platform, such as defective transmission of power in the belt and pulley mechanism, and the thorough calibration process of the force sensors, would make the teleoperation's success more difficult to achieve. Besides, the user's experience in this interaction is of significant importance. Simulating the teleoperation scenario in a virtual environment provides many benefits to the operator. He does not have to worry about restrictions imposed by the mechanical and electronic components, or time-consuming procedures to change and test new control parameters, or take measurements. Thereby, the user can focus almost exclusively on the teleoperation task.

Although the early proposal included the adaption of an existing V-REP model with similar characteristics, the possibility of creating a humanoid model and define its kinematic chains according to PHUA specifications, suggested a sounder way for the model's simulation. The definition of dynamically enabled *shapes* with correct inertial data, combined with *joint* and *force sensor* elements, made it possible to have dynamically-rich simulations with very satisfying results.

Even though the dynamic behavior of the default mechanism could be almost perfectly described, the implementation of the passive elastic elements existing in the hybrid actuation platform was not considered. These elements provide a passive mechanism to store and recover energy, mainly to enhance the system response and try to lessen the actuator's torque demands. Yet, since this is not a problem in simulation experiments, their use can be neglected.

Due to the impossibility of having two PHANToM Omni haptic devices running on the same machine, an approach based on a distributed ROS network was presented. Its application, fully described in Chapter 3, integrates software and hardware communication within several modules that interact with each other, and with the simulator. The modular capabilities of ROS are of great use for this project, having allowed for the successful implementation of the communication between the two haptic devices and the V-REP. The ROS functionality included in V-REP, which enables its operation as a ROS node, combined with the wide variety of *child script* commands associated with it, conceded the user a deep control over the simulation. Thanks to this modular architecture and publish/subscribe philosophy, the connection with the remaining applications, responsible for simulation's state retrieving and feedback force calculations, was easily resolved.

The low-level interaction with the PHANToM joystick was considered to be very complete, and not problematic on calculations and pre-processing routines. In fact, the force rendering frequency is not limited by hardware or dedicated software restrictions, but by the V-REP publishing frequency. Being integrated in one of the ROS modules implies that the joystick data gathering routines could not perform at frequencies above 20 Hz, the publishing frequency imposed by the simulator. As mentioned in Chapter 5, this was the value that best suited the user requirements, without compromising the simulation's success. The *servo loop* can update the device state at high frequencies ($\sim$ 1000 Hz), and the *main loop* sets no limits on the subscribing frequency. However, since the data publishing rate on the *feedback* topic is limited to 20 Hz, when the application's *main loop* subscribes to it and the callback is executed, if a new message has not yet been published, the force vector is not updated, and the joystick renders the old force values on the user.

The dependence of the ROS modules on each other, led to slight delays in information processing, which could be especially noted due to the low frequencies involved. An example was already shown in the experimental results description, when referring to *phantom_control* commands and *robot_state* perceived status. With the feedback generation, something similar happens. As the *haptic_feedback* module depends on the *robot_state* publishing data to compute the force vector sent to the joystick, the information available to the force calculation is about the last iteration, and not the simulation current state. Although the effects of these delays are not significant, an observation should be made.

Occasional glitches during the simulations were the main problem registered in this work. From time to time, the contact between the feet and the ground was not correctly defined, and the model *jumped* from one point to another. Although the contact might not be necessarily broken, this behavior led to appreciable distortions in the force sensor values. The force peaks caused by these disruptions had a huge influence on the CoP location, and consequently on the force feedback generation. In these conditions the simulations were seriously damaged.

The described behavior was caused by the incorrect, however necessary, construction of the feet plates. The physics engine might only register three contact points, which is enough for this type of contact. With three contact points, the robot can already stay in a stable position. However, each foot of the current model defines four contact points with the ground, in order to use the force sensor measurements, as described in Chapter 2. Thus, imagining the robot perfectly rigid, theoretically there are eight contact points, where only three are needed (considering the foot tips are not touching the ground, as illustrated in figures of the Chapter 5). Practically, there might be contact points that *jump* from one point of the feet contact to the other, causing an injurious effect on the teleoperation.

One way of making things easier was to have more compliance for the shapes defining the contact with the ground. The material properties for the ODE engine were modified in attempt to improve the simulation quality. In the experiments simulating the inverse kinematics motion, these modifications were applied, and a good improvement in the simulation consistency was verified. For the torque experiments, since the feet were not expected to be held together, the ODE properties that contemplate higher friction coefficients, for example, could not be applied, and the default Bullet engine was used instead. As a consequence, the unsatisfactory *jump* behavior was very common, and the force rendering on the user was disabled during that simulation. Anyhow, the resulting force was calculated, and the edited results are presented in Appendix A, for reference.

Using the developed force generation algorithms was successful for testing purposes, but this formulation still needs improvement for a better sense of feel by the operator. The force generation expressions used in this work are based on the CoP distance to the robot's support base. This formulation has proved to be useful and helpful for inverse kinematics simulations, offering a real perception of the system's dynamics. In contrast, it was not very appropriate for joint-by-joint simulations. In this kind of teleoperation, the feet were continuously moving, featuring rapid state changes that could not be covered by the current approach.

Despite the issues relating to force feedback generation, the position control method for both inverse kinematics and torque modes was considered successful. The robot's response to the user's teleoperation has proved to be reliable and very effective, under normal operation conditions. When controlled in the dual joystick configuration, the V-REP model offers a wide range of teleoperation possibilities, mainly providing an easy and clean way to test different scenarios. An experiment was proposed for the University of Aveiro Summer Academy activities, since this event represented a unique opportunity to test the system with inexperienced and unfamiliarized operators. Besides the experiments described in this work, which basically included the basic movements in the IK mode, and a bit more complex tasks with the two joysticks, it was proposed to the students that they teleoperate the robot to kick a virtual football. Surprisingly, some of the students revealed an extraordinary ability for teleoperation, having accomplished the kicking task while keeping the robot balanced. On the other hand, some of the operators could not perform even the simplest of tasks. Although without force feedback, this shows the very subjective nature of teleoperation.

