Ecole Centrale de Nantes | Warsaw University of Technology

MASTER ERASMUS MUNDUS

EMARO "European Masters In Advanced Robotics"

2012/2013

Thesis Final Report

Presented by

Louise Penna Poubel

On 28/08/2013

Title

WHOLE-BODY ONLINE HUMAN MOTION IMITATION BY A HUMANOID ROBOT USING TASK SPECIFICATION

JURY

President:	Wisama Khalil	Professor, IRCCyN, Ecole Centrale de Nantes
Evaluators:	Wisama Khalil	Professor, IRCCyN, Ecole Centrale de Nantes
	Sophie Sakka	Maître de conférences, IRCCyN, Université de Poitiers
	Christine Chevallereau	Director of research, IRCCyN, CNRS
	Damien Chablat	Director of research, IRCCyN, CNRS
	Arnaud Hamon	Research engineer, IRCCyN
	Teresa Zielińska	Professor, Warsaw University of Technology

Supervisor:

Sophie Sakka, Maître de conférences, Université de Poitiers
Laboratory: Institut de Recherche en Communications et Cybernétique de Nantes

Abstract

Humanoid robots are made to the image of human beings. Consequently, it is natural to use human motion as an input to generate humanoid motion. This process is commonly referred to as imitation and it poses several challenges due to the kinematic and dynamic differences between human beings and robots. Due to these difficulties, real-time human motion imitation while keeping balance and changing supports is still an unsolved problem in the literature.

In this research, imitation based on inverse kinematics with task specification was implemented. Four tasks were performed: end-effector tracking, keeping static balance, joint space tracking and avoiding joint limits. The imitation is carried out online using either a markerless or a marker-based motion capture system.

The captured human motion's task space is scaled to the robot's size segment by segment at each time step. From this scaled motion, the desired end-effector poses are taken. The joint space is captured from the human motion only for the joints which correspond to the robot's degrees of freedom. Distinction is made between single support and double support.

The task prioritization process takes into account equality tasks and optimization tasks. A clamping loop is added in order to keep joints within their limits: some angles are fixed to their limits and the inverse kinematics is solved with the joints left free. The constraints are chosen according to the type of support or transition between supports: during transition, the center of mass is placed onto the desired support polygon and the swing foot is kept flat when close to the floor.

The method was validated with the 23-degrees-of-freedom NAO robot. This approach allows for online imitation closely tracking a wide variety of slow human movements with the whole body, including locomotion, while keeping balance and maintaining the nuances of human motion. Several human performers were able to use the system quickly due to its intuitiveness.

Résumé de thèse

Robots humanoïdes sont faits à l'image des êtres humains. Par conséquent, il est naturel d'utiliser le mouvement humain comme entrée pour générer un mouvement humanoïde. Ce processus est communément appelé imitation et il pose plusieurs défis en raison des différences cinématiques et dynamiques entre humains et robots. En raison de ces difficultés, l'imitation des mouvements humains en temps réel, en gardant l'équilibre et l'évolution des supports, est encore un problème non résolu dans la littérature.

Dans cette recherche, l'imitation basée sur la cinématique inverse avec des spécifications des tâches a été mis en oeuvre. Quatre tâches ont été effectuées: le suivi des effecteurs, le maintien de l'équilibre statique, le suivi des articulations humaines et l'évitement de butées. L'imitation est réalisée en ligne en utilisant des systèmes de capture de mouvement sans marqueurs ou avec marqueurs.

L'espace opérationnel du mouvement humain mesuré est adapté à la taille du robot, segment par segment, à chaque pas de temps. De ce mouvement mise à l'échelle, les poses souhaitées pour les effecteurs sont prises. L'espace articulaire est capturé à partir du mouvement humain que pour les articulations qui correspondent aux degrés de liberté du robot . Distinction est faite entre le simple support et le double support. La définition des priorités des tâches tient compte des tâches de l'égalité et des tâches de l'optimisation.

Une boucle est ajouté afin de maintenir les articulations au sein de leurs limites: certains angles sont fixés à leurs limites et la cinématique inverse est résolue avec les articulations libres. Les contraintes sont choisies en fonction du type de support ou de transition entre supports: pendant la transition le centre de gravité est placé dans le polygone de support souhaitée et le pied pivotant est maintenu à plat quand près du sol.

La méthode a été validée avec un robot NAO de 23 degrés de liberté. La méthode permet d'imitation en ligne suivi d'une grande variété de mouvements humains lents avec l'ensemble du corps tout en gardant l'équilibre et les nuances du mouvement humain. Plusieurs interprètes humains étaient capables d'utiliser le système rapidement en raison de son caractère intuitif.

Acknowledgements

I would like to thank the EMARO Consortium for selecting me for this program and the European Commission for providing financial support during my master's studies. Studying in Poland and France for two years was a cultivating experience.

I owe my deepest gratitude to Prof. Sophie Sakka for her guidance and attention. She suggested a very interesting thesis topic and showed great confidence in me, while always making sure I was moving in the right direction throughout this research.

It's a pleasure to thank professors and researchers at IRCCyN, especially Prof. Wisama Khalil, Prof. Christine Chevallereau, Denis Creusot, and Mohammed Rharda, for their assistance and kindness. I'd like to extend my merci to all EMARO professors at Ecole Centrale de Nantes and my dziękuję to those at Warsaw University of Technology.

I am grateful to Denis Ćehajić for taking the time to explain his research to me and making his implementation available, hvala! I also owe a lot of gracias to Michelle García, the best lab partner one could ask for, and a lot of mulţumesc to Victor Rosenzveig, the most attentive sempai in the world. Grazie also to Marta Moltedo, Christian Vassallo, Matilde Marsano, Alejandro Crespo and Oscar Rueda for aiding my research in various ways. And I couldn't forget to thank all the students who volunteered to try out the developed program.

I am indebted to my parents, Marilda and Cláudio, and my brother Vinícius. Obrigada for the understanding, support and trust while I conduct my studies far away from Brazil for so many years.

Special arigatou, xièxiè, shukran, danyawad, salamat, terima kasih and spasibo are extended to all my friends scattered around the world for the moral support throughout this research. And last but not least, a muchísimas gracias to my girlfriend María José for always being present in my life.

Contents

List of Figures

List of Tables

Nomenclature

2D	Two dimensions
3D	Three dimensions
API	Application programming interface
CoM	Center of mass
CoP	Center of pressure
CSV	Comma separated value
DGM	Direct geometric model
DKM	Direct kinematic model
DOF	Degree of freedom
DS	Double support
EMG	Electromyography
GRF	Ground reaction forces
GUI	Graphical user interface
IK	Inverse kinematics
IR	Infra-red
ISB	International Society of Biomechanics
LC	Lateral point on the lateral tibial condyle
LM	Lateral malleolus
LS	Left support
M-DH	Modified Denavit-Hartenberg parameters
MC	Medial point on the medial tibial condyle
MM	Medial malleolus
NA	Not addressed
RGB	Red, green, blue
RMS	Root mean square
RS	Right support
SS	Single support
SVD	Singular value decomposition
TT	Tibial tuberosity
w.r.t.	With respect to
ZMP	Zero moment point

List of variables

$\mathbf{q}$	Joint vector
$\mathbf{X}$	Generalized task vector
$\Delta\mathbf{q}$	Difference in joint vector (discrete velocity)
$\Delta\mathbf{X}_j$	Difference in task vector for task j (discrete velocity)
$\mathbf{J}_j$	Jacobian matrix of task j
$\mathbf{J}^+$	Pseudo-inverse of a Jacobian matrix
$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$	Matrices resulting from SVD decomposition
$\mathbf{J}^t$	Transpose of a Jacobian matrix
$\mathbf{Z}_j$	Optimization vector of task j
κ_j	Weight of optimization task j
$f_j(q)$	Optimization function of task j
$\mathbf{P}_j$	Null space projector of task j
$\mathbf{I}$	Identity matrix
$\mathbf{J}_j^a$	Augmented Jacobian matrix of task j
$\mathbf{P}_j^a$	Augmented null-space projector of task j
N	Total number of equality tasks
M	Total number of optimization tasks
q_i^c, q_i^d	Current and desired values for joint i
q_i^{min}	Minimum limit of joint i
q_i^{max}	Maximum limit of joint i
q_i^{med}	Median value of joint i
n	Total number of joints
$\mathbf{X}_j^c, \mathbf{X}_j^d$	Current and desired task vectors for task j
x_p, y_p, z_p	Cartesian coordinates of point p
$\ddot{x}_p, \ddot{y}_p, \ddot{z}_p$	Cartesian accelerations of point p
$\mathbf{Z}_j[i]$	Element of joint i in task j's optimization vector
$X_\ell, Y_\ell, Z_\ell, U_\ell$	Axes of frame ℓ
a_ℓ, s_ℓ	Antecedent and subsequent of frame ℓ
X_a, X_s	Subscripts a and s refer to antecedent or subsequent axis
$\alpha_\ell, \theta_\ell, \gamma_\ell$	Rotation parameters of frame ℓ
d_ℓ, r_ℓ, b_ℓ	Translation parameters of frame ℓ
${}^a\mathbf{T}_\ell$	Transformation matrix to frame ℓ from its antecedent
C, S	Cosine, sine
$\mathbf{P}_p$	Cartesian position of point p
m_ℓ	Mass of link ℓ
l_ℓ	Length of link ℓ
Sp, Sp_{CoM}	Thresholds to resize task vectors
${}^{Sp}\mathbf{X}^d$	Resized desired task vector

Chapter 1

Introduction

1.1 Humanoid robots

The field of robotics encompasses a wide range of machines, from industrial manipulators to autonomous cars, biomimetic robots, *etc.* Although a vast topic, in popular culture the word *robot* commonly refers specifically to one of the many branches in robotics: humanoid robots. In fact, the word *robota*, which means "labour" in Czech language, was coined as a term for machines resembling human beings by Karel Čapek in his play *Rossum's Universal Robots (R.U.R.)*, which portrayed mechanized slaves [1]. Indeed, humanoid robots are a recurring subject in science fiction and thus one of the most fascinating and expected technological advancements of our time.

Besides the satisfaction of human curiosity and imagination, the integration of humanoid robots in our daily lives makes sense for a variety of practical reasons. Robots resembling us would make human-robot interaction more natural and thus more intuitive and pleasurable. Moreover, if robots are to assist people in daily chores, they have to fit the human environment, which is suitable for human morphology [2]. Performing tasks with two hands, handling tools, climbing stairs, reaching shelves - to name only a few tasks - require our assistants to have a similar morphology to ours, so robots can adapt to human lives, instead of us having to do the opposite. Furthermore, research in humanoid robots directly contributes to the field of prosthetics and exoskeletons.

The challenges in creating such machines are numerous. Human bodies are energy efficient machines with extremely elaborated mechanics and incredible cognitive and perceptual abilities. Thus, the creation of humanoid robots requires advances in more areas than one, from the improvement of sensors and processing of information, to efficient control techniques and suitable mechanical structures with constraints in shape, size and weight to improve the resemblance to human beings.

Figure 1.1 illustrates part of the evolution of humanoid robots throughout history, from Leonardo da Vinci's first drawn mechanism in 1495 to *Aldebaran Robotics*' affordable *NAO* robot nowadays [3].

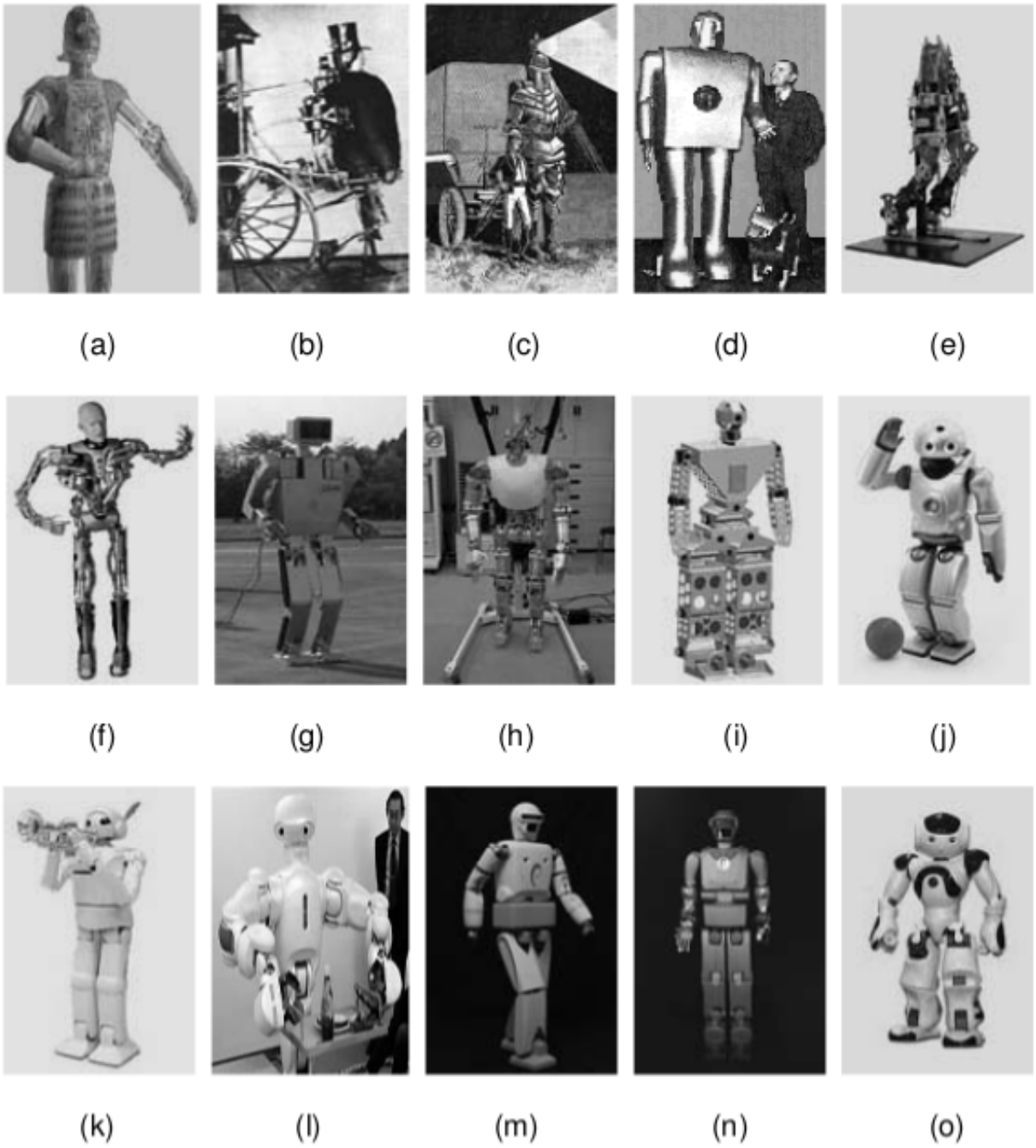


Figure 1.1: Some robots from ancient times to present time. (a) First humanoid by Leonardo da Vinci in 1495, (b) Steam Man in 1865, (c) Electric man in 1885, (d) ELEKTRO in 1938, (e) BIPER-4 in 1984, (f) Tron-Xm developed, Australia, 1997, (g) H6, Tokyo University, 2000, (h) Robot JACK in September 2000, (i) GuRoo in 2002, (j) QRIO from SONY in 2003, (k) Partnar Robot by Toyota Motor Company in 2004, (l) TwendyOne in 2007 from Waseda University, (m) REEM-A, chess player robot by UAE in 2007, (n) REEM-B by UAE and (o) *NAO*, in France 2008. Adapted from [3].

1.2 Human motion

Walking, running, jumping, dancing, ball-catching, manipulating delicate objects, climbing, swimming or simply standing up. The human body can perform an incredible wide range of motions using its 206 bones and 640 skeletal muscles guided by several senses: sight, hearing, taste, smell, touch, balance, proprioception and time. *Kinesiology* is the science which studies all the complexities of human motion. Human behaviour is characterized not only by the set of movements which our bodies can achieve, but also by the cognitive processes which lead us to the decisions of how to move [4]. The applications of the study of human motion include ergonomic products, prosthetics, animation and robotics.

Mechanically, the human body is so elaborate that simplified anthropometric models must be used in order to reduce it to a reasonable complexity. Commonly, the body's continuity is divided into a small number of discrete body parts which are approximated as geometrical solids. All the different tissues (bones, muscles, skin, *etc.*) are grouped into rigid solids with constant density for each body segment. For example, the *Hanavan Model* [5] is based on 25 anthropometric measures to estimate the dimensions of 15 body parts of simple geometry, as shown in Figure 1.2.

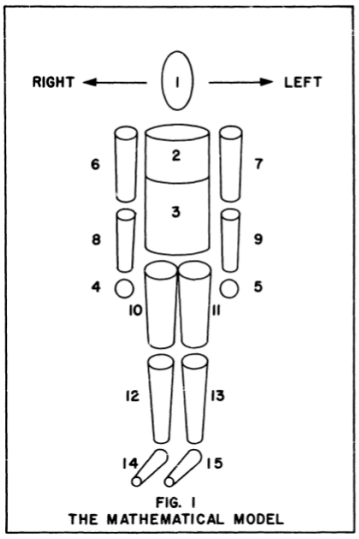


Figure 1.2: Human body approximation by rigid solids (*Hanavan model).* Adapted from [5].

In order to analyse the relative motions between these rigid bodies, they are commonly arranged in an open kinematic chain with a tree structure that has 5 end-effectors (one for each limb and the head) and its waist is the root which can be placed with any pose (6 *DOF*) in relation to the environment. These geometric and kinematic approximations are especially useful for robotics, since most existent humanoid robots are composed of a small number of rigid links connected in series.

Tracking human motion is another challenge. Kinematic motion capture can be performed with mountable sensors, such as suits, or remote sensors, like cameras. Note that some remote sensors might require the need for something to be mounted on the body,

but in this case the mounted objects are called markers, as they are tracked in *3D* by the the external sensors. Each motion capture system has its advantages and drawbacks according to the motion being investigated. A non-exhaustive review of current motion tracking technologies is proposed in Section 2.1.1.

Due to the body's non-uniform nature, measuring the body's dynamic parameters is also challenging. The inertial properties of body parts cannot be simply measured one by one and must be estimated based on statistical studies of the human population. Dynamic measurements during movement can be made with force sensors located in strategic places according to the observed motion. The dynamics of segments can only be estimated using kinematics and external force measurements.

1.3 Imitation

Given the resemblance between humanoid robots and human beings, it is natural to look for inspiration in human movements in order to generate motion for the humanoid. The most straightforward way is to have the robot observe what the human does and reproduce that behaviour, i.e. perform imitation. After all, even human beings themselves are able to acquire skills by imitation and learning [4, 6].

If humanoid robots are to interact with human beings, it is imperative that their gestures are human-like since much of human communication is non-verbal [7]. Programming each aspect of the motion detail by detail in order to make it human-like is time-consuming and not fit to handle the immense variety and complexity of human behaviours. Thus, imitation comes as a more natural and intuitive alternative to classical methods [4], since more information can be transmitted directly.

Imitation has been widely studied in psychology of animals and human beings. In the most advanced meaning of the word, imitation means to perform a behaviour after having witnessed it [4]. Thus, imitation as found in animals and human beings is more than the simple act of copying movements, since it includes the understanding of the task being performed and allows for flexibility in face of variant conditions (also referred to as *imitation learning* [8]).

However, several applications do not require the robot to fully understand the task being performed and its objectives. For example, if a robot is being teleoperated with the teleoperator's movements, the human being can adjust their movements to the task at hand according to the feedback received from the robot. In such situations, a simple reproduction of movements without providing the robot with understanding suffices (can be referred to as *motion imitation*). This could be seen not as teaching the robot a skill, but simply as using the robot as a performer for the needs of the human being. For this type of imitation, it is imperative that the motion retargeting occurs in real time. Teleoperation/telepresence is useful in hazardous situations where the human body could

be in danger, such as fires, radioactivity or even in space [9].