The haptic demonstrations presented fitted the purpose of showing the potential of the created model, the V-REP itself, and the dual teleoperation, but do not represent a final version. They are meant to describe and substantiate the model capabilities and its relevance for future applications. The *gait pattern* experiment described in Chapter 5 aimed to gather further knowledge for the preparation of the walking gait. Although more sophisticated joint control is needed, it proved the reliability of the simulation, and most important, of the concept. Despite being tested in the simulator, the results of applying the described methodologies reveal an underlying consistency with the reality. This shows, in a first stage, that it is possible and relevant the development of an infrastructure to operate the real robot, based on the approach here presented.

This work culminated with the publication of an article in the *IEEE-RAS Humanoids2014 Conference*, entitled: "Tele-Kinesthetic Teaching of Motion Skills to Humanoid Robots through Haptic Feedback". In this article, the control solution based on the dual joystick configuration is presented, referring the methodologies and the experimental setup. Simulation tests are described alongside with experiments in the real robot, providing a solid and complementary base for the progress of the project.

The simulation has proved to be a useful tool, providing excellent means to develop the first tests, and real robots are indeed the ultimate tool to obtain the real data for future learning from demonstration, which is now expected to be easier in a soon foreseeable future [13].

6.2 Future Work

The software developed with this work is now considered to be the basis for further progress in simulation. With the ROS implementation, the project can evolve to different approaches, adding and combining new software modules with the existing ones to fit the teleoperation needs. An introduction of more dedicated software, in particular different haptic rendering formulations, will still maintain the validity of the control methodologies applied.

Special attention should be given to the foot plates construction, since the current typology places some restrictions to the simulations, particularly when it comes to CoP calculations. New alternatives for force sensors positioning must be thought, without compromising the contact definition by the physics engine. A single *pure shape*, especially *spheres*, should be always preferred to characterize the contact points with the ground. The model would not be a completely accurate reproduction of the PHUA robot, but more reliable CoP estimations would be secured. Besides, since the feedback generation is directly related to the CoP location, the rendering process would be much more precise, allowing for a deeper and truer user immersion in haptic control.

Haptics and haptic interfaces are a distinctive feature of this project, and the future heading of the PHUA humanoid robot towards the goal of autonomous balancing and locomotion. Force generation algorithms can be further developed in terms of mathematical formulation to increase the user's perception about the robot dynamics. As does the real platform, accelerometers, gyrosensors, vision and proximity sensors can also be adapted to the model, providing inertial and visual data that can be combined to enhance the feedback quality. Additionally, a third component of force can be introduced to describe the gravity effects, easily calculated by the total forces on each foot.

The force rendering scheme adopted in this work was a first simulated attempt to provide a "real feel" to the user, during the demonstrations. However, a simple formulation like this, which uses the lower level of interaction with OpenHaptics, could not provide a complete abstraction into the haptic scene. More complex formulations involving haptic objects algorithms that OpenHaptics possesses, should be attempted. These algorithms are capable of defining and refining dynamic haptic surfaces and stiffnesses that would lead the force feedback to a higher sensing level.

Defining metrics of the user's performance during the demonstrations is another important topic. Although they were not applied in this work, declaring specific indicators of the demonstration's quality is crucial for user and experiment evaluations, in what concerns to the learning process. Based on the evaluation of performance, the human teacher may provide functional guidance and corrections on the executed behavior. In a parallel work with the real platform, a metric is used to study the user's performance during the teleoperation [39]. In this experiment, the user tries to keep the robot in balance by compensating the force sensations that were transmitted by the haptic device. Then, the area of a polygon encompassing all the points of the CoP trajectory (convex-hull) is calculated. The quality of the teleoperation is directly related with this area; lower values of the convex-hull's area represent higher quality of the robot teleoperation.

In what gives respect to simulation, there is much more to be done. Exploring new scenarios, as uneven terrains, and include external disturbances in the robot teleoperation are within the next goals. Control methodologies for the upper part of the body may well be tempted. Although a direct control may be impossible, self-reactions of torso and arm joints to CoP's deviation could be used to help the robot's balance.

In further development, typical gait patterns can also be tested in a quite easy way. The approach used by other humanoid models provided by the V-REP, shows a very interesting strategy based on inverse kinematics control. As usual, two IK elements are defined, in order to control each foot individually. However, instead of having an external application controlling the feet position, there is a *child script* forcing the foot targets to go through a pre-defined walking path. When the dynamics module is enabled, due to the robot's weight, the model is able to follow the dynamic trajectory, performing consecutive walking cycles. The implementation of such strategy on the PHUA model would provide excellent means to analyze the robot dynamics in the first locomotion tasks.

To truly implement the bimanual teleoperation, a support bracket for the haptic devices should be designed. Placing the joysticks vertically, as depicted in the PHANToM picture of Chapter 4, would greatly ease the user's immersion into the teleoperation scene. The joint correspondence between the joysticks and the robot legs would become clearer and pleasant to the operator, significantly improving the perception of himself, and consequently the quality of the teleoperation.

References

[1] P. Cruz, V. Santos, and F. Silva, "Tele-kinesthetic teaching of a humanoid robot with haptic data acquisition," in *IROS'2012 Workshop on Learning and Interaction in Haptic Robots*, 2012.