Motion imitation does not come without its challenges; even the most advanced humanoid robots have a different kinematic and dynamic structure to the human, such as number of degrees of freedom, joint, velocity and acceleration range, link lengths, mass distribution, *etc.* Thus, proper strategies must be made in order to transfer the captured human motion to the robot structure.

An implementation of human dance moves transferred to the *HRP2* robot can be seen in Figure 1.3. The robot was able to perform the moves while keeping balance, but the imitation was done with pre-processed data, not in real-time [10].

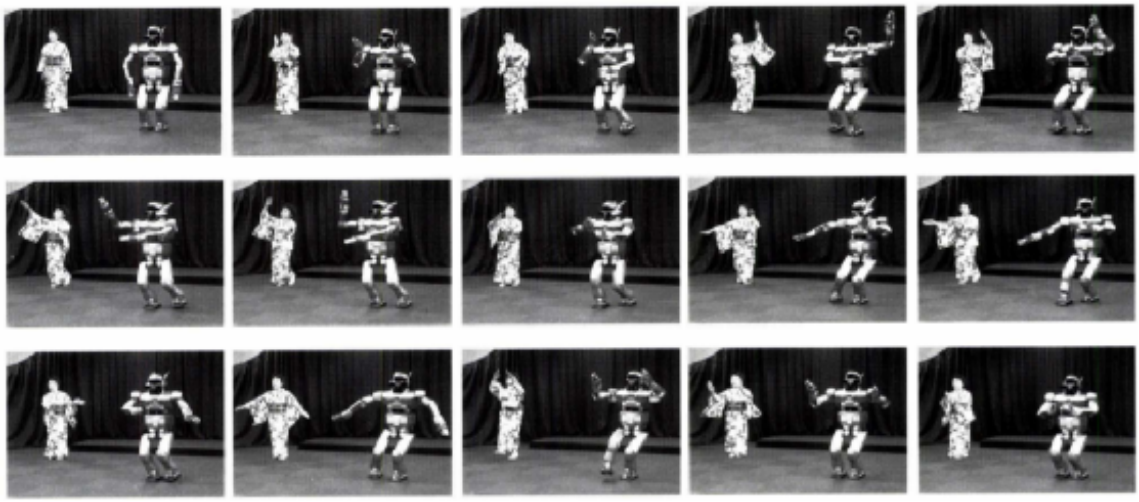


Figure 1.3: Dance performed by the *HRP2* robot through offline imitation. Adapted from [10].

1.4 Report overview

This chapter provided an overview of humanoid robots, human movement and imitation. In the following chapter, the state-of-the-art in capturing human motion and generating robot motions will be reviewed, while in Chapter 3, specific problems in imitation will be covered.

In Chapter 4, the theory used in this research, task priority-based inverse kinematics, will be covered in detail. In Chapter 5, the application of this theory to imitation in single support will be explained. Finally, imitation including support changes will be presented in Chapter 6. Chapter 7 will conclude the report with final remarks and perspectives for future work.

Chapter 2

Motion representation

Human motion - robot motion. This chapter deals with the specific problems of capturing the motion of a human being and of generating the motion for a robot. The motion capture section will mention the various systems found in the literature, while the motion generation section will explain the various methods of choosing trajectories for the robot to follow.

2.1 Human motion observation

This section will consider the literature on human motion capture and the subsequent kinematics and dynamics estimations.

2.1.1 Motion capture

Several motion capture systems have been developed in order to measure human motion. Each system comes with its advantages and drawbacks. We describe hereafter the most commonly used non-invasive systems for kinematic and dynamic measurements.

2.1.1.1 Embedded sensors

Some kinematic sensors can be directly attached to the body, such as electrogoniometers and accelerometers [11]. The former measures the angle between segments and the latter measures their acceleration. Therefore, the data obtained through these sensors must be derivated or integrated and filtered in order to be used. Figure 2.1 shows an example of a motion capture system where accelerometers were embedded on a shirt [12].

Figure 2.1: Motion capture system based on accelerometers. Adapted from [12].

2.1.1.2 Marker-based systems

Motion capture systems based on vision are very popular. These can be divided into systems which use markers, and markerless systems. Marker-based motion capture consists of tracking markers attached to the body by cameras in *3D* space as shown in Figure 2.2. Markers are made so they can be easily distinguished from the background. For example, reflective or light-emitting markers can be easily tracked by infra-red cameras. These markers are tracked by at least two cameras and their position in *3D* is estimated at each time step by the triangulation of the *2D* positions as is done with stereovision. The more cameras there are, the bigger is the volume of capture and the lower are the risks of occlusion (loss of marker).

Figure 2.2: Motion capture system based on markers. Adapted from [13].

Since the body is being approximated as rigid solids, it is of our interest to place markers in positions which move together with the big body masses which we are estimating. This means that we should choose our body landmarks as close as possible to bones, that is, at places where bones come close to the surface of skin. Nevertheless, no matter how well the landmarks are chosen, there will always be soft tissue between the bone and the marker, giving origin to movements between the two. These undesired movements are called *skin artefacts*, and they can be easily observed for example during impact, when the soft tissues vibrate in higher frequencies than the hard tissues. The method to deal with this problem is called *solidification* [14], which consists of measuring the relative motions between markers that should be fixed in relation to each other and choosing

the best fit position for them. Once the marker positions have been defined, anatomical frames of reference can be assigned to each moving body segment.

Marker-based systems are currently the ones which provide the most accurate results. However, the need to place markers on the subject makes the system inconvenient. Also, as many cameras as there might be, it is common for markers to be occluded during the motion and some preprocessing of the data must be made beforehand to fill these gaps [15].

Some experiments in this research are performed with a marker-based system. This specific system will be introduced in Section 5.1.2.2.

2.1.1.3 Markerless systems

Markerless systems come as a more convenient way of capturing human motion, as nothing must be placed on the subject. These systems work by tracking body descriptors in images [11]. Body descriptors are anatomical features such as the shape of a head, a hand, an elbow, *etc.*

Single camera based markerless systems acquire information in two dimensions and must therefore estimate the depth of each feature by its relative size. Systems with 2 cameras can use stereovision to rebuild the *3D* position of features from *2D* images. More recent systems combine camera information to depth information provided by laser depth sensors. An example of such a system is the *Microsoft Kinect* sensor [16]. Markerless methods in general have worse accuracy than marker-based ones due to the difficulties related to feature extraction in computer vision. Motion jittering and change in segment dimensions during the motion are common problems.

The specific markerless system used in this research will be introduced in Section 5.1.2.1.

2.1.1.4 Force capture

The systems described above focus on the kinematics of the movement. However, the dynamics are also important for robotic applications and increasingly for animation purposes in order to increase realism [17].

When not interacting with objects, human bodies are subject to only two forces: weight and ground reaction forces (*GRF*). Ground reaction forces can be measured by force plates equipped with piezoelectric materials or with strain gauges strategically arranged on the floor so they can give the 6 components of the ground reaction (3 forces and 3 moments) [11].

When human beings interact with objects the forces between the human body and the object are usually important as well. In this case, force sensors can be placed onto surfaces o interest. If the interaction is manipulation, force sensors can be placed on fingertips; for a sitting interaction the sensors are placed on a chair and so on.

2.1.1.5 Electromyographic measurements

Electromyographic (*EMG*) measurements (electrical activity of skeletal muscles) are also interesting for some studies as they give information about movement timing and force [11]. They can be measured with electrodes placed on the surface of the skin.

2.1.2 Motion analysis

After the human motion has been captured, the problem arises of how to analyse this motion. This section briefly explains how to extract kinematic and dynamic information from captured motion.

2.1.2.1 Kinematic chain

When using marker-based motion capture systems, segment frames and joint frames must be estimated from the captured marker positions. The *International Society of Biomechanics* (*ISB*) proposes a convention for the placement of markers on the body and frame definitions for the body segments and joints [18]. As an example, on Figure 2.3 the coordinate system definitions for the tibia/fibula (calf of the leg) and the calcaneus (ankle) are shown. The symbols *LC*, *MC*, *TT*, *LM* and *MM* correspond to the tracked markers placed over the skin near the bone. The frames are defined based on these markers according to the guidelines.

With all the body coordinate frames properly attached, the kinematic chain can be studied with the use of, for example, homogeneous matrices. The *ISB* does not have specific recommendations for frame attachment for motion captured with markerless systems, so the frames must be approximated as the researcher sees fit.

2.1.2.2 Body inertial parameters

In order to analyse the dynamics of motion, the *Body Inertial Parameters* of the human being under study must be estimated. This is a set of parameters used to represent the body through geometrical shapes. Unlike robots, whose parameters are well known, bodies are difficult to measure as the parts cannot be individually separated, measured and weighed.

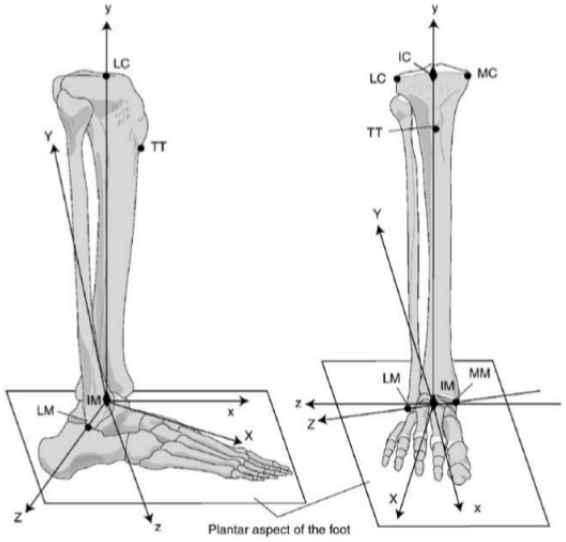


Figure 2.3: Conventions for human reference frames recommended by the *ISB*. Adapted from [18].

The first model to approximate the body with homogeneous rigid solids was proposed by Hanavan in 1964 [5] and was previously shown in Figure 1.2. Since then, several authors have proposed alternative divisions, for example, Yeadon [19] makes use of 95 anthropometric measurements to define 40 geometric solids. Figure 2.4 illustrates the division proposed by Kwon [20], known as the *Modified Hanavan Model*, formed by 16 solids approximated with 41 anthropometric measurements.

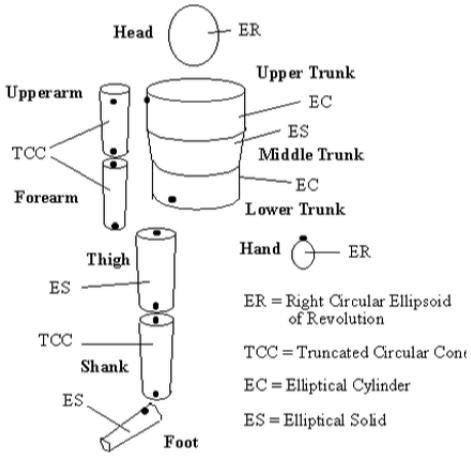


Figure 2.4: Modified Hanavan Model. Adapted from [20].

The difficulty in approximating the body with simple shapes is that during motion, human segments change in length, volume and mass distribution. Statistics made with thousands of human subjects gave origin to equations which approximate the density, inertia matrix and dimensions of the geometrical solids used in the *Modified Hanavan Model*.

Putting together the kinematic chain and the inertial parameters, kinematic (position, velocity, acceleration) and dynamic (momentum, forces, energies) analysis can be made.

2.2 Robot motion generation

Human beings are set apart from other animals by their impressive reasoning skills. Human beings are able to extract the important information from the task at hand in order to achieve the desired goals. But in similar ways to animals, our brains not only coordinate body movements, but also make elaborate decisions of how and when to move based on the feedback received from the senses.

There are various ways of generating motion to robots in order to handle their redundancy and rigidity while maintaining balance. This section reviews some of these strategies.

2.2.1 Direct kinematics

The geometric model of a robot is the function which calculates each end-effector's pose (task space) given the position of each joint angle in the chain (joint space) [21]. The kinematic model is the equivalent for the velocities, and in this case, the matrix which relates the joint space velocities and task space velocities is called the *Jacobian* matrix. In-depth mathematical aspects can be found on Chapter 4.

When using the direct kinematic model, the positions and velocities of each joint in the chain are known. Thus, generating motion for the robot based on the direct kinematics requires just the input of joint angles to the robot [8]. When the robot is programmed in a low level, joint trajectories are carefully designed so the robot performs as desired. This method can be time-consuming due to the large number of *DOF* in humanoids.

An alternative to describing each detail of the motion is to adapt captured human joint angles to the robot. This way, the redundancy of the robot is dealt with and the motion generated is human-like. Due to the kinematic differences between the human and the robot, however, this is not a straightforward process. Joint trajectories which ensure balance for the human might not do the same for the robot. Moreover, the limited number of *DOF* in robots mean that not all human joints can be imitated. These and other difficulties specific to imitation will be addressed in the next chapter.

2.2.2 Inverse kinematics

Inversely to the direct geometric and kinematic models, the inverse geometric and kinematic models are the ones which, given a desired end-effector pose or velocity, provide the necessary joint positions and velocities respectively [21].

With this method, the only desired feature is the pose of a few chosen effectors, while the state of all other frames is ignored. This can be used with pure programming or imitation: in the former, the end-effector pose to be tracked is designed by the programmer, while

in the latter the desired poses come from captured human motion.

In the field of computer animation, inverse kinematics (IK) has been widely used to generate the motion of animated characters from captured human motion [22]. As shown in Figure 2.5, depending on the geometric differences between the human and the character or robot, the resulting overall body configuration can be very different.

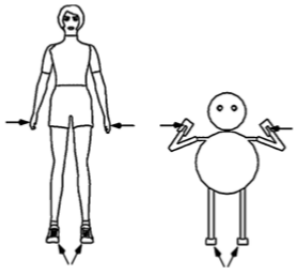


Figure 2.5: Inverse kinematics mapping from human to animated character. Adapted from [22].

It is worth noting however that there are tasks in which the overall configuration of the body is not as important as the position of the end-effector. For instance, when interacting with objects, it is necessary to track the absolute position of the end-effector in the Cartesian space so as to make sure that the object will be reached, as shown in Figure 2.6. Thus, whether to use direct or inverse kinematics should be chosen according to the task at hand [22].

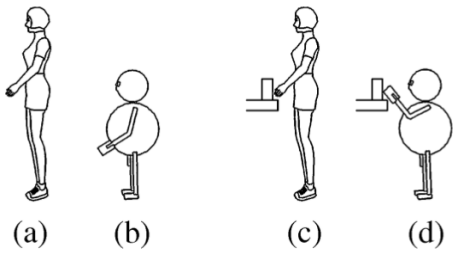


Figure 2.6: (a,b) Mapping by direct kinematics, (c,d) Mapping by inverse kinematics. Adapted from [22].

In robotics, this method has been widely implemented. Most authors do not use simple inverse kinematics though. The structure of a humanoid robot is redundant, which means that it has more DOF than the number needed to position its end-effectors during most applications. Thus, the extra DOF can be used to perform other tasks, such as balancing, avoiding singularities, dodging obstacles, minimizing energy consumption, *etc.*

Usually more than one aspect of the motion must be transmitted to the robot at once. To this goal, the concept of *task prioritization* comes in handy. When solving the inverse kinematics, the Jacobian of one task can be projected into the null space of another to allow secondary tasks to be performed [21]. Montecillo *et al.* [13], for example, divided their tasks into position, orientation and center of mass. This way, even if not all constraints can be solved, at least the most important aspects of the motion are followed.

Dariush *et al.* [23] also arranged their task descriptors according to priorities. In this research, inverse kinematics with task specification will be used, so a detailed description of the method is found in Chapter 4.

2.2.3 Motion primitives

Instead of being treated as a continuum, motion can be subdivided into a finite set of motion primitives. Motion primitives are a set of pre-designed joint trajectories commonly described in the joint space (a special case of direct kinematics). The advantage of this method is allowing high-level programming. On the downside, limiting the motion to only a few possibilities constraints the number of achievable poses. For example, instead of performing different types of kicks, whenever a kick motion is required, the robot always performs the same movement of the leg which is known to work.

Motion primitives were implemented by Nakaoka *et al.* [24] as a way to facilitate the imitation of leg motions. They divided all the possible leg motions during a dance into three primitives (stand, squat, step), as shown in Figure 2.7.

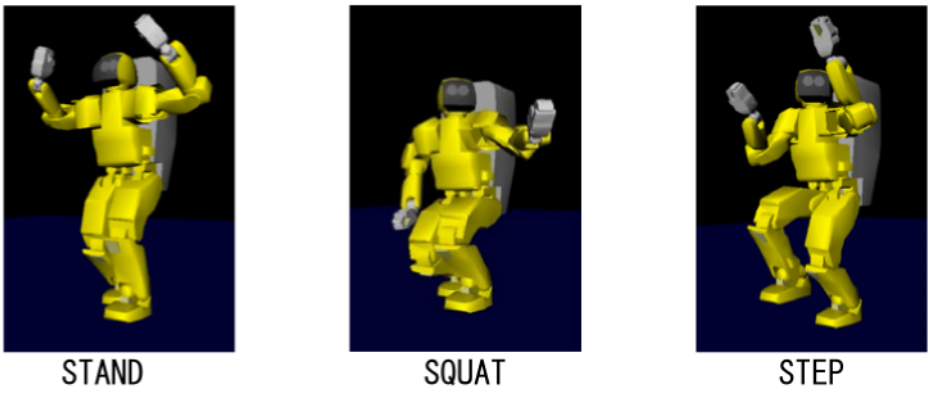


Figure 2.7: Leg motion primitives for dancing. Adapted from [24].

2.2.4 Task descriptors

Generating motion based on the task space rather than the joint space is more flexible as to the ways to represent the motion. Therefore, motion can be represented by high level task descriptors as was done by Dariush *et al.* [23], where high level tasks were organized according to their priorities. This is a special case of inverse kinematics. An example of the descriptors assigned for the upper body is seen in Figure 2.8. This simplification of representation releases some computational burden.

2.2.5 Locus of attention

Human beings coordinate the motion of several degrees of freedom at once, yet, this does not mean that conscious effort is being put to control each one of them. Neo *et al.* [25]

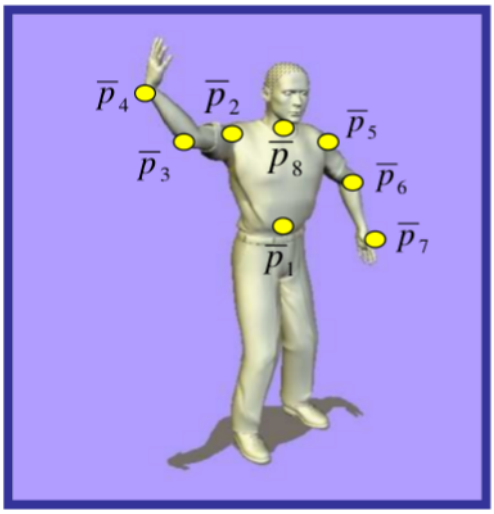
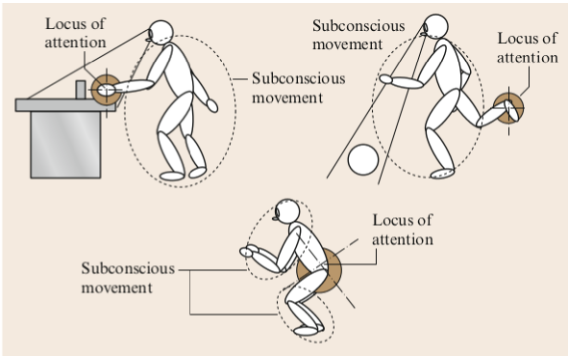
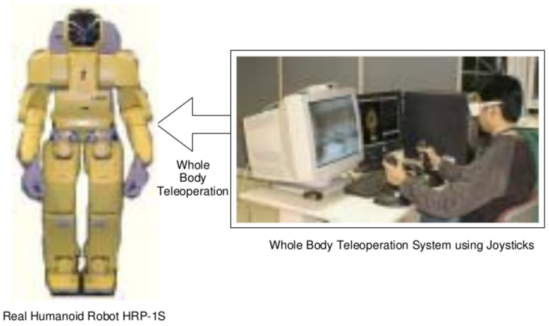


Figure 2.8: Upper body high level task descriptors. Adapted from [23].

used the concept of a "locus of attention" to teleoperate a humanoid in real-time by focusing on specific regions of the body one at a time as shown in Figure 2.9. In their work, the human operator successfully generated full motions for the *HRP-1S* robot with the help of a joystick.



(a)



(b)

Figure 2.9: Teleoperation in real-time by focusing on a locus of attention. (a) Different loci according to the task performed. (b) Joystick-based teleoperation system. Adapted from [2] and [25].

2.2.6 Resolved momentum

When performing fast motions, the dynamic effects cannot be ignored since a lot of momentum is built up. Considering dynamics into the motion, Kajita *et al.* [26] implemented a controller on the *HRP-2* robot which takes into account the linear and angular moments during the motion. In their work, the robot was able to walk and kick without losing balance.

2.3 Conclusions on motion representation

Human movement is locally observed while humanoid movement is globally generated. There are differences in their models which must be addressed before captured human motion can be used to generate humanoid movement. Human bodies are compliant systems which adapt easily, while robots are rigid systems which cannot easily adapt beyond what they've been programmed to do. There are many different ways of interpreting the complexities of human motion and adapting robotics techniques and models to human models so human motion can be used in robotics applications.

The following chapter will focus on the problem of motion transfer from human beings to robots and the specific issues which arise from this conversion.

Chapter 3

Challenges in imitation

Due to structural differences between human beings and robots, human motion cannot be directly transferred to the robot without choosing beforehand which aspects of the motion are worth transferring. Unfortunately, as resembling to human beings as they might be, even the most advanced humanoid robots cannot move completely like a human being.

When the robot motion is generated by pure programming, the limitations of the robot are considered beforehand. When the robot has to learn how to move by itself, it is intrinsically constrained by its limitations. But in imitation, the motions which can actually be performed by the robot are limited by differences to its human counterpart, such as the number of degrees of freedom, link lengths, motor torques, compliance, *etc.*

This chapter points out the main differences between human and humanoid systems and reviews each challenge in implementing imitation and the approaches found in the literature to deal with each one of them.

3.1 Human-humanoid differences

Humanoid robots interact with their environment through their physical bodies. Humanoid bodies are built to reflect the human body with different degrees of fidelity according to the intent of the robot. Some only reproduce the torso and arms, others only the legs, others only an arm, a hand, *etc.* Nevertheless, even robots which try to reproduce the whole human body inevitably have kinematic and dynamic differences from the human body they are trying to represent, not to mention differences in actuation systems and compliance.

3.1.1 Kinematic differences

The number of links and joints is the first main difference. Even the most elaborate humanoid robots have less degrees of freedom (*NAO*: 25, Asimo: 34, HRP-4C: 42) than the human body. This considerably limits the redundancy of the robot in relation to the human body. There are also differences in the joint ranges, velocities and accelerations. While performing imitation, these physical differences have to be taken into account when mapping the registered human movements to the robot. Geometrical differences such as link lengths and number of degrees of freedom must be properly mapped to the robot morphology in question.

Riley *et al.* [15] scaled the captured motion in joint space (direct kinematics) in order to fit the joint ranges of the robot. The scaling was performed globally, which means that the motions were kept large and didn't preserve nuances of the movement. This was addressed by Pollard *et al.* [27], who limited the captured human motion to a range achievable by the robot by locally scaling angles and velocities in order to preserve as much as possible local variations in the motion imitated by the *Sarcos* robot.

Although in [27] they initially scaled each *DOF* separately, in a following work [28] each joint was scaled as a whole (all *DOF* in a joint scaled together) in order to achieve movements which are more consistent with reality. In this follow-up work they also addressed the issue of the robot overall configuration. This is an issue due to the different proportions between the human being and the robot. They inserted a term in the optimization which maintains the relative positions between certain key points on the body. Finally, they added an exponential penalty to their formula in order to keep the joints away from their limits. Finally, Nakaoka *et al.* [24] extended Pollard's scaling method to include also accelerations by adding a term to the equation.

3.1.2 Dynamic differences

As for dynamics, the nature of the structure, such as material properties and mass distribution affect the way the robot moves in relation to its human counterpart. Human beings and humanoid robots differ in their actuation systems and thus in their reachable torques; even the most powerful motors still cannot reach the limits of human motion both in precision and force [24]. Some authors take this difference into account [29] and generate trajectories by optimization while taking into account not only the joint position, velocity and acceleration limits but also torque limits.

Dynamic differences also give rise to a problem in maintaining balance. This will be addressed in Section 3.2.3.

3.2 Humanoid constraints

Some issues arise directly from the design of the humanoid robot's structure. Singularities, self-collisions (interpenetration), balance and time to perform imitation.

3.2.1 Singularities

Singularities in the serial kinematic chain occur when one or more degrees of freedom are lost. This is reflected in a loss of rank of the Jacobian matrix. The most common singularity in humanoid robots is called the gimbal-lock, and it happens when the arm and forearm are aligned and their rotations coincide. In [27], the gimbal-lock was avoided by constraining some degrees of freedom when the robot is moving near the singularity.

3.2.2 Interpenetration

Due to differences between human and humanoid bodies, a certain pose which is achievable by the human might not be possible with the robot, not because of joint ranges but because of differences in the body geometry which might make the robot hit itself when the human does not. This is called the interpenetration or self-collision problem.

When performing optimization of a trajectory as a whole, self-collision can be avoided by adding constraints to the problem. For example, in [28] and [30], the body parts were approximated with spheres and cylinders which cannot intersect as seen in Figure 3.1.

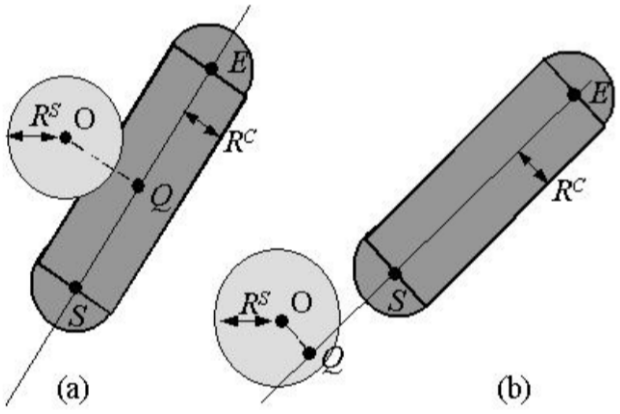


Figure 3.1: Interpenetration avoidance by approximating the body with spheres and cylinders. Adapted from [28].

In [29], interpenetration is avoided using the method in [31], which can be used for any shape, even if it isn't strictly convex, which yields better results than circle/sphere approximations that might exaggerate the shapes and over-constrain the movement.

None of the reviewed approaches which dealt with interpenetration during imitation were performed in real-time, which suggests that a more time-efficient self-collision avoidance must be developed.

Related to the interpenetration problem is the problem of collision with the ground. When changing support, humanoids must be aware of where the ground is located so they don't interact with it in undesired ways, such as kicking it by mistake.

3.2.3 Balance

Kinematics retargetting is enough to perform imitation in very slow velocities (quasistatic). For most computer animation purposes, usually kinematics suffices, since the characters do not need to interact with real-world physics. Yet, some researchers in animation added dynamics to their models in order to increase realism [17]. In robotics, dynamics becomes very important when moving in higher velocities, because inertial forces come into play as the robot body masses and achievable accelerations are different from those of the human. Different dynamics mean that the robot might need to perform slightly different motions than the human, when it comes down to joint angles, in order to keep balance or to faithfully perform a given task.

Methods based on directly mapping each individual joint (direct kinematics) [27, 15] do not take balance into account and put the robot in risk of falling. Indeed, these experiments either focus on the robot upper body or perform whole body imitation but have the robot fixed to some structure so it does not need to stand its own weight.

Some works focus on keeping static balance by maintaining the projection of the CoM within the support polygon. Koenemann *et.al.* [32] were able to imitate human motions with the NAO robot online while keeping static balance by solving inverse kinematics for the legs.

Vukubratovic [33] introduced the concept of a *zero-moment-point* (ZMP) in 1972. This point is defined as the point on the ground about which there is no momentum around the horizontal axes. For balanced motions, the ZMP coincides with the center of pressure (CoP) under the feet [34]. During quasistatic movements, the ZMP is equivalent to the projection of the center of mass (CoM) on the ground. Balance is maintained as long as the ZMP is located within the convex hull delimited by all feet on the ground. Conversely, during movements with high accelerations, the projection of the CoM might come out of the support area while the ZMP can remain there, ensuring stability. This method works best on flat horizontal grounds, not handling well uneven terrain [2].

Dasgupta *et al.* [35] used human motion together with the ZMP concept in order to generate a trajectory in which the robot is balanced (Figure 3.2). In [36], a more realistic trajectory for the ZMP was generated making use of an approximated human model.

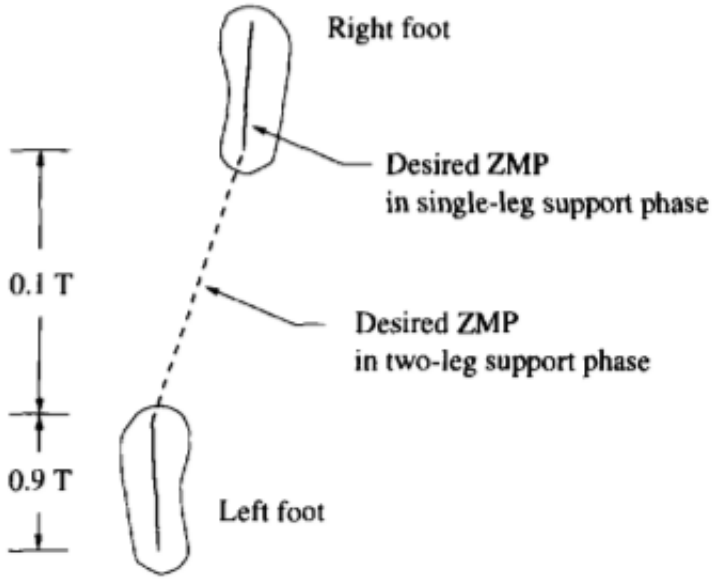


Figure 3.2: *ZMP* trajectory from modified human motion. Adapted from [35].

Nakaoka *et al.* [24] pointed out that leg motion in many humanoid robots is too limited in order to properly imitate human motion. Some limitations include the lack of toes and the impossibility of crossing the legs. Thus, they use motion primitives for the legs and inverse kinematics for the arms. This allows for easier computation of balance since the legs can only assume three different patterns. The balance is ensured by moving the waist in order to guarantee that the *ZMP* is within the robot soles.

3.2.4 Time constraint

Imitation can be performed either offline, online or in real-time. Offline imitation implies that the human motion was captured, processed and afterwards sent to the robot. Online imitation refers to when this process is realized at each time sample. And finally, real-time is an online imitation where time is a hard constraint, which means that the delay between motion capture and generation has to be kept to a minimum.

Real-time imitation is necessary for interactive applications or teleoperation. For that, all the problems mentioned above must be taken into account in real-time, thus it is imperative to develop quick algorithms. To this end, methods which require the optimization of the trajectory as a whole [27, 24, 37] cannot be used in real-time.

Riley *et al.* [38] divided the inverse-kinematics computation into 6 hierarchical chains to speed up computation. The solution takes advantage of the kinematic dependency between body parts to solve the problem numerically. They were able to perform whole body real-time imitation, but the robot was not standing its own weight and thus didn't need to keep balance.

Demicran *et al.* [39] implemented an algorithm which finds the desired torques directly from the task space variables, thus eliminating the need for the calculation of inverse kinematics and allowing for real-time imitation.

In [30], it was suggested that by parametrizing the motion with B-splines, the convergence speed of the trajectory calculation could be increased. Montecillo *et al.* [13] sped-up the retargeting process by developing a "Humanoid normalized model" which can be retargeted online with the use of Kalman filters.

In [32], online imitation was achieved by only taking a few points from the human motion and interpolating trajectories between them. This results in a fluid motion but with visible delay and doesn't retain the nuances of the human motion.

3.3 Summary of the literature background

Chapters 2 and 3 introduced the theoretical background referring to the use of human motion in controlling humanoid robots. Several methods have been introduced and implemented by researchers. A summary of the most relevant contributions to the topic is found on Table 3.1. Although many problems were addressed, many remain barely touched, such as slippage avoidance and change of support. It is sometimes unclear which problems were addressed on each paper and how; in this case, the mention "*NA*" (not addressed) is reported on the table.

We conclude that most works were done offline and several considered only the upper body, which only requires kinematics. No work focused on real-time imitation tried to solve the interpenetration problem. In the cases where dynamics were considered, the *ZMP* criterion is widely applied for balance.

Table 3.1: Comparison of different approaches to imitation

Paper	**Motion capture**	**Time**	**Body parts**	**Physical limits**	**Singularities**	**Interpenetration**	**Balance**	**Validation**
Dasgupta *et al.* 1999 [35]	Marker-based	Offline	Whole body	No	NA	No	ZMP	Simulator
Billard *et al.* 2001 [40]	Marker-based	Offline	Upper body	No	No	No	No	Simulator
Pollard *et al.* 2002 [27]	Marker-based	Offline	Upper body	Angles and velocities	Yes	No	No	Robot
Riley *et al.* 2003 [38]	Color-tracking	Real-time	Whole body	NA	NA	No	No	Robot
Safanova *et al.* 2003 [28]	Marker-based	Offline	Upper body	Angles and velocities	Yes	Yes	No	Robot
Naksuk *et al.* 2005 [41]	Marker-based	Offline	Whole body	NA	Yes	No	ZMP	Simulator
Nakaoka *et al.* 2003, 2006 [24, 10]	Marker-based	Offline	Whole body	Angles, velocities, accelerations	Yes	NA	ZMP	Robot
Ruchanurucks *et al.* 2006 [30]	Marker-based	Offline	Whole body	Angles, velocities, accelerations	NA	Yes	No	Robot
Demircan *et al.* 2008 [39]	Marker-based	Real-time	Upper body	DOF	NA	No	No	Simulator
Do *et al.* 2008 [8]	Marker-based and markerless	Online	Upper body	Angles and velocities	Yes	No	No	Robot
Suleiman *et al.* 2008 [29]	Marker-based	Offline	Upper body	Angles, velocities, torques	NA	Yes	No	Simulator
Dariush *et al.* 2008 [23]	Markerless	Online	Upper body	Angles and velocities	Yes	Yes	ZMP	Robot
Kim *et al.* 2009 [36]	Marker-based	Offline	Whole body	Angles and velocities	NA	NA	ZMP	Robot
Montecillo *et al.* 2010 [13]	Marker-based	Real-time	Whole body	Angles and velocities	NA	No	ZMP	Robot
Koenemann *et al.* 2012 [32]	Embedded	Online	Whole body	Angles and velocities	NA	No	Static	Robot

Chapter 4

Task Specification

The overall literature on human motion imitation by humanoid robots was covered in the previous chapters. This chapter will describe in detail the the approach taken in this research.

Much effort has been put into both kinematic and dynamics imitation. While kinematic approaches are suitable for quasi-static motions, dynamic approaches allow for faster imitation, as they take momentum into account. As well explored as it may seem, kinematic imitation in real time is yet to be considered a closed topic though. Although many works have shown good results offline, real-time approaches still lack in the quality of whole-body imitation, successfully keeping static balance, changing support and meeting time constraints. The present research aims to contribute to the research in real-time whole-body kinematic imitation, providing a general framework which can be used with any humanoid robot and motion capture system.

As mentioned in the previous chapter, imitation involves the reproduction of several aspects of human motion by humanoid robots. When using inverse kinematics to reproduce various aspects at once, the concept of *Task specification* (or *Task classification*, or *Task prioritization*) comes in handy. *Task specification* allows the use of the robot's redundancy to perform several tasks; when these tasks are not compatible, that is, the robot cannot satisfy all of them at once, the tasks of higher priority are solved to the detriment of the lower priority ones [21, 42].

4.1 Types of tasks

A task refers to any geometric goals which must be achieved by the robot. For example, reaching a certain point in the world frame, reaching a particular joint angle, avoiding an obstacle, placing the center of mass in a specific location and so on. Another way of seeing them is as constraints which limit the robot's allowed motion: the robot can place itself in any way as long as it doesn't disturb the task. The tasks dealt with in this research are either equality tasks or optimization tasks. Inequality tasks, which maintain a value within

a desired range, have also been developed [42], but won't be implemented in this research.

Several tasks of each type will be performed. Each type will be described separately below. The mathematical description will be given as well as examples of when that type of task is useful. The way of arranging these tasks according to their priorities will be detailed. And finally, the tasks used in this work will be defined.

4.1.1 Equality tasks

This type of task is the most commonly found when dealing with inverse kinematics. It consists of tracking values which can be described in function of the robot's joints. For example, the position and orientation of an end-effector, the position of the center of mass, and many other key points in the robot's body can be described in function of the joint configuration. Here, the term *direct geometric model* (*DGM*) will be used to refer to the description of any vector $\mathbf{X}$ (*task vector*) in function of the *joint vector* $\mathbf{q}$, thus:

$$\mathbf{X} = DGM(\mathbf{q}) \tag{4.1}$$

While the *DGM* gives a geometric relation between the task vector and the joint vector, the model which relates their velocities is called the *Direct Kinematic Model* (*DKM*). In this work we'll be dealing with discretized motion, so rather than velocity $\dot{\mathbf{q}}$, we'll be talking about position difference $\Delta\mathbf{q}$ (for most purposes they're interchangeable though). The *DKM* is given as follows, where $\Delta\mathbf{q}$ is the vector of joint changes and $\Delta\mathbf{X}$ is the vector of the corresponding change in generalized task coordinates. The matrix $\mathbf{J}(\mathbf{q})$ is called the *Jacobian* and relates the changes in velocity in task space and joint space.

$$\Delta\mathbf{X} = DKM(\Delta\mathbf{q}) \tag{4.2a}$$
$$\Delta\mathbf{X} = \mathbf{J}\Delta\mathbf{q} \tag{4.2b}$$
$$\mathbf{J}(\mathbf{q}) = \frac{\partial\mathbf{X}}{\partial\mathbf{q}} \tag{4.2c}$$

Having $\mathbf{X}^d$ as the desired task vector and $\mathbf{X}^c$ as the current one, the change in the robot's joint configuration $\Delta\mathbf{q}$ needed to bring their difference to zero is given as:

$$\Delta\mathbf{q} = \mathbf{J}^+(\mathbf{X}^d - \mathbf{X}^c) \tag{4.3}$$

Where $\mathbf{J}^+$ is the generalized Moore-Penrose pseudo-inverse of the task's Jacobian $\mathbf{J}(\mathbf{q})$. When the Jacobian matrix is full-rank and square, that is, the number of *DOF* in the task is equal to that of the robot and it is far from singularities, the inverse $\mathbf{J}^{-1}$ can be taken. However, the pseudo-inverse is needed when the inverse of a Jacobian matrix is not unique. For redundant robots, such as humanoids, the Jacobian is not easily invertible since it usually has more columns (joint space) than rows (task space). The pseudo-inverse finds a solution based on least squares which finds the minimum $\Delta\mathbf{q}$ which

satisfies the task. The validity of the pseudo inverse is only true for small intervals, thus it is necessary to break down the final goal into shorter steps.