[2] V. Hayward, O. R. 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.

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

[4] M. A. Srinivasan and C. Basdogan, "Haptics in virtual environments: Taxonomy, research status, and challenges," *Computers and Graphics (Pergamon)*, vol. 21, no. 4, pp. 393–404, 1997.

[5] N. F. Vaibhav, V. S. Mohit, and A. A. Anilesh, "Applications of haptics technology in advance robotics," in *INTERACT-2010*, pp. 273–277, IEEE, Dec. 2010.

[6] A. Talvas, M. Marchal, and A. Lecuyer, "A survey on bimanual haptic interaction.," *IEEE transactions on haptics*, vol. 7, pp. 285–300, Jan. 2014.

[7] 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.

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

[9] F. J. Clark and K. W. Horch, "Kinesthesia," *Handbook of Perception and Human Performance*, vol. 1, pp. 13–62, 1981.

[10] A. M. Howard, "Transfer of skills between human operators through haptic training with robot coordination," in *2010 IEEE International Conference on Robotics and Automation*, pp. 229–235, IEEE, May 2010.

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

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

[13] J. Barros, F. Serra, V. Santos, and F. Silva, "Tele-kinesthetic teaching of motion skills to humanoid robots through haptic feedback," in *IEEE-RAS Humanoids2014 Workshop on Policy Representations for Humanoid Robots*, 2014.

[14] P. Cruz, "Haptic interface data acquisition system," Master's thesis, University of Aveiro, 2012.

[15] E. Estrelinha, "Tele-operation of a humanoid robot using haptics and load sensors," Master's thesis, University of Aveiro, 2013.

[16] M. Whittle, *Gait Analysis: An Introduction.* Butterworth-Heinemann, 2007.

[17] CoppeliaRobotics, *V-REP User Manual.* [Online]. Available: `http://www.coppeliarobotics.com/helpFiles/index.html`.

[18] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1321–1326, IEEE, Nov. 2013.

[19] R. Sabino, "Estrutura híbrida de locomoção para um robô humanóide," Master's thesis, University of Aveiro, 2009.

[20] R. Godinho, "Desenvolvimento do tronco e braços de uma plataforma humanóide híbrida," Master's thesis, University of Aveiro, 2011.

[21] S. P. M. M. S. S. M. Kumar, P, "Gravity compensation for phantom omni haptic interface," in *First Joint International Conference on Multibody System Dynamics, IMSD-2010*, 2010.

[22] SensAble, *PHANTOM Omni User's Guide.* USA: SensAble Technologies, 2008.

[23] A. J. Silva, O. A. D. Ramirez, V. P. Vega, and J. P. O. Oliver, "PHANToM OMNI Haptic Device: Kinematic and Manipulability," in *2009 Electronics, Robotics and Automotive Mechanics Conference (CERMA)*, pp. 193–198, IEEE, Sept. 2009.

[24] SensAble, *OpenHaptics Toolkit Programmer's Guide.* USA: SensAble Technologies, 2008.

[25] SensAble, *OpenHaptics Toolkit API Reference Manual.* USA: SensAble Technologies, 2008.

[26] A. Martinez and E. Fernández, *Learning ROS for Robotics Programming.* Packt Publishing, Sept. 2013.

[27] ROS.org, *Documentation - ROS Wiki.* [Online]. Available: `http://wiki.ros.org/`.

[28] *Geometry Algorithms.* [Online]. Available: `http://geomalgorithms.com/`.

[29] A. Goswami, "Postural stability of biped robots and the foot-rotation indicator (FRI) point," *International Journal of Robotics Research*, vol. 18, no. 6, pp. 523–533, 1999.

[30] H. Hauser, G. Neumann, A. J. Ijspeert, and W. Maass, "Biologically inspired kinematic synergies enable linear balance control of a humanoid robot.," *Biological cybernetics*, vol. 104, pp. 235–49, May 2011.

[31] M. Vukobratović and D. Juricić, "Contribution to the synthesis of biped gait.," *IEEE Transactions on Biomedical Engineering*, vol. 16, no. 1, pp. 1–6, 1969.

[32] M. VUKOBRATOVIĆ and B. BOROVAC, "ZERO-MOMENT POINT - THIRTY FIVE YEARS OF ITS LIFE," *International Journal of Humanoid Robotics*, vol. 01, pp. 157–173, Mar. 2004.

[33] M. B. Popovic, "Ground Reference Points in Legged Locomotion: Definitions, Biological Trajectories and Control Implications," *The International Journal of Robotics Research*, vol. 24, pp. 1013–1032, Dec. 2005.

[34] M. Popovic, A. Hofmann, and H. Herr, "Angular momentum regulation during human walking: biomechanics and control," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, pp. 2405–2411 Vol.3, IEEE, 2004.

[35] S. Kajita, T. Yamaura, and A. Kobayashi, "Dynamic walking control of a biped robot along a potential energy conserving orbit," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 4, pp. 431–438, 1992.

[36] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, "The 3D linear inverted pendulum mode: A simple modeling for a biped walking pattern generation," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 1, pp. 239–240, 2001.

[37] S.-H. Lee and A. Goswami, "Reaction Mass Pendulum (RMP): An explicit model for centroidal angular momentum of humanoid robots," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 4667–4672, IEEE, Apr. 2007.

[38] H. Hauser, G. Neumann, A. J. Ijspeert, and W. Maass, "Biologically inspired kinematic synergies provide a new paradigm for balance control of humanoid robots," in *2007 7th IEEE-RAS International Conference on Humanoid Robots*, pp. 73–80, IEEE, Nov. 2007.

[39] F. Serra, "Haptic interface for the tele-operation of a humanoid robot," Master's thesis, University of Aveiro, 2014.