The pseudo-inverse can be calculated by singular value decomposition (SVD) as follows, where $\mathbf{\Sigma}$ is a diagonal matrix. This simplifies the calculation, since the pseudo-inverse of a diagonal matrix is found by taking the reciprocal of each of its non-zero elements.

$$\mathbf{J} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^t \tag{4.4a}$$
$$\mathbf{J}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^t \tag{4.4b}$$

The pseudo-inverse can also be easily calculated for matrices with linearly independent columns ($\mathbf{J}_L^+$, called the left inverse) or linearly independent rows ($\mathbf{J}_R^+$, called the right inverse), as the case might be.

$$\mathbf{J}_L^+ = (\mathbf{J}^t\mathbf{J})^{-1}\mathbf{J}^t \tag{4.5a}$$
$$\mathbf{J}_R^+ = \mathbf{J}^t(\mathbf{J}\mathbf{J}^t)^{-1} \tag{4.5b}$$

When applied to humanoid imitation of human motions, this type of task can be used, for example, to track end-effector position and orientation. It is also possible to set the robot's CoM to be projected onto a specific point on the ground in order to keep balance. In fact, the tracking of points on the robot is not limited to the end-effector. Not only the end-effector of the arm (usually the hand) can be tracked, but also the elbow or the shoulder can be placed at a specific pose in Cartesian.

4.1.2 Optimization tasks

Some tasks, rather than prescribing a specific point to be reached, require the minimization or maximization of a value. As long as the function is described only in function of the robot's joints, it can be made into an optimization task. These tasks are described by a $\mathbf{Z}$ vector consisting of the weighted gradient of a function.

$$\mathbf{Z}_j = \kappa_j \nabla f_j(\mathbf{q}) \tag{4.6}$$

Here, $f_j(\mathbf{q})$ is the function to be optimized and κ_j is a weight whose value is chosen according to task j's priority. The value of κ_j is positive for maximization tasks and negative for minimization ones.

This type of task is very useful, for example, for minimizing the difference between captured human joint angles and the robot's angles. It is also commonly used in industrial robots to maximize the distance from the joint angles and the joint limits. In summary, anything which can be made into an equality task can be made into an optimization.

Unlike equality tasks, optimization tasks cannot be hard constraints, since the optimal value might not be reached.

4.2 Prioritization

If a task doesn't make use of all of the robot's *DOF*, the extra *DOF* can be used to solve other tasks. Task prioritization arranges these tasks in a hierarchy so that lower priority tasks do not disturb tasks above them. Clearly, the robot has to be redundant with respect to each individual task (have more *DOF* than needed) and for all tasks to have a chance of being performed, the sum of the *DOF* they require must be less or equal the robot's *DOF*.

In fact, humanoid robots are usually redundant with respect to the tasks they are asked to perform. As seen in Figure 4.1, if the task is to place three end-effectors in space, there is an infinite number of joint configurations which can achieve this goal. The space where motions can be performed without perturbing a task is called the task's *null space*. In Figure 4.1, any movement done without moving the three end-effectors is within this task's null-space.

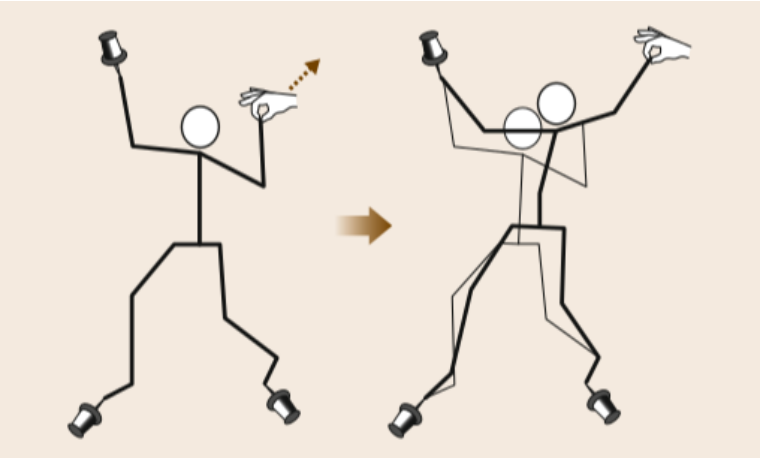


Figure 4.1: Humanoid redundancy. Adapted from [2].

4.2.1 Null space projector

The null space is directly connected to the task's Jacobian and is commonly referred to as the *Jacobian's null space*. The act of performing a task 2 within task 1's null space is called *projecting* the task vector $\mathbf{Z}_2$ into Jacobian $\mathbf{J}_1$'s null space [43]. The projector $\mathbf{P}_j(\mathbf{q})$ of the Jacobian $\mathbf{J}_j(\mathbf{q})$ is defined as:

$$\mathbf{P}_j = \mathbf{I} - \mathbf{J}_j^{+}\mathbf{J}_j \tag{4.7}$$

Here, $\mathbf{I}$ is the identity matrix. The priority of tasks is given so task j is of higher priority than task $j+1$. To use the projector in the solution of inverse kinematics, it suffices to add it multiplied by the joint velocity vector to be projected ($\mathbf{Z}_2$) to the original *IK* ($\Delta\mathbf{q} = \mathbf{J}^+\Delta\mathbf{X}$). In the case of two tasks we have the following solution for the *IK*:

$$\Delta\mathbf{q} = \mathbf{J}_1^+\Delta\mathbf{X}_1 + \mathbf{P}_1\mathbf{Z}_2 \tag{4.8}$$

The projector successfully keeps task 2 (described by $\mathbf{Z}_2$) from interfering in task 1. This can be checked by reversing task 1, that is, multiplying $\mathbf{J}_1$ by the obtained joint vector $\Delta\mathbf{q}$ should result in $\Delta\mathbf{X_1}$.

$$\mathbf{J}_1\Delta\mathbf{q} = \mathbf{J}_1\mathbf{J}_1^+\Delta\mathbf{X}_1 + \mathbf{J}_1\mathbf{P}_1\mathbf{Z}_2 \tag{4.9a}$$
$$\mathbf{J}_1\Delta\mathbf{q} = \Delta\mathbf{X}_1 + \mathbf{J}_1(\mathbf{I} - \mathbf{J}_1^+\mathbf{J}_1)\mathbf{Z}_2 \tag{4.9b}$$
$$\mathbf{J}_1\Delta\mathbf{q} = \Delta\mathbf{X}_1 + \mathbf{J}_1\mathbf{Z}_2 - \mathbf{J}_1\mathbf{J}_1^+\mathbf{J}_1\mathbf{Z}_2 \tag{4.9c}$$
$$\mathbf{J}_1\Delta\mathbf{q} = \Delta\mathbf{X}_1 + \mathbf{J}_1\mathbf{Z}_2 - \mathbf{J}_1\mathbf{Z}_2 \tag{4.9d}$$
$$\mathbf{J}_1\Delta\mathbf{q} = \Delta\mathbf{X}_1 \tag{4.9e}$$

In case task 2 is described by a Jacobian (equality task), we substitute $\mathbf{Z}_2 = \Delta\mathbf{q}_2 = \mathbf{J}_2^+\Delta\mathbf{X}_2$:

$$\Delta\mathbf{q} = \mathbf{J}_1^+\Delta\mathbf{X}_1 + \mathbf{P}_1\mathbf{J}_2^+\Delta\mathbf{X}_2 \tag{4.10}$$

For optimization tasks, $\mathbf{Z}_2$ is the same as described in section 4.1.2:

$$\Delta\mathbf{q} = \mathbf{J}_1^+\Delta\mathbf{X}_1 + \mathbf{P}_1\kappa_2\nabla f_2(\mathbf{q}) \tag{4.11}$$

The null space projector can be used in nests in order to insert any number of tasks into the ones before, as long as tasks which will be above others in priority are described by Jacobians, because they must have a null space. This means that optimization tasks cannot make use of the projector to be higher than others, although, as seen above, they can be in the lowest priority. For 3 strict tasks:

$$\Delta\mathbf{q} = \mathbf{J}_1^+\Delta\mathbf{X}_1 + \mathbf{P}_1(\mathbf{J}_2^+\Delta\mathbf{X}_2 + \mathbf{P}_2\mathbf{J}_3^+\Delta\mathbf{X}_3) \tag{4.12}$$

This time however, upon checking the interference between tasks, an unfortunate consequence is discovered. Let's first check the completion of task 1, as done above. Recall that $\mathbf{J}_1\mathbf{P}_1\mathbf{Z} = 0$ for any $\mathbf{Z}$, as shown in equation 4.9.

$$\mathbf{J}_1\Delta\mathbf{q} = \mathbf{J}_1\mathbf{J}_1^+\Delta\mathbf{X}_1 + \mathbf{J}_1\mathbf{P}_1\mathbf{J}_2^+\Delta\mathbf{X}_2 + \mathbf{J}_1\mathbf{P}_1\mathbf{P}_2\mathbf{J}_3^+\Delta\mathbf{X}_3 \tag{4.13a}$$
$$\mathbf{J}_1\Delta\mathbf{q} = \Delta\mathbf{X}_1 \tag{4.13b}$$

It is confirmed that task 3 does not interfere with task 1. However, when checking for task 2, we notice that the introduction of task 3 might influence its completion. Therefore, a method which insures the hierarchy of all tasks is needed.

$$\mathbf{J}_2\Delta\mathbf{q} = \mathbf{J}_2\mathbf{J}_1^+\Delta\mathbf{X}_1 + \mathbf{J}_2\mathbf{P}_1\mathbf{J}_2^+\Delta\mathbf{X}_2 + \mathbf{J}_2\mathbf{P}_1\mathbf{P}_2\mathbf{J}_3^+\Delta\mathbf{X}_3 \tag{4.14a}$$

$$\mathbf{J}_2\Delta\mathbf{q} \neq \Delta\mathbf{X}_2 \tag{4.14b}$$

4.2.2 Ensuring priority order

As seen above, when adding more than 2 tasks, simply nesting null space projectors is enough to insure the 1st task will not be perturbed, but nothing can be guaranteed for the 2nd task onwards. Therefore, a way of ensuring each new task is projected into the null space common to all previous tasks is necessary.

Baerlocher *et al.* [44] proposed a change in the way the null space projector is used in order to more efficiently handle conflicting tasks and ensure the hierarchy of priorities. For two tasks, equation 4.10 is adapted in 2 steps. First, the influence of the 1st task on the 2nd is subtracted ($-\mathbf{J}_2\Delta\mathbf{q}_1$):

$$\Delta\mathbf{q} = \mathbf{J}_1^+\Delta\mathbf{X}_1 + \mathbf{P}_1\mathbf{J}_2^+(\Delta\mathbf{X}_2 - \mathbf{J}_2\mathbf{J}_1^+\Delta\mathbf{X}_1) \tag{4.15}$$

Then the 2nd task's Jacobian itself is limited to stay within the null space of the 1st task: $\mathbf{J}_2^+ \rightarrow (\mathbf{J}_2\mathbf{P}_1)^+$. Once this is done, the pre-multiplication by $\mathbf{P}_1$ is not necessary any longer. Hence, the completely adapted equation becomes:

$$\Delta\mathbf{q} = \mathbf{J}_1^+\Delta\mathbf{X}_1 + (\mathbf{J}_2\mathbf{P}_1)^+(\Delta\mathbf{X}_2 - \mathbf{J}_2\mathbf{J}_1^+\Delta\mathbf{X}_1) \tag{4.16}$$

This new formulation can be used for any number of tasks while ensuring that the lower priority tasks don't perturb the ones above them. The iterative process considering N tasks arranged by their respective priorities is the following, for $j = 1..N$.

$$\begin{cases} \Delta\mathbf{q}_0 &= 0 \\ \Delta\mathbf{q}_{j+1} &= \Delta\mathbf{q}_j + (\mathbf{J}_{j+1}\mathbf{P}_j^a)^+(\Delta\mathbf{X}_{j+1} - \mathbf{J}_{j+1}\boldsymbol{\Delta}\mathbf{q}_j) \end{cases} \tag{4.17}$$

Here, $\mathbf{P}_j^a$ is the projector of Jacobian $\mathbf{J}_j^a$, which is called the *augmented Jacobian*. It consists of piling up all the Jacobians up to task j. The augmented Jacobian takes all higher priority tasks into account, making sure that they won't be disturbed by the task being introduced.

$$\mathbf{J}_j^a = \begin{pmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_j \end{pmatrix} \tag{4.18}$$

4.2.3 Including optimization tasks

After N tasks with rigid constraints have been arranged, we can introduce M optimization constraints and get the following equation. The priority of each optimization task is given by the magnitude of its weight κ_j, recalling that κ_j is positive for maximization tasks and negative for minimization ones.

$$\Delta\mathbf{q} = \Delta\mathbf{q}_N + \mathbf{P}_N^a \sum_{j=1}^{M} \kappa_j \nabla f_j(\mathbf{q}) \tag{4.19}$$

We can add as many tasks as strict or optimization constraints as wished, checking beforehand that the redundancy order of the considered robot with respect to the considered tasks descriptions is sufficient.

4.3 Tasks in imitation

When imitating human motion, it is necessary to first clarify exactly which aspects of the motion are to be copied. Among these, it is also necessary to specify which ones are the most important. Is tracking the hand position in the world frame more important than tracking the elbow angle? Does the robot need to keep balance or is it attached to some structure? The answers to these questions will influence in the quantity, type and priority of the tasks chosen.

In this work, four general tasks will be considered: end-effector tracking, joint tracking, keeping balance and avoiding joint limits. Each task will be discussed individually.

4.3.1 Joint limits avoidance

It might seem counter-intuitive at first, but no matter what is the goal of the imitation, staying away from joint limits should always be the task with the highest priority of all. In fact, in any robotics application, avoiding joint limits is usually of the highest importance. That is because if the inverse kinematics' solution is not within limits, that value will not be physically feasible. If the resulting joint vector is limited after the solution is found, the overall configuration is changed and nothing insures the new configuration will comply with other tasks. Placing limit avoidance in the highest priority assures that all other tasks will be physically achievable.

Staying away from the joint limits can be described as staying as close as possible to the median value for the joint position. Thus, the task can be described as a minimization between the current joint value q_i^c and the median joint value q_i^{med}. The function is given

as:

$$f_{limits}(\mathbf{q}) = \sum_{i=1}^{n} \left(\frac{q_i^c - q_i^{med}}{q_i^{max} - q_i^{min}} \right)^2 \tag{4.20}$$

Consequently, after differentiating the $f_{limits}(\mathbf{q})$ according to equation 4.6, each element i of the task vector becomes:

$$\mathbf{Z}_{limits}(i) = 2\kappa_{limits} \frac{q_i^c - q_i^{med}}{(q_i^{max} - q_i^{min})^2} \tag{4.21}$$

Although fit for many motions, it is not possible to place this as the highest priority task using the prioritization method described, since it is an optimization task. Another drawback is giving priority to the median position when in fact any position within the range should be equally allowed.

A different way of dealing with joint limits is proposed by Baerlocher [44]. In this method, if the pseudo-inverse finds a solution which doesn't fit into joint limits, the joints in question are fixed at their limit value and the *IK* is resolved using only the other joints. After the *IK* has been numerically solved, for the surpassing joints the following two measures are taken:

- Fix those joints' values to the closest limits
- Zero the columns on all Jacobians which correspond to those joints

The *IK* is then solved again. This loop is repeated as many times as needed until a solution is found where all joints fit within their limits. This method successfully avoids joint limits, but can be too constraining as it might quickly fix many joints and reduce the available redundancy for other tasks. When used in conjunction with the minimization task above however, fewer joints can be fixed and leave more room for other tasks while insuring the joint limits will be avoided.

4.3.2 Balance

Two types of balance can be kept making use of inverse kinematics: static balance tracking the projection of the *CoM* or dynamic balance tracking the *ZMP*. For either point being tracked, the robot will remain balanced as long as the point falls within the convex hull delimited by the support feet.

It is possible to use equality tasks to place these points on a specific position on the floor. The desired position $\mathbf{X}_{CoM}^d$ has the two horizontal coordinates $[x, y]$. To bring the current projection $\mathbf{X}_{CoM}^c$ to that desired point, the robot will need to move its joints by $\Delta\mathbf{q}_{CoM}$:

$$\Delta\mathbf{q}_{CoM} = \mathbf{J}_{CoM}^{+}(\mathbf{X}_{CoM}^d - \mathbf{X}_{CoM}^c) \tag{4.22}$$

We can extend the static criterion to *ZMP* criterion by substituting the task space vector for $\mathbf{X}^d_{ZMP}$, where the *ZMP*'s absolute position in the horizontal plane can be obtained from the absolute position of the *CoM* and its derivatives.

$$\begin{cases} x^c_{ZMP} &= x^c_{CoM} - \frac{z^c_{CoM} - z^c_{ZMP}}{g + \ddot{z}^c_{CoM}} \ddot{x}^c_{CoM} \\ y^c_{ZMP} &= y^c_{CoM} - \frac{z^c_{CoM} - z^c_{ZMP}}{g + \ddot{z}^c_{CoM}} \ddot{y}^c_{CoM} \end{cases} \tag{4.23}$$

Where z^c_{ZMP} is the level of the floor and $g = -9.81$ m/s^2 the gravity acceleration.

Fixing these points onto a specific position on the floor is overconstraining and leaves less free *DOF* for the other tasks. However, this precise control over the point positioning is advantageous for choosing safer positions far from the foot's edges and for avoiding points too far from the ankle which would require too much torque to hold the whole body.

The balance task can also be described as an optimization task minimizing the distance from the center of the support. This kind of task does not have a high priority though. As an inequality task, the points could be directed to any point within the support area instead of a specific point.

4.3.3 End-effector tracking

The task vector $\Delta\mathbf{X}_{tracking}$ is given by the difference between the desired and the current poses of the end-effectors. Both positions and orientations can be considered. Considering that there are $1..e$ end-effectors being tracked, we have:

$$\Delta\mathbf{X}_{tracking} = \begin{bmatrix} \Delta\mathbf{X}_1 \\ \Delta\mathbf{X}_2 \\ \vdots \\ \Delta\mathbf{X}_e \end{bmatrix} \tag{4.24}$$

The Jacobian $\mathbf{J}_{tracking}$ is found by concatenating the end-effector Jacobians for the current configuration $\mathbf{q}^c$.

$$\mathbf{J}_{tracking} = \begin{bmatrix} \mathbf{J}_1(\mathbf{q^c}) \\ \mathbf{J}_2(\mathbf{q^c}) \\ \vdots \\ \mathbf{J}_e(\mathbf{q^c}) \end{bmatrix} \tag{4.25}$$

4.3.4 Joint trajectories tracking

Imitation based on direct kinematics consists of extracting angles from the human motion which correspond to the robot's *DOF* and sending this data directly to the robot. As

explained on Section 2.2.1, this approach is limited and does not insure balance. Nevertheless, keeping the robot joint values as close as possible to that of the actor's improves the quality of the imitation, making the robot move in a similar manner to the human. Thus, it is interesting to minimize the differences in joint angles between the human actor and the humanoid robot as an optimization constraint.

$$f_{joints}(\mathbf{q}) = \sum_{i=1}^{k} (q_i^c - q_i^d)^2 \tag{4.26}$$

where $\mathbf{q}^d$ is the human generalized joint vector matching the dimension and the joints of the robot's joints and $\mathbf{q}^c$ their current value. Each angle q_i denotes the i^{th} component of vector $\mathbf{q}$. Note that only $k < n$ joints can be tracked, as not always all the robot's DOF can be correlated to an angle on the human, depending on the robot's kinematics and on the motion capture system. Differentiating, each element of the optimization vector becomes:

$$\mathbf{Z}_{joint}[i] = 2\kappa_{joint}(q_i^c - q_i^d) \tag{4.27}$$

4.4 Summary

This chapter described the general formulation for inverse kinematics with task specification. Four tasks for human motion imitation by humanoid robots were explained, these being: avoiding joint limits, keeping balance, tracking end-effectors and tracking joint values. These tasks can be described either as equality tasks or optimization tasks. The priority of tasks is insured by projecting lower priority tasks into high priority tasks' null spaces with the help of the augmented Jacobian. To improve the convergence of the limit avoidance task, a clamping loop can be added.

Chapter 5

Imitation in single support

The first stage of this research was to implement motion imitation while on single support, as shown on Figure 5.1. This implementation helps validate the method of task specification, as well as the processing of human data. Single support was chosen over double support because the *NAO* robot never loses balance while moving arms only, therefore it wouldn't be enough to check whether the balance constraints work well.

In this chapter, the experimental tools used will be introduced and the implementation of single support imitation will be detailed. In the results section, various aspects of the imitation will be discussed.

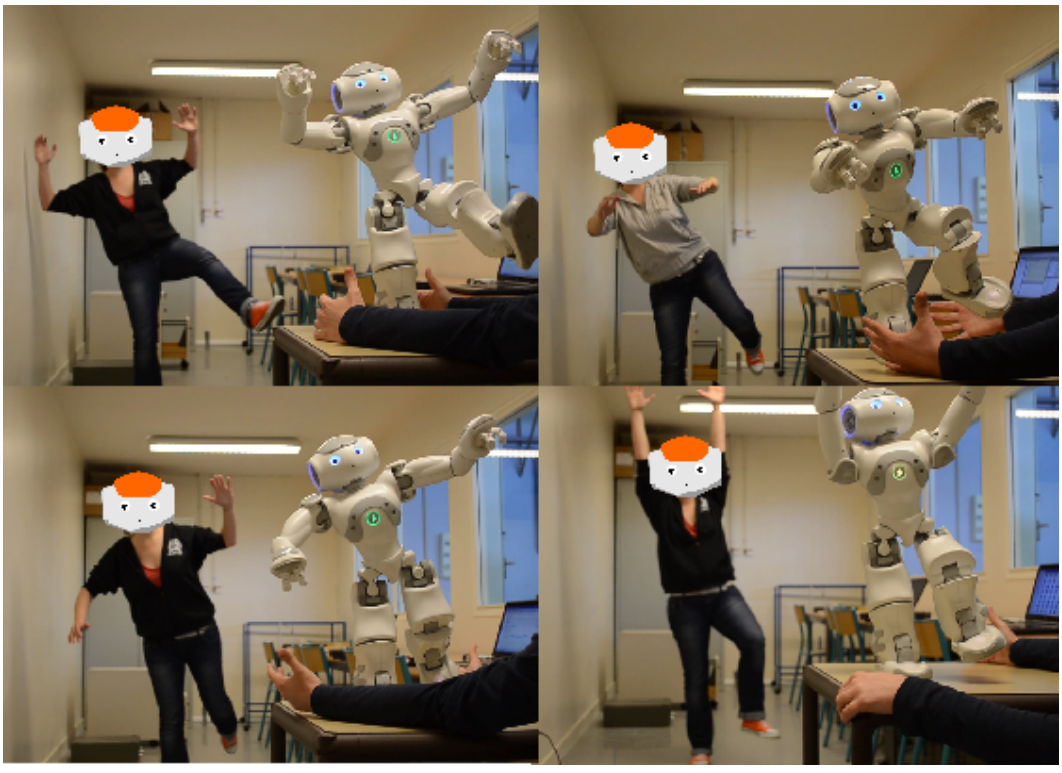


Figure 5.1: Example of poses achievable with online imitation in single support.

5.1 Experimental tools

5.1.1 Robotic platform

The *NAO* robot is a 25 *DOF* humanoid robot produced by *Aldebaran Robotics* [45], shown in Figure 5.2. It is equipped with a series of sensors: microphone, cameras, sonars, infra-red, gyroscope, accelerometer, rotary encoders in all its motors, force sensors on its feet and contact sensors on various body parts.

Figure 5.2: *NAO* humanoid robot. Adapted from [46].

The robot's embedded software, *NAOqi*, provides a framework so the robot can be programmed in various operating systems, and in various languages, including *C++*, *Python* and *Matlab*. The manufacturer also provides a software with which the robot can be programmed through a simple graphic interface, called *Coreographe*. The dynamics simulator *Webots*, powered by *Cyberbotics*, can be used to simulate the robot's behaviour in real physics.

For this research, a program was written in *C++* and compiled with *CMake*, a cross-platform build system, to make use of *NAOqi* libraries. The program runs in a computer which communicates with the robot via *wi-fi*.

5.1.2 Motion capture systems

Two different motion capture systems were used in this project, one is marker-based and the other is markerless.

5.1.2.1 Markerless

The markerless motion capture system used in this research is the *Microsoft Kinect* [16]. As shown in Figure 5.3, this system comes equipped with an *RGB* camera, an *IR* depth sensor and a microphone array. In order to detect the human actor, the camera and the depth sensor are used in conjunction to extract position and orientation information about the human body.

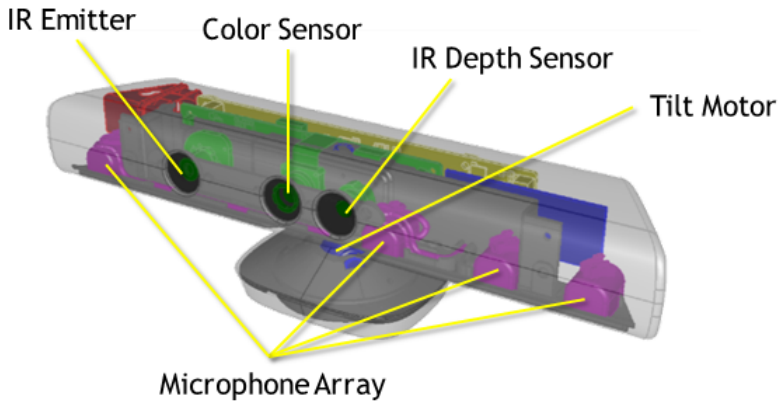


Figure 5.3: Sensors included in the *Microsoft Kinect*. Adapted from [16].

To detect the human skeleton, the *OpenNI* [47] development framework was used. This open source framework offers libraries which facilitate the extraction of meaningful data from *natural interaction* devices, such as the *Kinect*. Among these libraries, the *NiTE* middleware was used to track the human skeleton. This library returns 15 joint frames for the human body, including position and orientation for each of them, as shown in Figure 5.4.

Figure 5.4: Skeleton tracking with *NITE*.

The *NiTE* middleware offers the source code for several sample programs in *C++*. The sample *Players* was modified to extract the skeleton data and translate it into the data needed to solve the *IK*.

5.1.2.2 Marker-based

The marker-based motion capture system used in this research is the *ARTtrack* system based on 8 infra-red cameras and passive retro-reflective markers [48]. The *3D* position of each marker is calculated with the principles of stereovision, as shown in Figure 5.5.

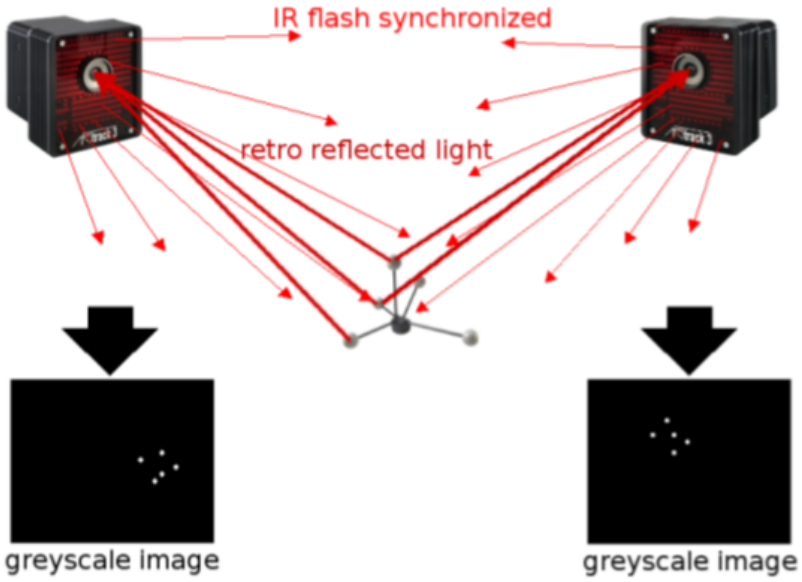


Figure 5.5: Principle of optical tracking in the *ARTtrack* system. Adapted from [48].

A different rigid arrangement of 4 markers is attached to each tracked body part, as shown on Figure 5.6. The software can tell apart each rigid part and return the position and orientation (6 DOF) of each body segment and joint for each time instant in $60Hz$.

This system is different from traditional marker-based systems since the markers are arranged in rigid bodies instead of being attached to the human body directly. This means that the *ISB* recommendations do not apply to this system and a way of extracting joint frames from the captured data must be proposed if the approximation provided by *ARTtrack* is not sufficient. During this research, the joint positions provided by the software were used directly.

During this research, it wasn't possible to access the *ARTtrack* system online, therefore, this system was only used for offline validation.

Figure 5.6: *ARTtrack*'s markers placed on the human.

5.1.3 Other software

Besides the previously mentioned software, the numerical computing environment *Matlab* [49] was used to model the robot and analyse results. Also, the *Qt* [50] framework was used to create a *graphical user interface* (*GUI*) to aid in the development and to facilitate access to the program developed (Appendix A). All the implementation was done under the *Linux* operating system.

5.2 Implementation

The implementation began by modelling the robot's kinematics in order to obtain a relation between its joint and task spaces. Once the robot's geometry was understood, the captured human data was scaled to the robot's size, and its desired features tracked with task specification *IK*.

5.2.1 Robot model

5.2.1.1 Geometric and kinematic models

During the course of this research, two robots were used: one *NAO H25 V3.3* with a head *V4.0* and one fully *V4.0*. While these robots have an identical kinematic description for their effectors, their masses greatly differ. *NAO H25* possesses 25 *DOF* in total, however, for the purposes of this research, the hands (open/close) will be ignored. Thus, we will

be controlling 23 *DOF* in total (vector $\mathbf{q}$), arranged as in Fig. 5.7.

$$\mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_{23} \end{bmatrix} \tag{5.1}$$

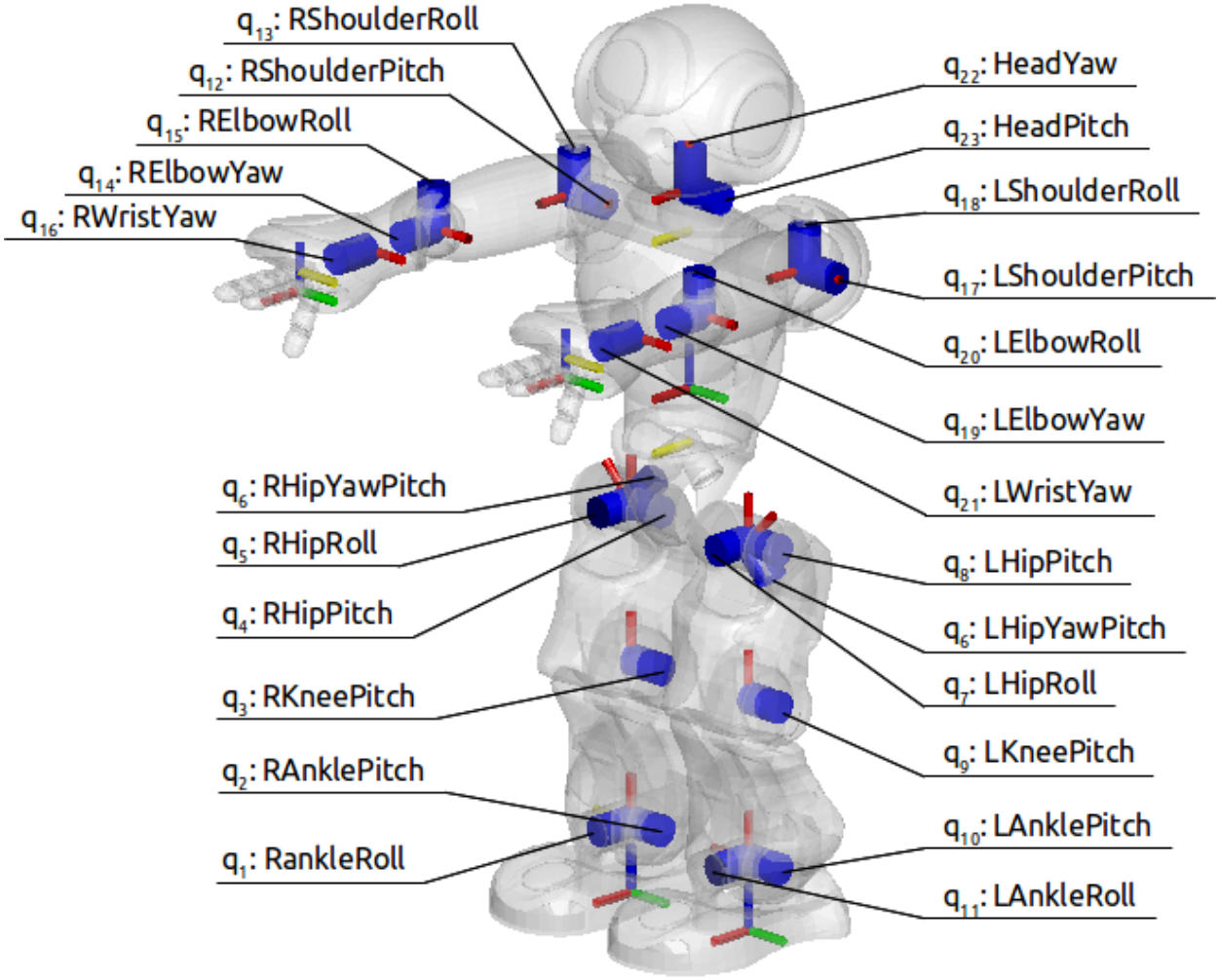


Figure 5.7: *NAO* Robot's *DOF* as arranged in the joint vector.

The robot's geometric model was calculated using *Modified Denavit-Hartenberg* (*M-DH*) parameters according to the guidelines in [21]. The most relevant aspects of this parametrization for the current work are here described. For a tree structure without prismatic joints where each link ℓ has an antecedent link a and a subsequent link s, the frames are assigned as:

- Z_ℓ axis: joint ℓ's rotational axis
- X_ℓ axis: orthogonal to Z_ℓ and Z_s
- Y_ℓ axis: $Z_\ell \times X_\ell$

Since this robot is a tree structure, some links have more than one subsequent link. In addition, it might be convenient to include frames in the model which are not necessarily defined according to the previous rules. In these cases, an axis U_s can be introduced. This axis will play the role which X_ℓ was previously playing, while we are free to set X_ℓ as wished. Axis U_s is indexed according to the subsequent link because several of such axes might be associated to one Z_ℓ.

- If X_ℓ is not defined as above, define a U_s axis orthogonal to Z_ℓ and Z_s.

After all frames have been defined, the *M-DH* parameters describing their relation can be taken. Each parameter describes either an angle or a distance between 2 axes about another axis. In total there are 6 parameters used to define the relation between 2 frames, given as follows.

- α_ℓ: angle from Z_a to Z_ℓ about X_a
- d_ℓ: distance from Z_a to Z_ℓ along X_a
- θ_ℓ: angle from X_a to X_ℓ about Z_ℓ
- r_ℓ: distance from X_a to X_ℓ along Z_ℓ
- γ_ℓ: angle from X_ℓ to U_s about Z_ℓ
- b_ℓ: distance from X_ℓ to U_s along Z_ℓ

When the U_s axis is not necessary, it is the same as to say that it is equal to X_ℓ, so the parameters γ_ℓ and b_ℓ are both zero (so the relation between frames is fully described by 4 parameters). These parameters can then be used to get the homogeneous transformation ${}^a\mathbf{T}_\ell$ from frame a to frame ℓ as follows. Note that for compactness, C represents *cosine* and S represents *sine*:

$$ {}^a\mathbf{T}_\ell = \begin{bmatrix} C\gamma_\ell C\theta_\ell - S\gamma_\ell C\alpha_\ell S\theta_\ell & -C\gamma_\ell S\theta_\ell - S\gamma_\ell C\alpha_\ell C\theta_\ell & S\gamma_\ell S\alpha_\ell & d_\ell C\gamma_\ell + r_\ell S\gamma_\ell S\alpha_\ell \\ S\gamma_\ell C\theta_j + C\gamma_j C\alpha_j S\theta_j & -S\gamma_j S\theta_j + C\gamma_j C\alpha_j C\theta_j & -C\gamma_j S\alpha_j & d_j S\gamma_j - r_j C\gamma_j S\alpha_j \\ S\alpha_j S\theta_j & S\alpha_j C\theta_j & C\alpha_j & r_j C\alpha_j + b_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2} $$

The parameters used follow the nomenclature and values in the robot's documentation [45] and are seen in Figure 5.8. The resulting tree structure is detailed in Figure 5.9 and the corresponding parameters are shown on Table 5.1. The X, Y, Z axes are coloured according to the *RGB* convention and the U axes are shown in yellow. When the Z axis (revolution axis) belongs to a *DOF*, it is represented with a thick cylinder. Although there are 24 movable axes, frames 6 and 8 are coupled and controlled by the same motor, so the robot has only 23 controllable *DOF*.

In this preliminary implementation, the robot is kept balanced on the right foot. Therefore, it was modelled considering that the right foot is fixed to the floor, so frame 0 is taken as its sole. Frames 1 to 6 describe the *DOF* of the right leg from *RAnkleRoll* to *RHipYawPitch*. Frame 7 is fixed to its antecedent and corresponds to the torso described in the documentation. The other chains are described succeeding frame 7 for convenience

and simplicity, each one ending in its respective end-effector (*LFoot*, *RHand*, *LHand*, *Head*). Note that the five fixed frames $(0, 7, 14, 20, 27)$ are oriented the same way as in the documentation, so they can also be used directly with functions in *NAOqi* such as *getCOM* and *getPosition*.

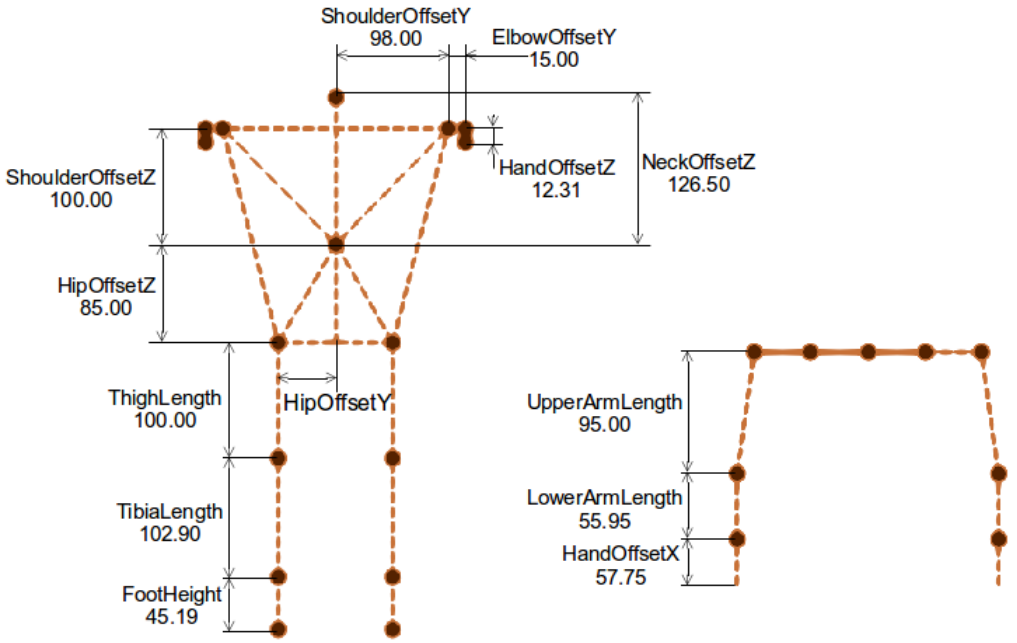


Figure 5.8: Measures of *NAO* robot's body.