Appendix A

Experiments Additional Data

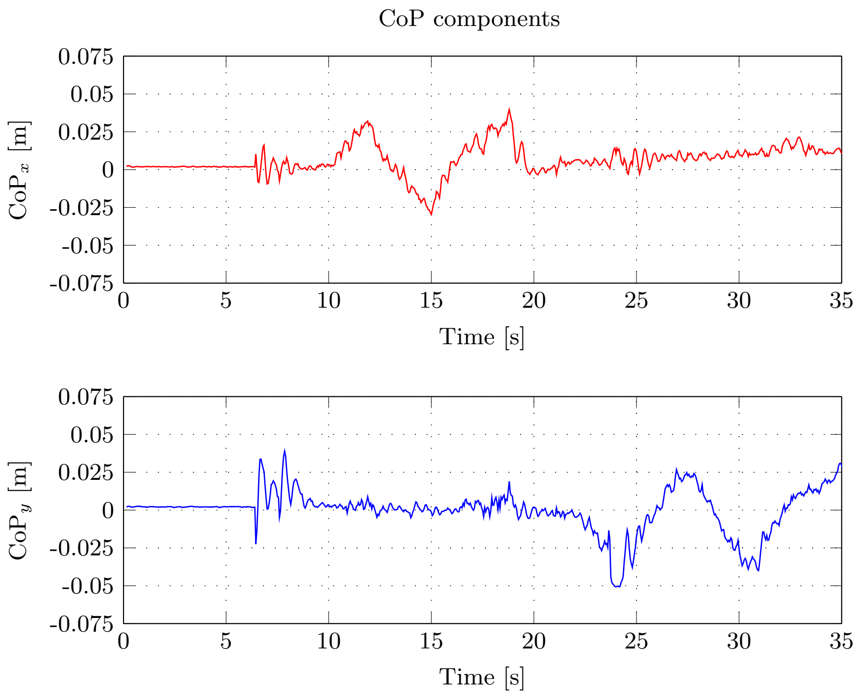


Figure A.1: Bi-dimentional CoP components variation during the lateral and sagittal motion of the body, in the torque mode simulation.

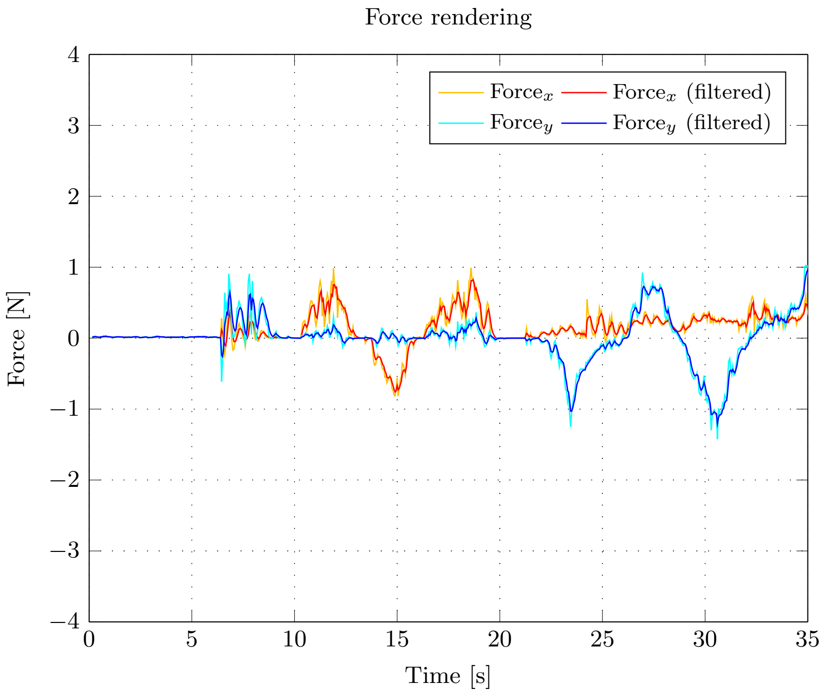
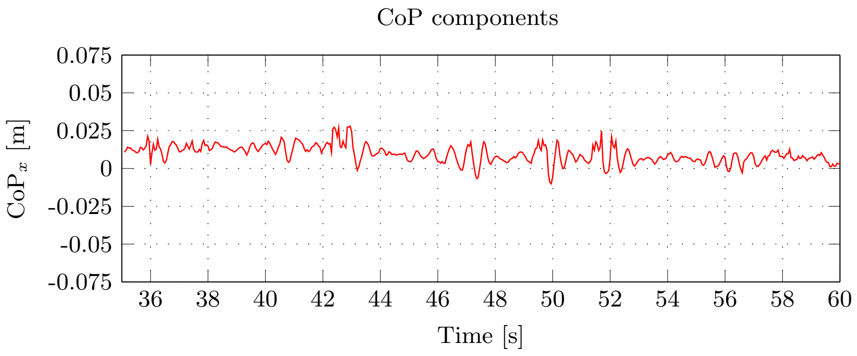


Figure A.2: Force rendered by the haptic device during the lateral and sagittal motion of the body, in the torque mode simulation.



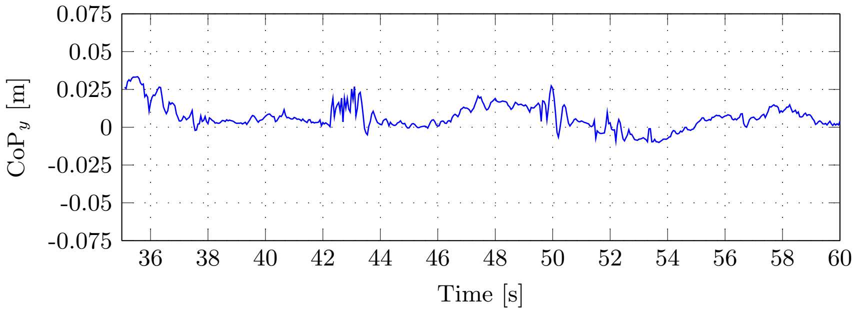


Figure A.3: Bi-dimentional CoP components variation during the vertical motion of the body, in the torque mode simulation.

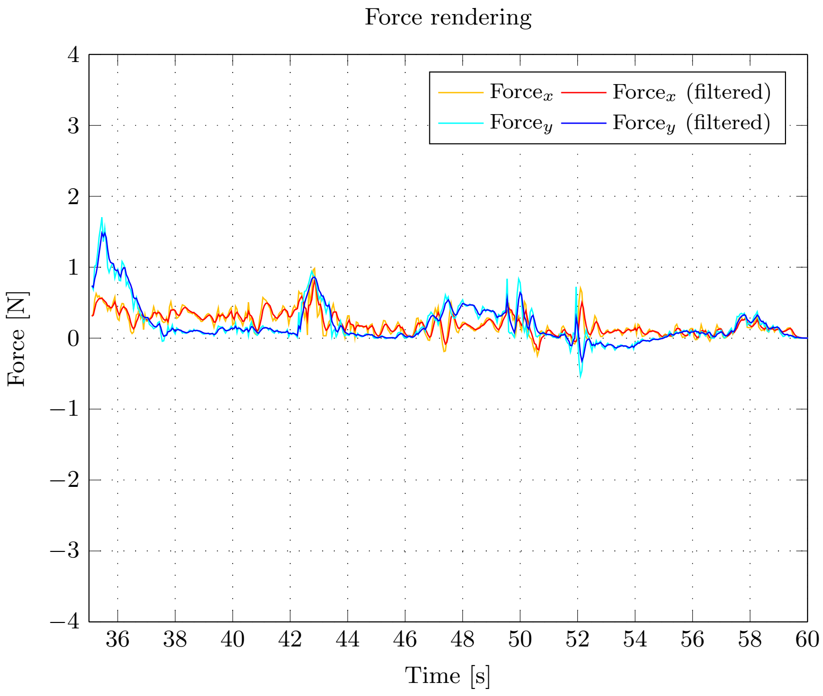


Figure A.4: Force rendered by the haptic device during the vertical motion of the body, in the torque mode simulation.

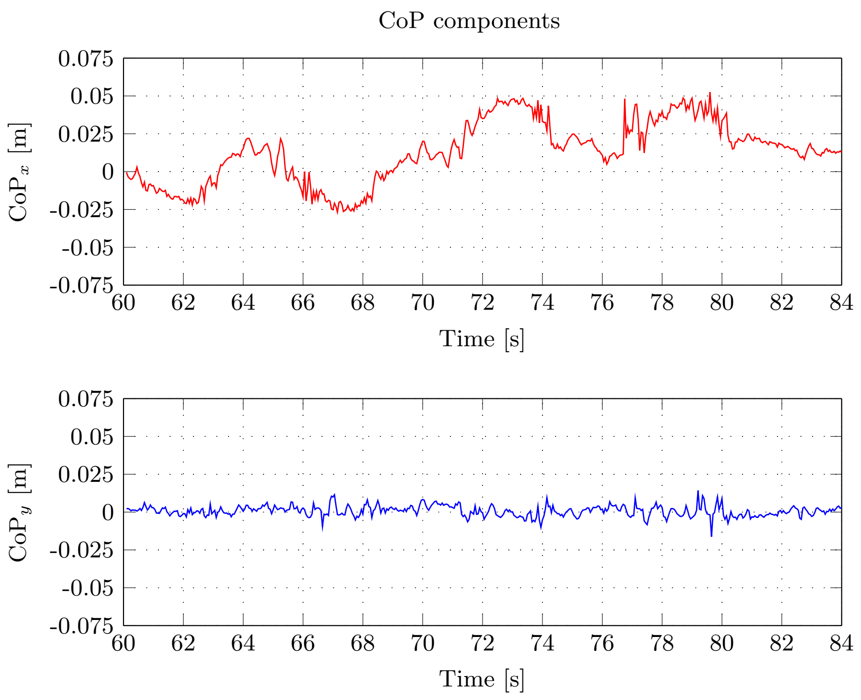


Figure A.5: Bi-dimentional CoP components variation during the lateral motion of a single leg, in the torque mode simulation.