As previously explained, the *DGM* gives the task-space poses according to the joint variables. The right foot is kept immobile on the floor at all times, thus it is taken as the absolute frame of reference. It is also worth noting that here the head is not considered as an end-effector. Thus, only 3 end-effectors are considered: the left ankle, the left hand and the right hand.

Although the orientation of the end-effectors can be found, only their Cartesian positions with respect to the right foot ${}^{RFoot}\mathbf{P}_j$ (3 dimensions $\times$ 3 end-effectors $=$ 9 *DOF*) will be tracked. This is because whole poses would take up to $6 \times 3 = 18$ *DOF*, leaving only 5 *DOF* for the other tasks, of which 2 belong to the head chain. Thus, from the *DGM* we have:

$$ {}^{RFoot}\mathbf{P}_{LAnkle} = DGM(\mathbf{q}) \tag{5.3a} $$

$$ {}^{RFoot}\mathbf{P}_{RHand} = DGM(\mathbf{q}) \tag{5.3b} $$

$$ {}^{RFoot}\mathbf{P}_{LHand} = DGM(\mathbf{q}) \tag{5.3c} $$

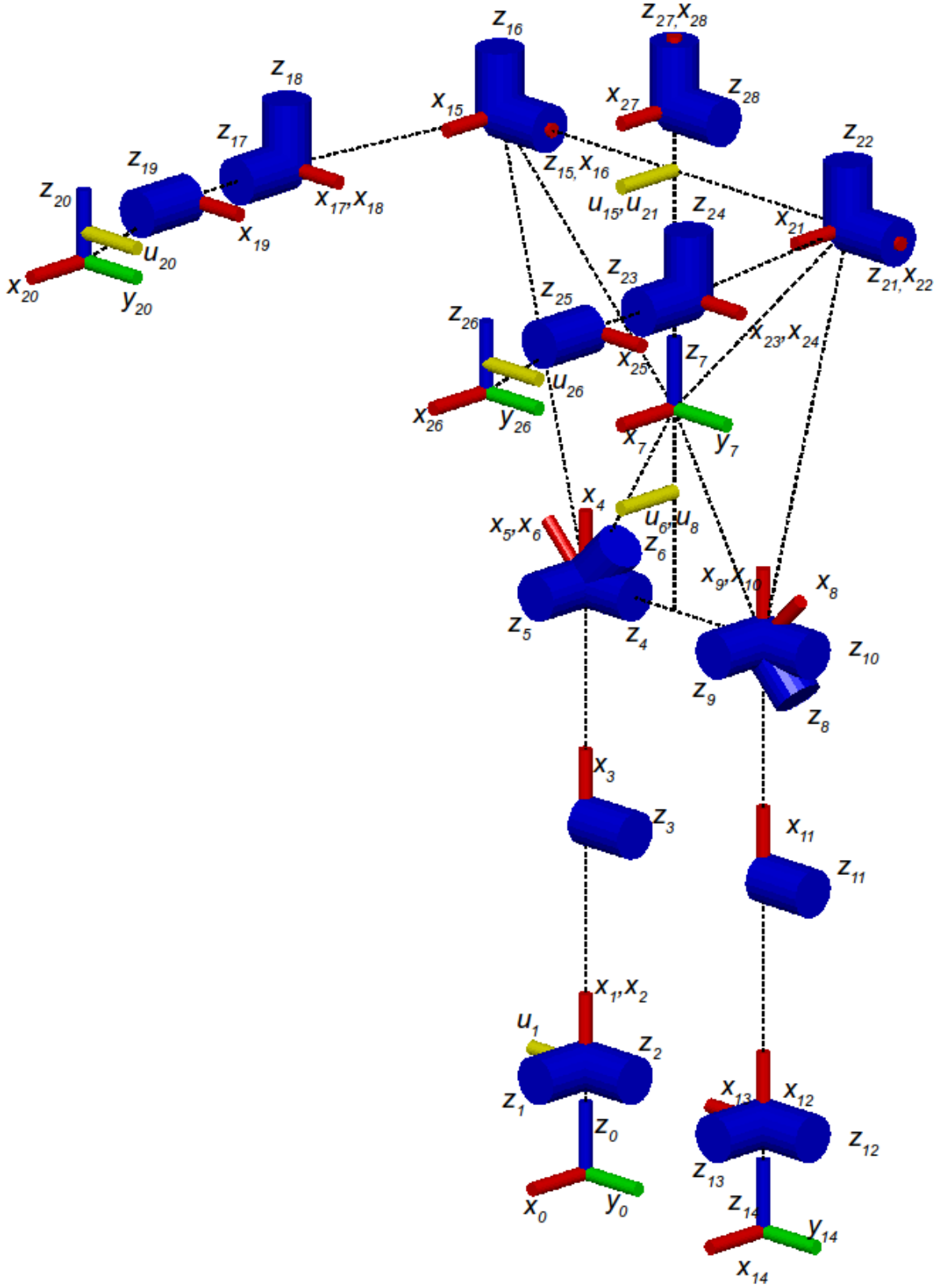


Figure 5.9: *NAO*'s kinematic chain based on the right foot, modelled with the *M-DH* parameters on Table 5.1.

Table 5.1: *NAO*'s *M-DH* parameters based on the right foot.

ℓ	a_ℓ	σ_ℓ	γ_ℓ	b_ℓ	α_ℓ	d_ℓ	θ_ℓ	r_ℓ
1	0	0	$-\frac{\pi}{2}$	$FootHeight$	$-\frac{\pi}{2}$	0	$-\frac{\pi}{2} - RAnkleRoll$	0
2	1	0			$\frac{\pi}{2}$	0	$-RAnklePitch$	0
3	2	0			0	$TibiaLength$	$-RKneePitch$	0
4	3	0			0	$ThighLength$	$-RHipPitch$	0
5	4	0			$-\frac{\pi}{2}$	0	$\frac{\pi}{4} - RHipRoll$	0
6	5	0			$\frac{\pi}{2}$	0	$-(L/R)HipYawPitch$	0
7	6	$2(Torso)$	$\frac{\pi}{2}$	$\sqrt{HipOffsetY^2 + HipOffsetY^2}$	$\frac{\pi}{4}$	0	0	$\frac{HipOffsetZ - HipOffsetY}{\sqrt{HipOffsetY^2 + HipOffsetY^2}}$
8	7	0	0	$-(HipOffsetZ - HipOffsetY)$	$-\frac{3\pi}{4}$	0	$-\frac{\pi}{2} + (L/R)HipYawPitch$	
9	8	0			$-\frac{\pi}{2}$	0	$\frac{\pi}{4} + LHipRoll$	0
10	9	0			$\frac{\pi}{2}$	0	$LHipPitch$	0
11	10	0			0	$-ThighLength$	$LKneePitch$	0
12	11	0			0	$-TibiaLength$	$LAnklePitch$	0
13	12	0			$-\frac{\pi}{2}$	0	$\frac{\pi}{2} + LAnkleRoll$	0
14	13	$2(LFoot)$			$\frac{\pi}{2}$	0	$\frac{\pi}{2}$	$-FootHeight$
15	7	0	0	$ShoulderOffsetZ$	$-\frac{\pi}{2}$	0	$RShoulderPitch$	$-ShoulderOffsetY$
16	15	0			$\frac{\pi}{2}$	0	$\frac{\pi}{2} + RShoulderRoll$	0
17	16	0			$\frac{\pi}{2}$	$-ElbowOffsetY$	$RElbowYaw$	$UpperArmLength$
18	17	0			$-\frac{\pi}{2}$	0	$RElbowRoll$	0
19	18	0			$\frac{\pi}{2}$	0	$RWristYaw$	$LowerArmLength$
20	19	$2(RHand)$	0	$HandOffsetX$	$-\frac{\pi}{2}$	0	$-\frac{\pi}{2}$	$-HandOffsetZ$
21	7	0	0	$ShoulderOffsetZ$	$-\frac{\pi}{2}$	0	$LShoulderPitch$	$ShoulderOffsetY$
22	21	0			$\frac{\pi}{2}$	0	$\frac{\pi}{2} + LShoulderRoll$	0
23	22	0			$\frac{\pi}{2}$	$ElbowOffsetY$	$LElbowYaw$	$UpperArmLength$
24	23	0			$-\frac{\pi}{2}$	0	$LElbowRoll$	0
25	24	0			$\frac{\pi}{2}$	0	$LWristYaw$	$LowerArmLength$
26	25	$2(LHand)$	0	$HandOffsetX$	$-\frac{\pi}{2}$	0	$-\frac{\pi}{2}$	$-HandOffsetZ$
27	7	0			0	0	$HeadYaw$	$NeckOffsetZ$
28	27	0			$-\frac{\pi}{2}$	0	$-\frac{\pi}{2} + HeadPitch$	0

From the geometric model, the direct kinematic model is found by partial derivatives as follows. The dimension of each Jacobian is $[3 \times 23]$.

$$^{RFoot}\mathbf{J}_{LAnkle} = \frac{\partial^{RFoot}\mathbf{P}_{LAnkle}}{\partial \mathbf{q}} \tag{5.4a}$$

$$^{RFoot}\mathbf{J}_{RHand} = \frac{\partial^{RFoot}\mathbf{P}_{RHand}}{\partial \mathbf{q}} \tag{5.4b}$$

$$^{RFoot}\mathbf{J}_{LHand} = \frac{\partial^{RFoot}\mathbf{P}_{LHand}}{\partial \mathbf{q}} \tag{5.4c}$$

The modelling was implemented on *Matlab*, since this language allows for easier plotting and checking the model. A script was developed where the user can input the *M-DH* parameters and immediately get a plot like Figures 5.7 or 5.9, including fixed frames, *DOF*s, lines connecting adjacent frames and, for the *NAO* robot, even its body parts. A *GUI* where the user can change each *DOF* and see the resulting configuration was also implemented on *Matlab*.

5.2.1.2 Masses

To keep static balance during imitation, it is necessary to track the *CoM* of the robot to ensure it fulfils balance conditions. The *CoM* position is given as:

$$^{RFoot}\mathbf{P}_{CoM}(\mathbf{q}) = \frac{\sum_{\ell=1}^{n} {}^{RFoot}\mathbf{P}_{CoM_\ell}(\mathbf{q}) m_\ell}{\sum_{i=1}^{n} m_\ell} \tag{5.5}$$

Where n is the number of moving masses in the robot's body. In this case, it is equal to the number of *DOF* ($n = 23$), since each *DOF* moves a mass m_ℓ which must be taken into account. The position of each mass w.r.t. the absolute frame is given as $^{RFoot}\mathbf{P}_{CoM_\ell}$.

The masses values and positions, as given by the documentation, are shown on Figure 5.10, where each mass point is represented by a yellow sphere centred at its position with a radius proportional to its magnitude. The red cross represents the robot's *CoM* and the black cross its projection on the plane of the right foot's sole.

The same way as for the end-effectors, the Jacobian of the *CoM* projection onto the floor (x and y coordinates of the right foot's frame) is found. Its dimension is $[2 \times 23]$.

$$^{RFoot}\mathbf{J}_{CoM_{xy}} = \frac{\partial^{RFoot}\mathbf{P}_{CoM_{xy}}}{\partial \mathbf{q}} \tag{5.6}$$

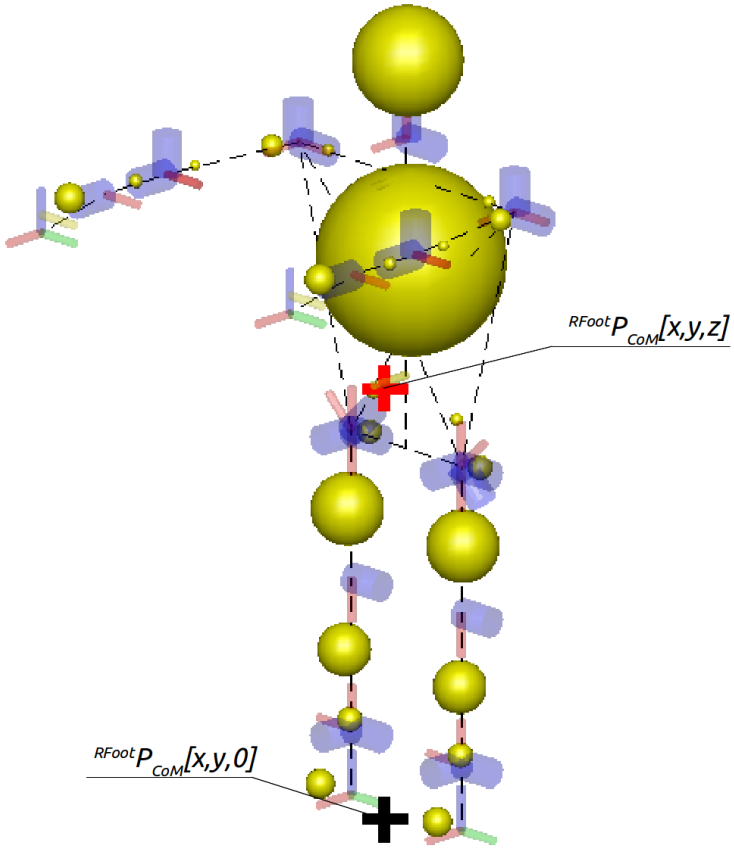


Figure 5.10: *NAO* robot's mass distribution.

It's desired that the *CoM* projection falls onto a specific point within the support area where it is known that the robot keeps balance. Among all points within the convex hull of the right footprint, the point immediately under the right ankle was chosen. That's so as to minimize the torque needed by the ankle to support the whole body's weight.

Using the same *Matlab* script which plots the images, the *DGM* and *DKM* were calculated symbolically and pasted directly into the *C++* program so they can be quickly accessed. Implementing the *DKM* for the end-effectors directly in *C++* would be a straight forward task using the explicit form of the Jacoban matrix. On the other hand, implementing the Jacobians for the *CoM* wouldn't be as simple. Thus, for uniformity, all Jacobians were calculated by derivation with *Matlab*. It is possible that in a future work all Jacobians can be implemented on *C++*.

5.2.1.3 Model accuracy

The robot's software contains methods which return the pose of end-effectors and position of the *CoM* in task space. There's the option of having these values either returned by the actuators themselves or by sensors. The values given by the actuators seem to follow a geometric model according to the joint values set, while those returned by sensors should represent values closer to reality, as they are not aware of the commands which were sent to the actuators.

Since calibration is not one of the goals of this research, divergences between actuator and sensor values will not be addressed. It is important to keep in mind that they exist though, and inaccuracies in the geometric description can be a big source of errors, possibly causing loss of balance. What is being verified here is whether the model implemented corresponds to the robot's geometry and no offsets were ignored or angles misplaced. To this end, the task space poses returned by the model above will be compared to those returned by *NAOqi*'s actuators. Since this value corresponds to the joint angle inputs, the values should be the same.

The method *getPosition* was used to get the position of the end-effectors (*LLeg*: *LFoot* (frame 14); *LArm*: *LHand* (frame 26); *RArm*: *RHand* (frame 20). Frame numbers refer to Figure 5.9). The method *getCOM* was used to return the *3D* coordinates of the robot's center of mass. All these values come expressed with respect to the torso frame, so these values are transformed to the *RFoot* frame making use of the inverse transform returned with the method *getTransform* for the right leg.

The robot starts from a pose balanced on the right foot. Then it performs 3 motions in order: bend *LKnee* (green), stretch *LElbow* (blue), stretch *RElbow* (magenta). Several points of this trajectory are taken as shown on Figure 5.11.

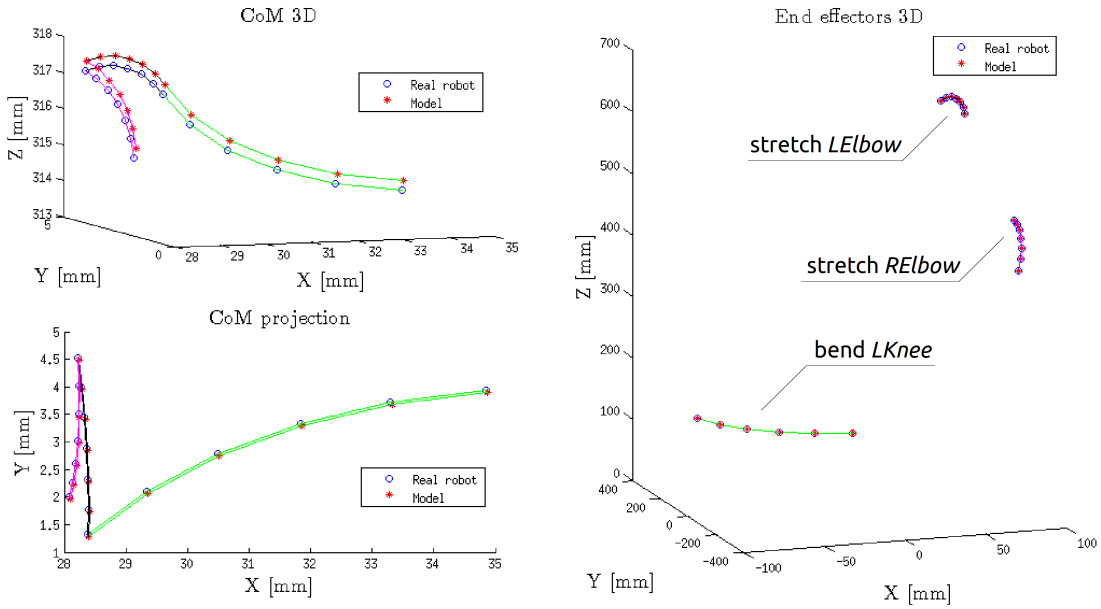
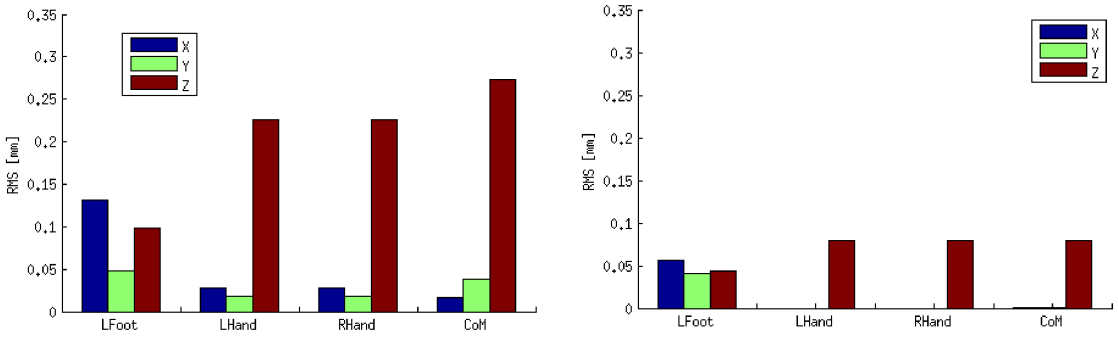


Figure 5.11: Comparison of the developed model and the values returned by the *API* for the V3.3/V4.0 robot.