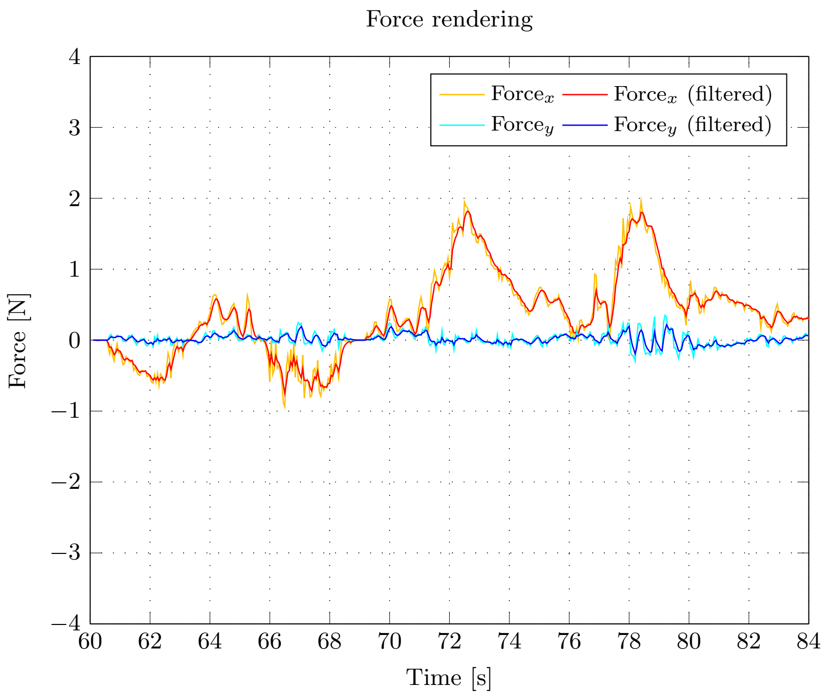


Figure A.6: Force rendered by the haptic device during the lateral motion of a single leg, in the torque mode simulation.

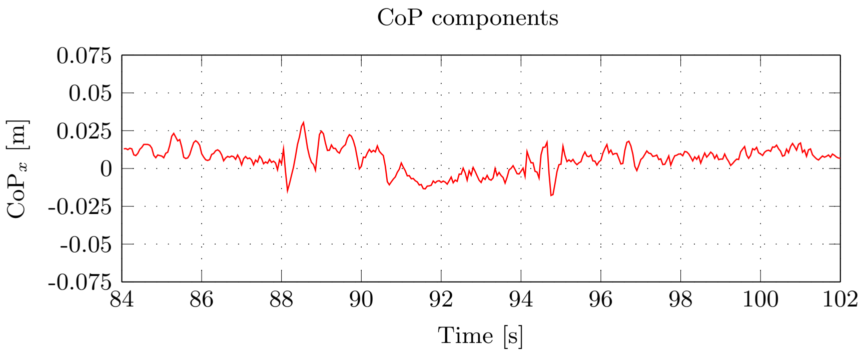


Figure A.7: Bi-dimentional CoP components variation during a typical gait cycle, in the torque mode simulation.

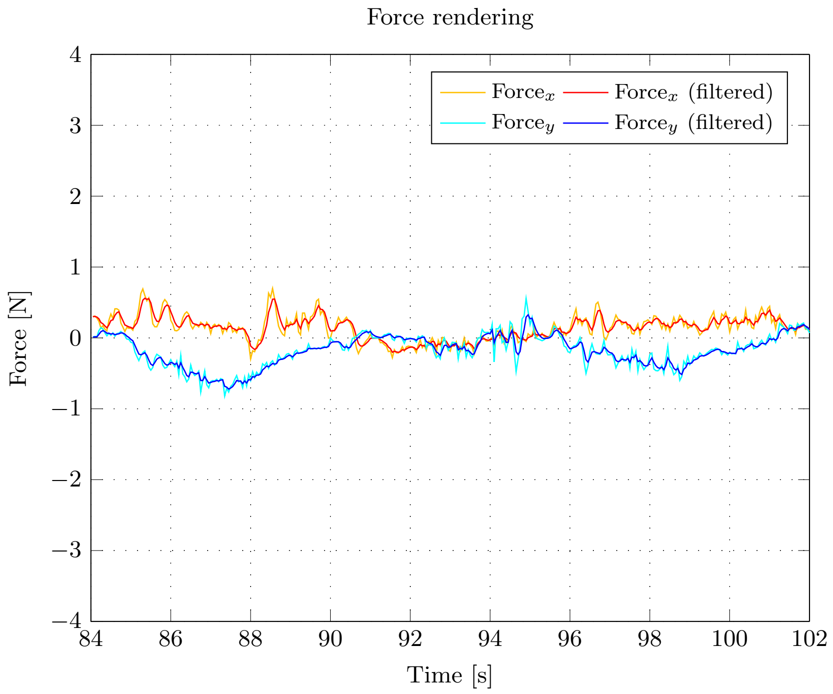


Figure A.8: Force rendered by the haptic device during a typical gait cycle, in the torque mode simulation.

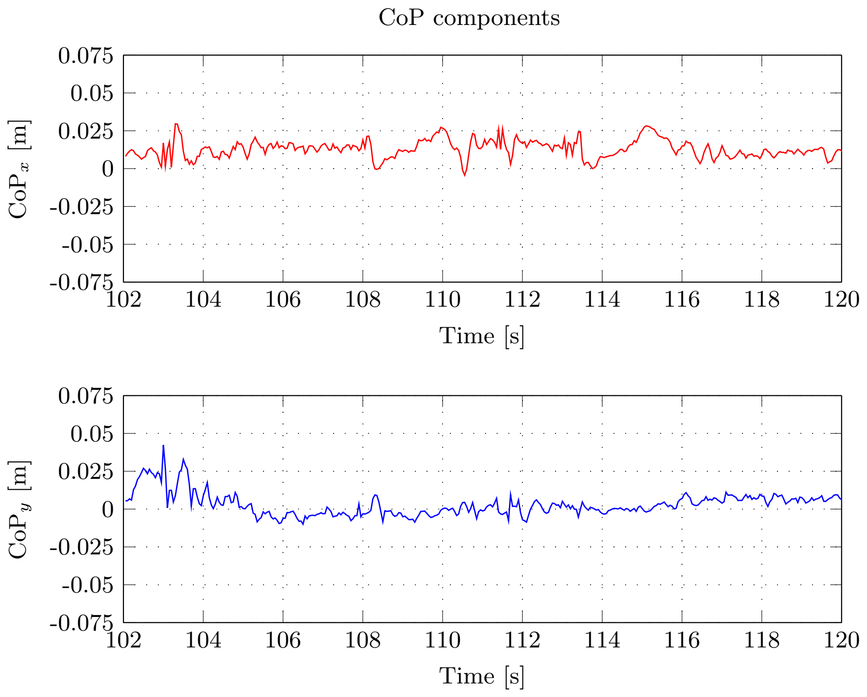


Figure A.9: Bi-dimentional CoP components variation during the lateral motion of the two legs, in the torque mode simulation.