For both robots, the captured points didn't coincide exactly with the model. Root mean square (*RMS*) errors are shown on Figure 5.12. For the *V3.3/V4.0* robot, errors as high as 0.3 mm were observed, while for the *V4.0* robot the highest error was of 0.08 mm. For both robots, the highest errors were for the *Z* axis.

Recalling that the same end-effector model was used for both robots and that the same



(a) V3.3/V4.0 (b) V4.0

Figure 5.12: Root mean square error between the developed model and the robot's *API* for both robots.

movements were performed on both experiments, it is curious that the robot's *API* returns different values for each robot concerning *LFoot*, *LHand* and *RHand*. It would be interesting to further check the sources of these errors and to use external sensors to calibrate the robot. As of now, these errors are being considered acceptable.

5.2.2 Human motion

Using the *Kinect* sensor, the *NiTE* middleware returns a skeleton with 15 human joints as shown on Figure 5.13. The data is quite noisy and highly affected by lighting conditions. In particular, it was observed that the joint orientation is more prone to large errors than its position. Thus, it was decided that the orientation information wouldn't be taken into account.

The *ARTtrack* system, on the other hand, returns 20 joint frames, including 2 toes, 2 wrists and a point between the hips. These points are seen on Figure 5.14. The returned orientation for all frames is also very reliable. However, since the same implementation is being used for all systems, the *ARTtrack* skeleton will be treated the same way as the *Kinect*'s, the extra frames and orientations ignored. The greatest difference is in the quality of the data, since the marker-based system is less noisy, doesn't have sudden limb displacements and allows for a wider range of motions.

5.2.2.1 Scaling task space

In the previous chapter, it was mentioned that imitation based on inverse kinematics is commonly used when the robot must reach the same point in task space as the human being. In our case however, the robot used is much shorter than the human and thus wouldn't be able to reach most of the human actor's task space. Therefore, for the purpose of this research, the task space is not treated as absolute but as relative to the imitator's size.

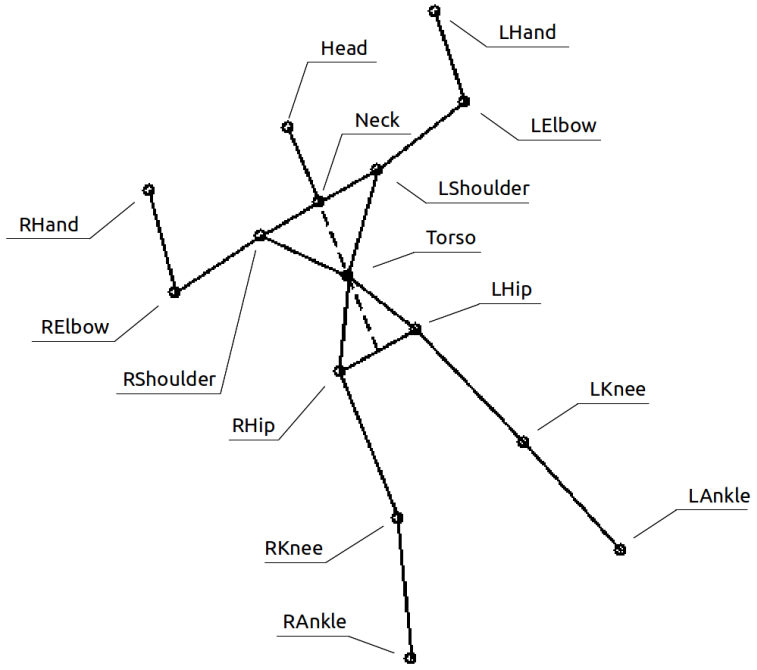


Figure 5.13: Human joints captured with the *Kinect* sensor.

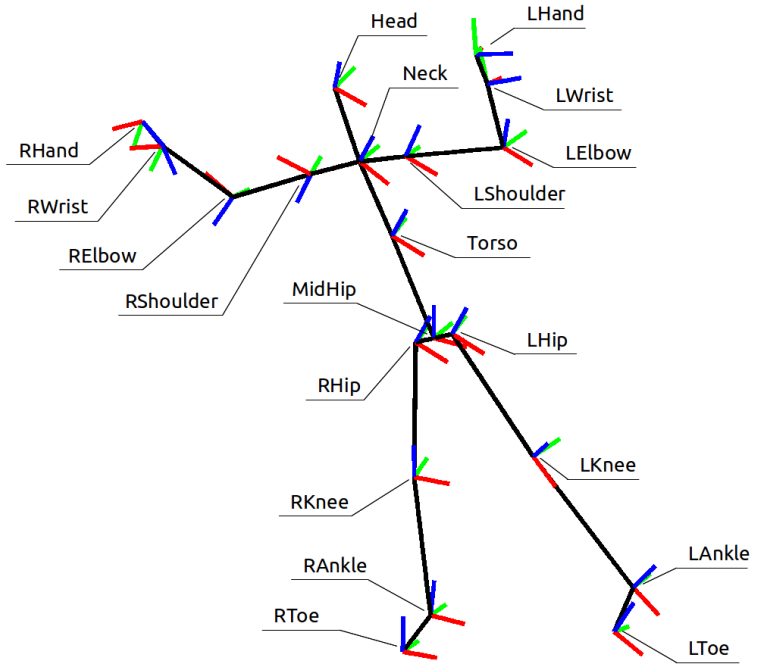


Figure 5.14: Human joints captured with the *ARTtrack* system.

For both capture systems, the data comes expressed with respect to an absolute frame in space. The first step is thus to express all the data with respect to the right foot with the orientation of our model. The imitation starts at the *psi pose*, where the human is standing with both hands up as the letter Ψ. During this pose, the *RFoot* frame is defined with the Z axis pointing upwards parallel to the legs, the Y axis towards the left foot and the origin on *RFoot*. This is done once in the beginning and this transform is used throughout the imitation, so if the human moves *RFoot*, the imitation is spoiled.

In a previous work [51], the use of a scaling factor for each limb based on the ratio between the lengths of the kinematic chain of actor and imitator was proposed. This factor was then multiplied by the captured human data to obtain the desired positions for the robot. Since each limb was a simple serial chain, a proportional scaling factor could be applied satisfactorily.

However, in the present work, the balance is being taken into account and thus the robot has to be modelled as a whole, which gives rise to s tree-structure. In this case, keeping the direction of the captured human data means that the robot's end-effectors will fall on the line which connects the tracked points to the origin on the right foot. This might result in very different body configurations as shown on Figure 5.15.

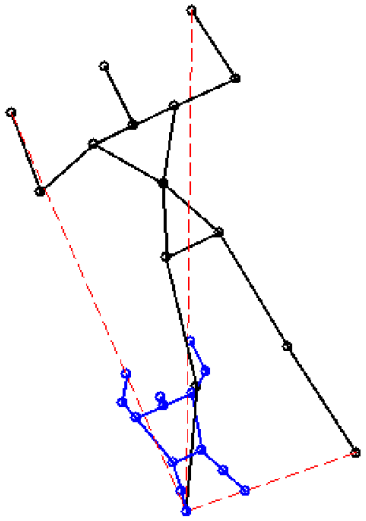


Figure 5.15: Scaling human motion with a proportional scaling factor results in different poses.

In this work, a new scaling method is proposed which takes into account not only the lengths of each of the segments but also each of their directions. This way, the captured human motion is scaled segment by segment to the lengths of the robot l_ℓ^{robot} while keeping the direction of that segment. The iterative process starts from the right ankle and moves upwards towards each of the end-effectors.

Assuming that segment ℓ is the one which comes after joint ℓ, the scaling is performed in 3 steps:

- Human segment ℓ's direction is taken (vector normalization)
- The free vector is multiplied by the corresponding length on the robot l_ℓ^{robot}
- The scaled segment is placed on the chain after its antecedent $\mathbf{P}_a^{robot}$.

In summary:

$$\mathbf{P}_{\ell}^{robot} = \frac{\mathbf{P}_{\ell}^{human} - \mathbf{P}_{a}^{human}}{\|\mathbf{P}_{\ell}^{human} - \mathbf{P}_{a}^{human}\|} l_{\ell}^{robot} + \mathbf{P}_{a}^{robot} \tag{5.7}$$

A question arises as to how to scale the torso. Since the robot doesn't have a spine, the distances connecting shoulders and hips should remain constant. The shoulders cannot be scaled directly from the hips because of the different torso proportions between the human and the robot. It was chosen to scale a segment which goes from the point between the hips to the point between the shoulders, preserving symmetry. The skeleton scaled by this method is seen on Figure 5.16.

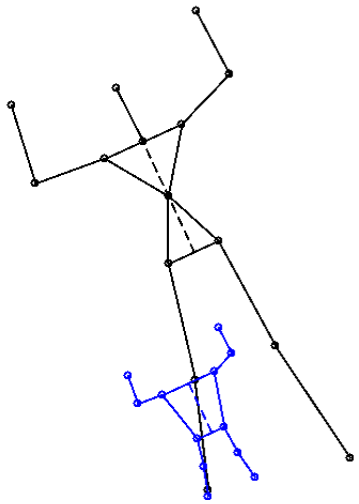


Figure 5.16: Scaling human motion segment by segment.

5.2.2.2 Capturing joint space

In order to keep the robot's joint space close to that of the human's, it is necessary to find the angles which correspond to the robot's *DOF* in the human data.

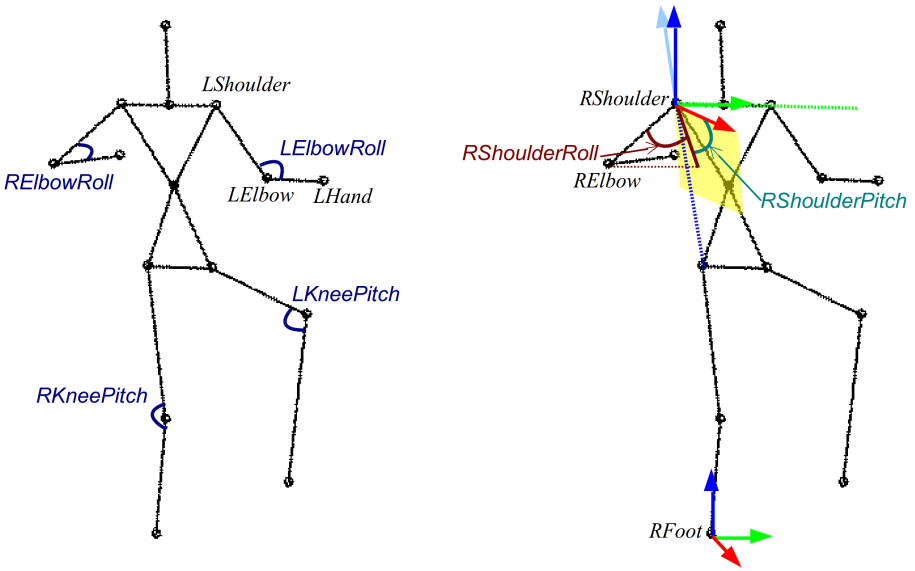
Since we're not relying on orientation information from the motion capture system, as many angles as possible are extracted making use of position data only.

Elbow roll and knee pitch angles can be easily found making use of the dot product between body segments. For example, the angle *LElbowRoll* can be found as the angle between the left upper arm (defined as the segment from $\mathbf{P}_{LElbow}$ to $\mathbf{P}_{LShoulder}$) and the forearm (segment from $\mathbf{P}_{LElbow}$ to $\mathbf{P}_{LHand}$, as shown on Fig. 5.17(a). The equation can be seen below. The same is done for the right elbow and the two knees. For the knees, the upper leg goes from the knee to the hip and the lower leg from the knee to the ankle.

$$LElbowRoll = acos \frac{(\mathbf{P}_{LShoulder} - \mathbf{P}_{LElbow}) \cdot (\mathbf{P}_{LHand} - \mathbf{P}_{LElbow})}{\|\mathbf{P}_{LShoulder} - \mathbf{P}_{LElbow}\| \|\mathbf{P}_{LHand} - \mathbf{P}_{LElbow}\|} \tag{5.8}$$

For each shoulder and hip, 2 *DOF* corresponding to *NAO*'s joints are found. To find these angles it is necessary to find the projection of a segment onto a specific plane. For example, for the right shoulder, the *RShoulderRoll* angle is given as the angle between the upper arm ($\mathbf{P}_{RElbow}$ to $\mathbf{P}_{RShoulder}$) and the sagittal plane containing the right shoulder and the right hip. Refer to Fig. 5.17(b). The *RShoulderPitch* angle is the one from the projection to the axis within the plane which points forward. The plane is defined with the help of a transformation matrix ${}^{RShoulder}\mathbf{T}_{RFoot}$ centred at the right shoulder instead of the right foot. The steps to find 2 *DOF* joints from *RShoulder* are:

- Find ${}^{RShoulder}\mathbf{T}_{RFoot}$
- Express $\mathbf{P}_{RElbow}$ in the frame centered at $\mathbf{P}_{RShoulder}$
- Project $\mathbf{P}_{RElbow}$ on $X-Z$ (sagittal) plane
- $RshoulderRoll$ is the angle between ${}^{RShoulder}\mathbf{P}_{RElbow}$ and the $X-Z$ projection
- $RShoulderPitch$ is the angle between the $X-Z$ projection and the X axis



(a) Elbows and knees (b) Shoulders and hips

Figure 5.17: Extracting human angles from joint positions.

When using the marker-based motion capture system, the orientation of all joints and segments are available. Moreover, more segments are being tracked. This makes it simpler to obtain human joint angles which correspond to the robot's. Also, more angles are obtainable, as the ankle, wrist and head orientations can be found. However, for consistency between the two systems, in this research the same angles were extracted from both systems in the same way. It is very interesting for future work to extract more angles from the marker-based system.

Note that the scaling process has been formulated for the *NAO* robot, but it can be adapted to other robots which have, for example, a spine, toes, *etc.*

5.2.3 Program

The imitation is performed by the combination of two programs. One extracts the desired aspects of the human data, scaling the task space and grabbing the desired joint values. The other one receives this information to calculate the inverse kinematics and send the results to the robot.

5.2.3.1 Motion capture

For the *Kinect* sensor, the motion capture program works either online or offline. For online imitation, a *C++* program communicates directly with the *IK* program via a shared memory segment. When the imitation is offline, the same program can be used to save the needed data on a *CSV* (*comma-separated values*) file for each time step. This program works in a $60Hz$ frequency, but note that during online imitation not all samples are used.

It was not possible to access the *SRTtrack* system online, so only offline data could be used. The same *CSV* file as that of the *Kinect* is generated, but for this system, a *Matlab* script is used to generate the file. This data is also acquired in $60Hz$.

Online or offline, marker-based or markerless, all motion capture programs have the same flow:

1. Receive data from sensor expressed w.r.t. an absolute frame
2. Find a transform from the absolute frame to *RFoot*
3. Express all joints w.r.t. the *RFoot* frame
4. Scale to the robot's size according to Section 5.2.2.1
5. Extract 12 joint angles according to Section 5.2.2.2
6. Send 3 end-effector positions $[3 \times 3 = 9]$ and 12 angles to the robot control program or a *CSV* file (21 values)

5.2.3.2 Robot control

This program controls the robot based on the data which is received from the motion capture system. A general flow chart of the program is shown on Fig. 5.18. Each element is explained below in some detail, as they appear on the flow chart. The *IK* solution is foung numerically [21] making use of the differential model. The difference in task vector $\Delta\mathbf{X}$ is diminished little by little adapting the joint configuration $\mathbf{q}^c$.

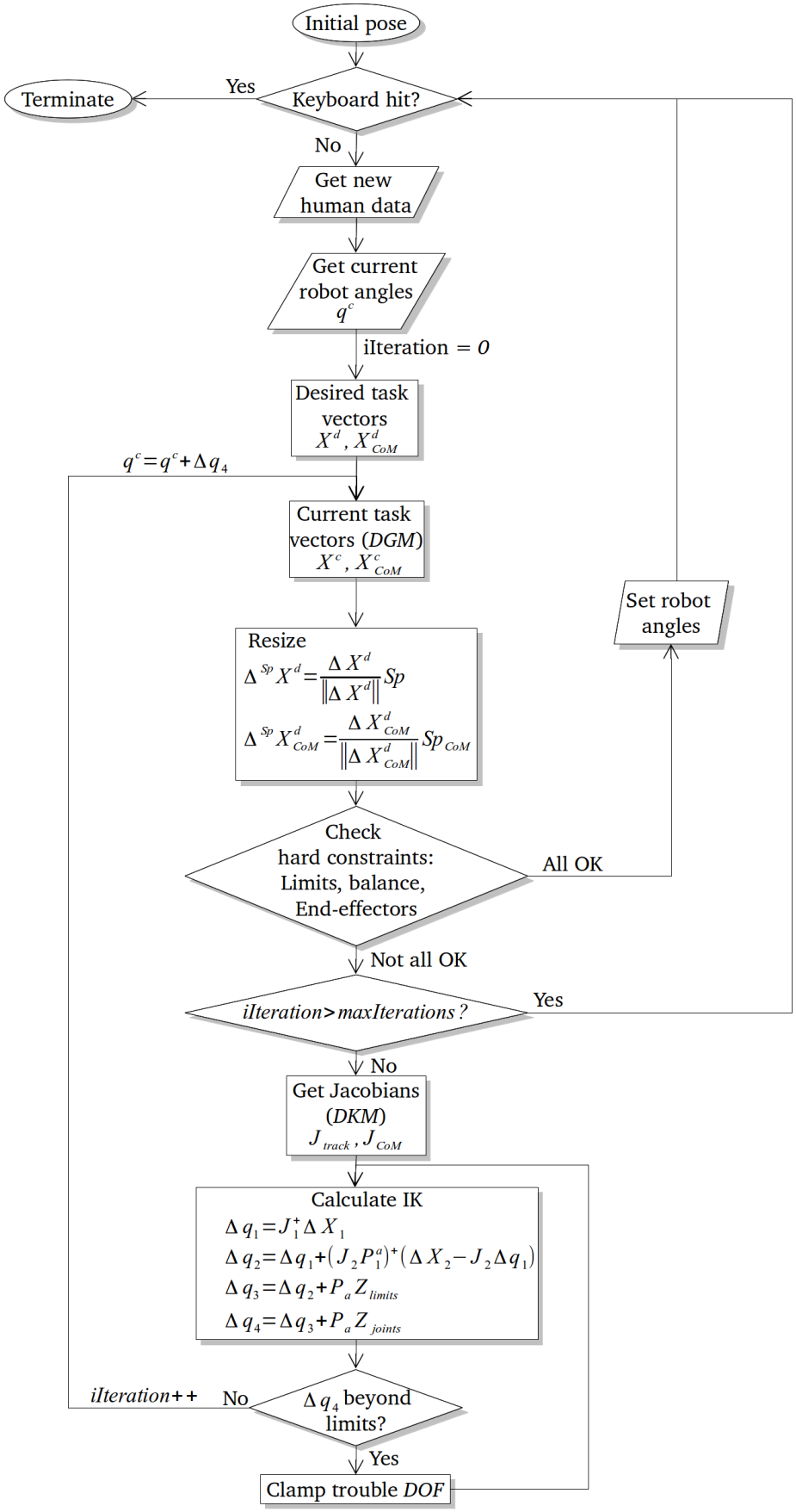


Figure 5.18: Single support *IK* program flow chart.

Initial pose

Fix robot angles in a hand-picked pose which is balanced on *RFoot*. *NAO* says "I'm ready" and someone must manually place him standing on his right foot. The first human data will be sent when the human makes a *psi pose* in single support, similar to the robot's, but this decision is made by the *Kinect* program.

Terminate program

If any keyboard key is hit, the program is properly terminated. Note that ending the program with *Ctrl+C* doesn't save some needed data correctly and leaves the robot stiff.