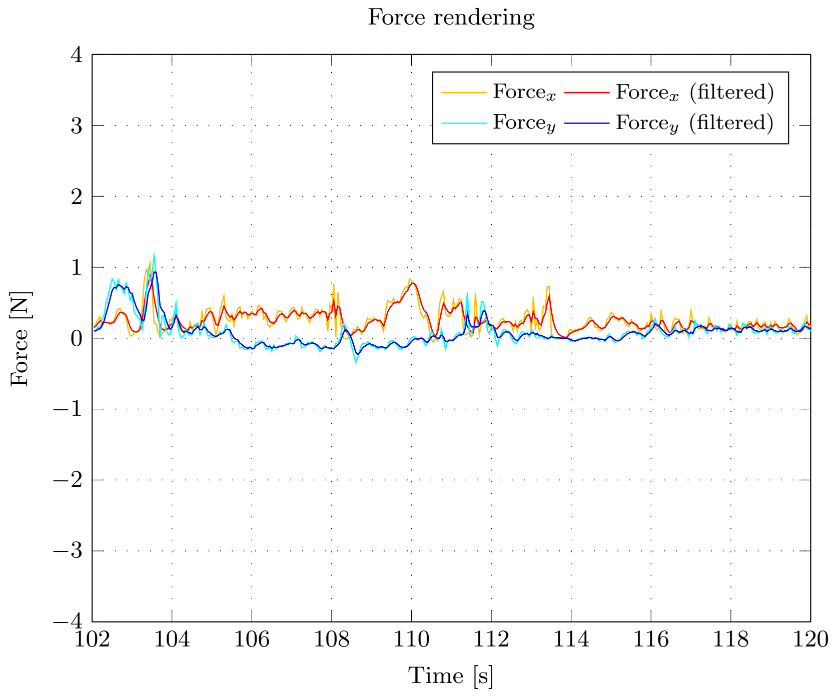


Figure A.10: Force rendered by the haptic device during the lateral motion of the two legs, in the torque mode simulation.

Appendix B

Software Notes

B.1 PHANToM Device Drivers and OpenHaptics

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

The default OpenHaptics Academic Edition drivers are not compatible with Ubuntu 12.04 LTS, the oldest version of the system supported by ROS *Hydro*. The installation procedure configures a beta version of the PDD (PHANToM Device Drivers) based on the new *FireWire* kernel driver stack (alias JUJU), that replaces the older *ieee1394* stack. [1] The code base of the new stack is smaller, cleaner, and closer follows modern Linux kernel programming paradigms, allowing for proper functioning of the device in recent systems.

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. Also, distribution-dependent repositories may have to be updated, and dependencies set, for system components.

Instructions for the migration to the new *FireWire* driver stack are detailed next, with special reference to the Ubuntu 12.04 64-bit filesystem. Dependencies and requisites mentioned on the file "HW_userguide_Linux.pdf" are installed, before attempting to install the PHANToM drivers.

B.1.1 Sensable Software Pre-Requisites

After signing up (ask for credentials), download the OpenHaptics Academic Edition for Linux v3.0 installer from DSC at `http://dsc.sensable.com/3dtouch/OpenHaptics_Academic_linux/software_downloads.asp`, and extract it to the */home/user* folder.

Download the JUJU PDD PRE-BETA package for Linux from `dsc.sensable.com/forum`.

Install freeglut:
sudo apt-get install freeglut3-dev

Download Mesa version 7.5.2 from `ftp://ftp.freedesktop.org/pub/mesa/older-versions/7.x/7.5.2/MesaLib-7.5.2.tar.bz2`.

[1]The *raw1394* (*FireWire*) module was replaced in Linux kernel versions 2.6.36 or newer.

For DRI-based hardware acceleration with Mesa, there are some requirements. Download dri2proto version 1.99.3 from `http://xorg.freedesktop.org/releases/individual/proto/dri2proto-1.99.3.tar.bz2` and extract it to the local folder. Then configure, compile, and install it with the commands (*sudo* is optional if the working sub-directory is within the */home* directory):

```
./configure
make
sudo make install
```

Build libDRM and Mesa by doing:

```
sudo apt-get build-dep libdrm
sudo apt-get build-dep mesa
```

If necessary, install libwayland0 package to build mesa:

```
sudo apt-get install libwayland0
```

Install libmotif-dev package:

```
sudo apt-get install libmotif-dev
```

Go to the extracted Mesa-7.5.2 folder and run:

```
./configure --enable-motif
```

Compile the libraries and install:

```
make
sudo make install
```

Make sure the symbolic links for *libGLw.so* and *libGLw.so.1* point properly to the newly build *libGLw.so.1.0.0*. Go to */usr/local/lib* and type *ls -la* to check. If necessary symlink:

```
sudo ln -s libGLw.so.1.0.0 libGLw.so.1
sudo ln -s libGLw.so.1.0.0 libGLw.so
```

B.1.2 Sensable Software Installation

Go to */home/user/OpenHapticsAE_Linux_v3_0/PHANTOM\ Device\ Drivers/64-bit/* folder, and install the drivers *.deb* package:

```
sudo dpkg -i phantomdevicedrivers_4.3-3_amd64.deb
```

Copy the *libPHANToMIO.so.4.3* into */usr/lib64* and overwrite the existing one. Go to *Linux_JUJU_PDD_64-bit* folder and type:

```
sudo cp -i libPHANToMIO.so.4.3 /usr/lib64
```

If not created, create the following symbolic links in */usr/lib64* folder:

```
sudo ln -s libPHANToMIO.so.4.3 libPHANToMIO.so
sudo ln -s libPHANToMIO.so.4.3 libPHANToMIO.so.4
```

Copy the binary *PHANToMConfiguration* from *Linux_JUJU_PDD_64-bit* folder into */usr/sbin* and overwrite the old one as well:

```
sudo cp -i PHANToMConfiguration /usr/sbin
```

When applied to 64-bit systems, the installation procedure described above defines some libraries out of the */usr/lib* folder. This can lead to some errors when trying to run *PHANToMConfiguration* and *PHANToMTest*. To avoid it, ask the dynamic linker to check for other locations, */usr/local/lib* and */usr/lib64* too. With root privileges edit the file */etc/ld.so.conf* and add the following lines:

```
/usr/local/lib
/usr/lib64
```

To update the cache run:
sudo ldconfig

These two small applications are installed by default with the drivers. The *PHANToMConfiguration* is an application for managing the connected hardware, defining the default device. The *PHANToMTest* is a small application designed to test the device's capabilities, and calibrate it. They must be executed before using the device.

PHANToMConfiguration and *PHANToMTest* need *libraw1394.so.8* to run, so in */usr/lib/x86_64-linux-gnu* do:
sudo ln -s libraw1394.so.11.0.1 libraw1394.so.8

PHANToMConfiguration and *PHANToMTest* should work by now. However, during the procedure some problems can be experienced with respect to the *libPHANToMIO.so.4.3* and *PHANToMConfiguration* copying process. In case of "PDD Error: raw1394 module not loaded, modprobe raw1394 to correct" occurs, try to rename the existing files and copy the original again from *Linux_JUJU_PDD_64-bit* folder. Verify symbolic links.

Although the applications work, the device cannot be detected yet. The PHANToM drivers do not have access to the *FireWire* communication port object, so it falls to the user responsibility to change the access rules. To set the permissions do:
*sudo chmod 777 /dev/fw**

This command have to be run before any call for device applications, or in alternative, added to the *shell's* startup script, or to the system's startup routines.

To install OpenHaptics libraries and examples, go to */home/user/OpenHapticsAE_Linux_v3_0/Open Haptics-AE\ 3.0/64-bit/* folder and run the OpenHaptics package, by typing:
sudo dpkg -i openhaptics-ae_3.0-2_amd64.deb

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

To run the examples, it is necessary to setup the OpenHaptics environment variable *3DTOUCH_BASE.* However, the Ubuntu *bash* does not allow to initiate an environment variable started by a number. A workaround for this problem is to install *tcsh shell* or *csh shell*, and old *C-shell* still supported on Linux, and run the commands:
sudo csh
setenv 3DTOUCH_BASE /usr/share/3DTouch

Type *exit* command to return to *bash.* In an alternative way, it is possible to do:
PATH=$PATH:/usr/share/3DTouch/
export PATH.

Before testing the OpenHaptics examples, compile them. For HD and HL there is a *makefile* in both console and graphics examples. Run *make*, with root privileges if necessary, in each one of those folders to compile the examples.

Mesa and graphical interface incompatibilities.
In case of incompatibility issues between Mesa and Linux graphical interface, try the following procedure:

Install some dependencies:
```
sudo apt-get install build-essential cdbs fakeroot dh-make debhelper debconf
```

Install Open source Graphic Drivers (recommended). The below command will remove all traces of Ubuntu's default fglrx drivers (if installed):
```
sudo apt-get remove --purge fglrx fglrx_* fglrx-amdcccle* fglrx-dev*
```

Remove the existing *xorg.conf*:
```
sudo rm /etc/X11/xorg.conf
```

Reinstall the *xorg.conf* (in this case for 64-bit):
```
sudo apt-get install --reinstall xserver-xorg-core libgl1-mesa-glx:amd64 libgl1-mesa-dri:amd64
```

If the following error appears ”The following packages have unmet dependencies: libgl1-mesa-glx : Depends: libglapi-mesa”, do:
```
sudo apt-get install -f
sudo dpkg --purge --force-depends ”libgl1-mesa-glz*”
sudo apt-get install
```

Configure xorg:
```
sudo dpkg-reconfigure xserver-xorg
```

Reboot the system:
```
sudo reboot
```

B.2 Additional software

B.2.1 Virtual Robot Experimentation Platform (V-REP)

The V-REP simulator, can be downloaded on `http://www.coppeliarobotics.com/`. Official documentation, support foruns, and additional information can also be accessed from this site.

To launch V-REP, run from the command line:
```
./vrep.sh
```

The ROS functionality in V-REP is enabled via a plugin: *libv_repExtROS.so* or *libv_repExtROS.dylib*. The code to the plugin is open source and is located in the *programming/ros_stacks* folder, in the V-REP directory. The plugin can easily be adapted to the user's needs. The 3.1.3 version is already compiled for ROS *Hydro*, using the *catkin* build system of ROS. However, previous versions of the *vrep_common* and *vrep_plugin* packages had to be recompiled, and adapted to this work. These packages can be found on LAR's repositories.

The plugin is loaded when V-REP is launched, but the load operation will only be successful if *roscore* was running at that time. Make sure to inspect V-REP's console window or terminal for details on plugin load operations.

B.2.2 Robot Operating System (ROS)

In this work, the ROS version installed was *Hydro*. The installable Linux version is system dependent, but the detailed instructions can be found on `http://wiki.ros.org/`. Tutorials, code examples, and associated libraries can also be found here.