Get new human data

If online, new data is copied from the memory segment shared with the *Kinect* program. If offline, a new line is read from a *CSV* file. The data received corresponds to the 21 values mentioned in section 5.2.3.1. This data is already scaled and ready to be used by the *IK*.

Get current robot angles

The current robot configuration $\mathbf{q}^c$ is got with the method *getAngles*. Normally, it should be close to the pose previously sent to the robot.

Begin solving

At this point, the program starts to solve for the newest data. The solution might take a few iterations to be found. The iteration counter is set to 0. The human data is distributed into the desired end-effector positions $\mathbf{X}^d$ and desired joint angles $\mathbf{q}^d$.

Current task vectors (*DGM*)

Use the *DGM* (previously calculated symbolically with *Matlab*) to obtain the current task vectors in function of the current joint angles ($\mathbf{X}^c = f(\mathbf{q}^c)$ and $\mathbf{X}^c_{CoM} = f(\mathbf{q}^c)$).

Resize task vectors

In case the difference between current and desired task vectors is larger than their thresholds ($Sp = 10mm$ and $Sp_{CoM} = 10mm$ are being used), the desired vectors are scaled down. The resized goal is referred to as ${}^{Sp}\mathbf{X}_j^d$, as shown on Figure 5.19.Recall that the validity of the pseudo-inverse is only for small task space differences. Moreover, it is important to insure that every step in the robot's trajectory complies with the hard constraints. In case a large solution were found at once, there are higher chances that some points in the trajectory won't comply with balance constraints, for example.

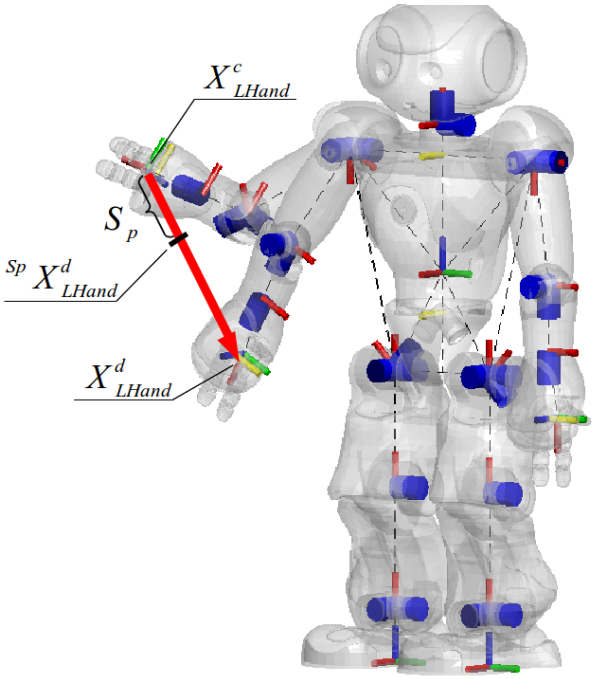


Figure 5.19: The desired end-effector positions are resized to a maximum distance Sp. Here, the example for the left hand.

Check hard constraints

The results are only sent to the robot if all hard constraints have been fulfilled, these being joint limits, balance and end-effector tracking. Each of these constraints is checked separately. Joint clamping keeps the result within limits, but this check is necessary during experiments when the clamping is turned off. Balance and tracking constraints are considered fulfilled when $\Delta\mathbf{X}_{tracking} < max\_Error\_X$ and $\Delta\mathbf{X}_{CoM} < max\_Error\_X\_CoM$. Both errors have been set to $0.2mm$. Note that the goal here is ${}^{Sp}\mathbf{X}^d$, the scaled goal, so if the final goal is too large, it won't be reached.

Set robot angles

If all hard constraints have been satisfied, the angles calculated are sent to the robot. If more than one iteration was needed to solve the IK, the steps are sent one at a time. The

function of *NAOqi* used to send the joint values is *angleInterpolationWithSpeed*. This was chosen for it is a blocking call, and the program only moves on when the goal has been reached. With this function, the speed of motion can be chosen. The default has been set to 10% the maximum speed.

Check for *maxIterations*

Several iterations might be needed for the *IK* to reach the final goal, even if the final goal has been scaled down. However, since this is a numerical approach, it is not assured that a solution will be found. In fact, after some iterations, if the hard constraints are not reachable or compatible, the pseudo-inverse won't come to any useful results. Thus, a maximum number of iterations is set beforehand, so if the *IK* is taking too many iterations to be solved, the current goal can be abandoned and new human data should be taken. It was observed that good results are usually obtained with less than 6 iterations, so a safe limit was set at $maxIterations = 20$.

Task specification

If the current configuration $\mathbf{q}^c$ doesn't comply with all constraints and the maximum number of iterations hasn't been reached, the task-based *IK* is calculated. First, the *DKM* is calculated for the current joint values using symbolic matrices previously got with *Matlab*. Then the task specification is calculated with 4 tasks: balance, end-effector tracking, joint tracking and avoiding joint limits. Most matrices dealt with here have more columns than rows, so the right pseudo-inverse is being used. Among the equality tasks, balance has a higher priority than tracking, so:

$$\mathbf{J}_1 = \mathbf{J}_{CoM} \tag{5.9a}$$

$$\Delta\mathbf{X}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \mathbf{X}^c_{CoM} \tag{5.9b}$$

$$\mathbf{J}_2 = \mathbf{J}_{track} \tag{5.9c}$$

$$\Delta\mathbf{X}_2 = \mathbf{X}^d_{track} - \mathbf{X}^c_{track} \tag{5.9d}$$

Clamping joint limits

The final result of the *IK* including all 4 tasks is $\Delta\mathbf{q}_4$. Recall that this vector contains the values for 23 *DOF*. If some of these *DOF* are not within their limits, they are fixed to their limit values and the corresponding Jacobian columns are zeroed as explained on Section 4.3.1.

5.3 Results

The previously described algorithm was tested online and offline for a wide range of motions. Some results will be analysed in this section.

5.3.1 End-effector tracking

While avoiding joint limits and keeping balance are the most important tasks to assure imitation is successful, the quality of the imitation is measured by how similar the robot's motion is to that of the human actor. The similarity of motion can be measured by how closely the end-effectors and joint values are being tracked. Since we're talking about online imitation, the tracking has to be close not only in value but also in time.

Since the imitation performed involves static balance, the human must move quasi-statically so the robot doesn't gain too much momentum. In this sense, it is easier to perform online imitation than offline, because when registering motion offline, the human doesn't have a feedback on how slowly the robot is moving and is not able to slow down accordingly, which will result in very different motions between the human and the robot.

An example for a simple motion is shown in Figure 5.20. This imitation was performed online using 10% of the robot's maximum speed. The norm of the desired positions and the obtained positions for each end-effector during this motion are shown on Figure 5.21. On the left, the limb trajectories in Cartesian space are plotted for the scaled human and the robot. The skeleton shown corresponds to the initial pose for both. Note that the human data from the *Kinect* sensor is quite noisy.

When the human moves too quickly, such as the hand movements from $7s$ to $17s$, the robot is not able to catch up because of the limitation imposed on its speed. As previously explained, the speed was limited in order to avoid high accelerations, since this is a kinematic approach. However, if the human is able to move slowly, the algorithm is able to track closely in time, as can be seen for the ankle motion from $23s$ onwards.

Note that due to the choice of dividing the goal positions into smaller steps, the robot is constantly moving towards the new goal and forgetting the previous. This means that the amplitude of motion is not as large as that of the human's, for example, the hand motion at $15s$[1] wasn't tracked because the human changed the direction before the robot could catch up. It was observed that for online imitation, this results in higher quality of motion with less delay, as the robot quickly adapts to new goals.

It is very common that the *Kinect* sensor loses a limb for some time steps, causing a large discontinuity in the motion. Such a case is illustrated in Figure 5.22, where the

[1]The quick human motion at $15s$ corresponds to a moment when the human lost balance and had to move quickly to recover balance.

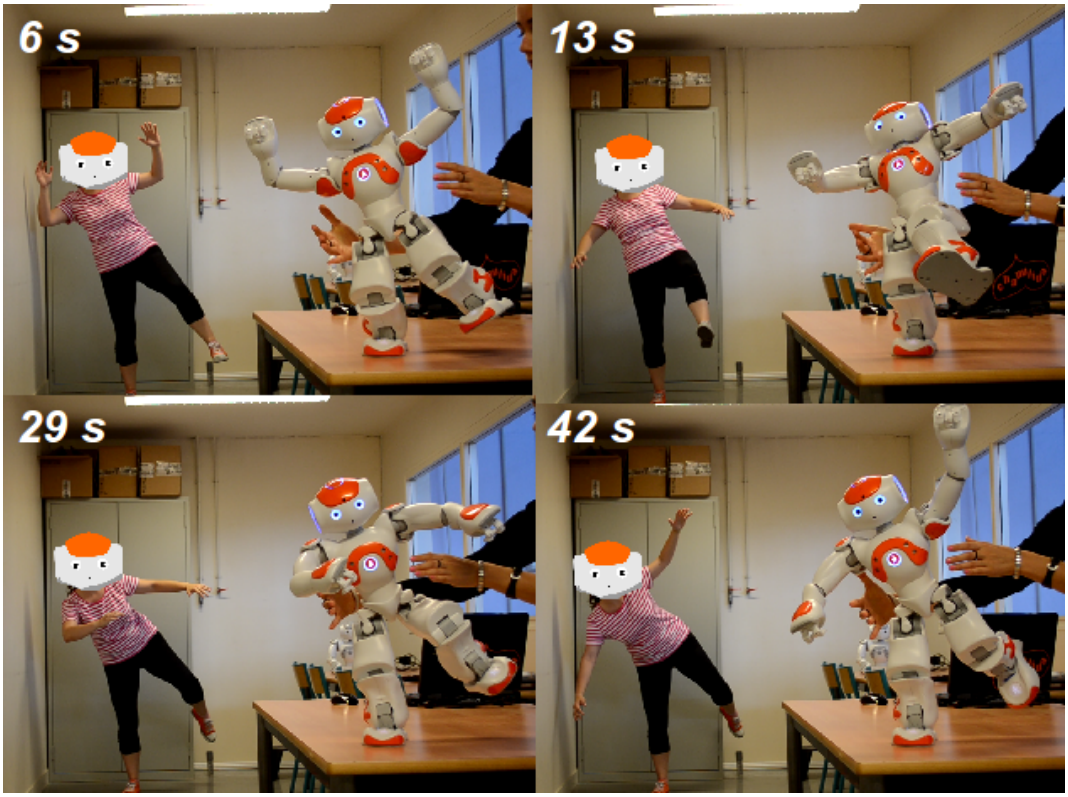


Figure 5.20: Four moments during single support imitation.

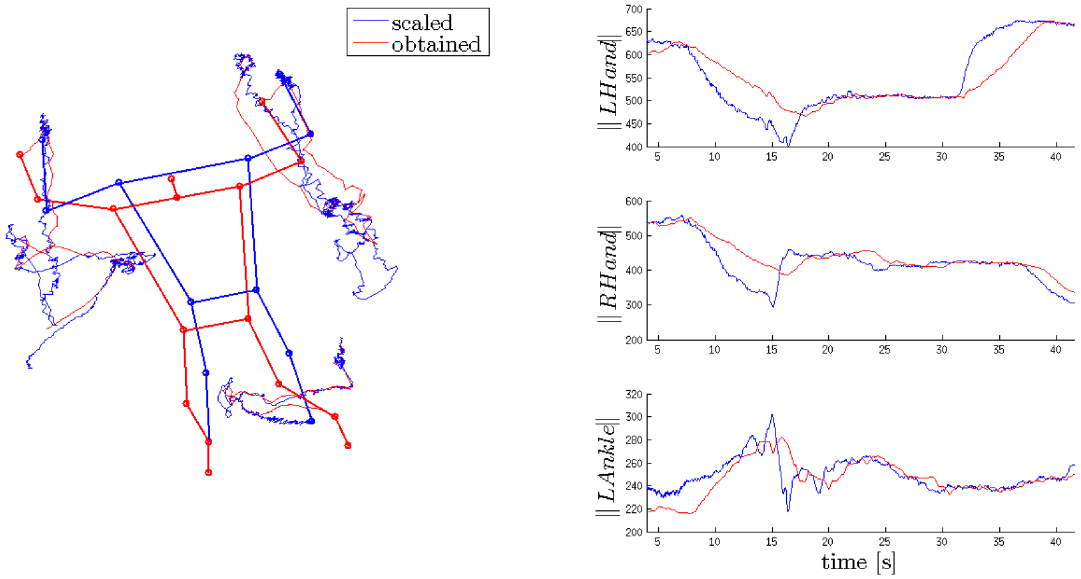


Figure 5.21: End-effector tracking results corresponding to the motion on Figure 5.20.

Kinect misplaced the ankle from 30 s to 33 s close to the support foot, when the actor actually had the left ankle raised. If the motion weren't divided into smaller steps, the robot would follow the misplaced limb all the way and move very differently from the human. Thus, dividing the motion also copes with some of the sensor's limitations.

It can thus be concluded that the end-effectors were successfully tracked in time and space given the speed limitations of a kinematic approach and the quality of the data

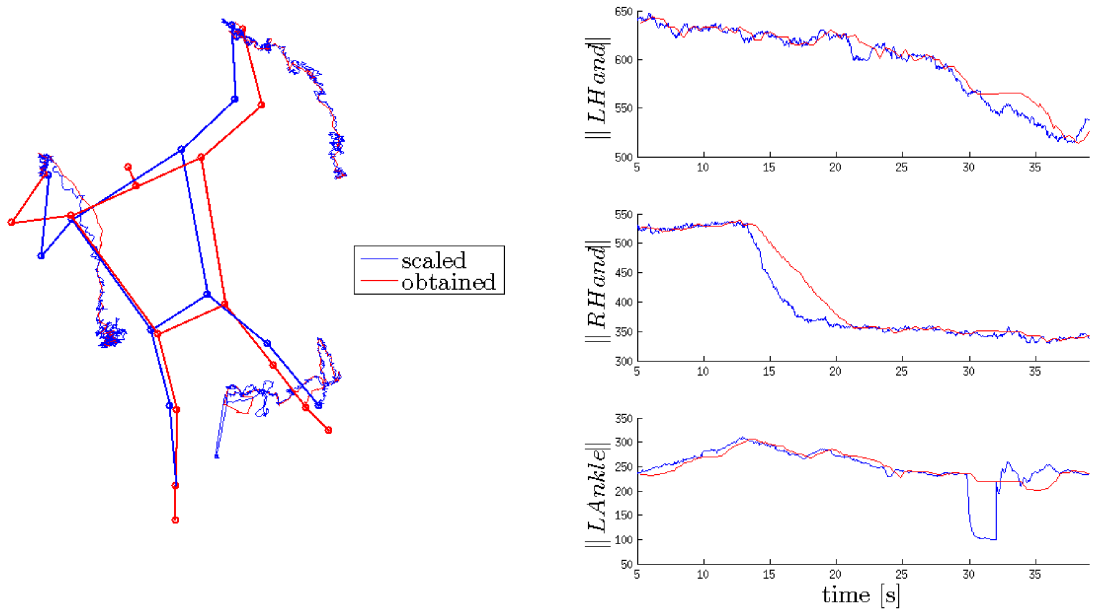


Figure 5.22: End-effector tracking when the *Kinect* misplaces a limb.

obtained from the motion capture system.

5.3.2 Balance

For the motion of Figure 5.20, the evolution in time and space of the *CoM* projection on the ground is shown on Figure 5.23. The *CoM* projection is estimated using the developed model and the robot configuration after each time step obtained with *getAngles*.

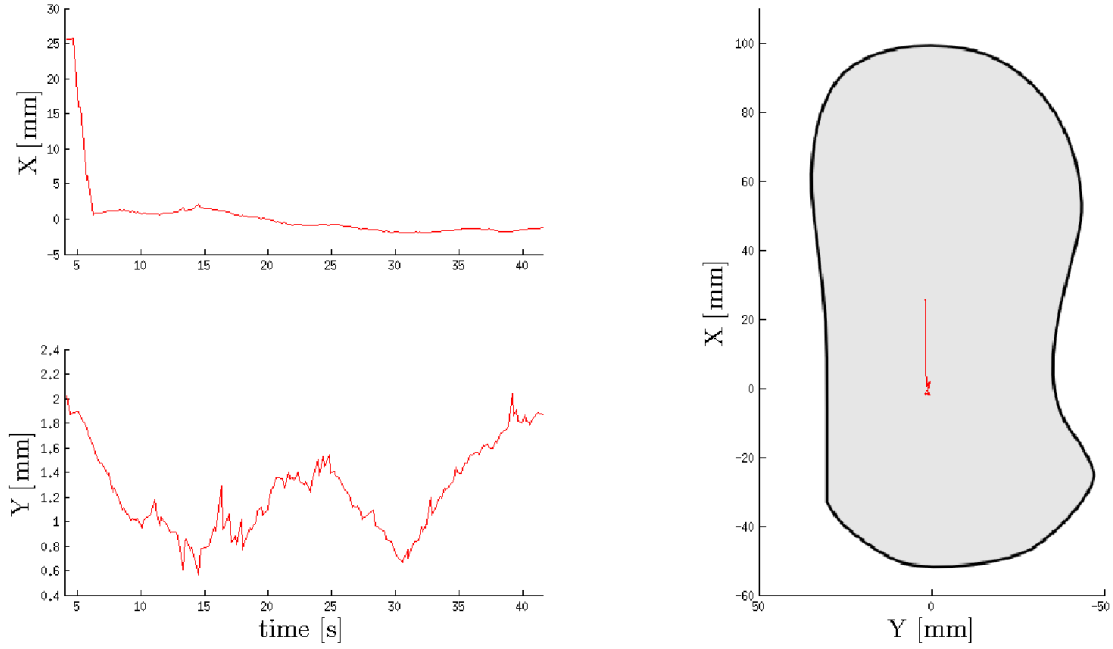


Figure 5.23: *CoM* projection on the floor throughout motion imitation.

The hand-picked initial position had the projection about $25mm$ in front of the ankle projection ($x = 0, y = 0$), but when the imitation begins, the robot successfully brings the CoM below the ankle and keeps it in that vicinity throughout the imitation. The error allowed for the CoM projection was of $0.2mm$ for each coordinate, but errors as high as $2mm$ were observed. This shouldn't challenge the robot's balance however, since the projection stays well within the support area.

Although the robot was able to stay balanced throughout most of the experiments, occasional falls were also observed. These falls are attributed to sudden jerks, which can occur even at 10% the maximum speed; or to mechanical limitations, such as motors getting hot and suddenly losing stiffness. Errors due to the lack of calibration are also a possible cause of falls. Moreover, it is important to notice that during single support imitation, issues such as interpenetration and contact between the swing foot and the floor weren't addressed. Therefore, it is still necessary to insure security because in rare occasions the robot might fall.

5.3.3 Joint space

Two of the four tasks are directly related to the joint space: joint limit avoidance and tracking the human joints.

5.3.3.1 Joint limits

The joint limits were avoided by a combination of an optimization task and a clamping loop (Section 4.3.1). Figure 5.24 shows how even though some human joints went beyond the robot's limits, the robot was able to imitate while staying within its limits.

Avoiding the joint limits was treated as a hard constraint: if the IK solution doesn't fit into one of the joints' limits, the results are not sent to the robot at all. After trial and error, the weight κ_{limits} was set to -0.02. This value will be more deeply analysed in the next chapter, during double support. The necessity of using both clamping and optimization will also be investigated then.

5.3.3.2 Joint tracking

Finally, the task with the lowest priority is that of tracking the human angles in joint space. This task is added because in case all the previous tasks are easily satisfied, it would be nice to have the robot perform the task in a manner as close as possible to the human. This is the only out of the four tasks which is being treated as a soft constraint.

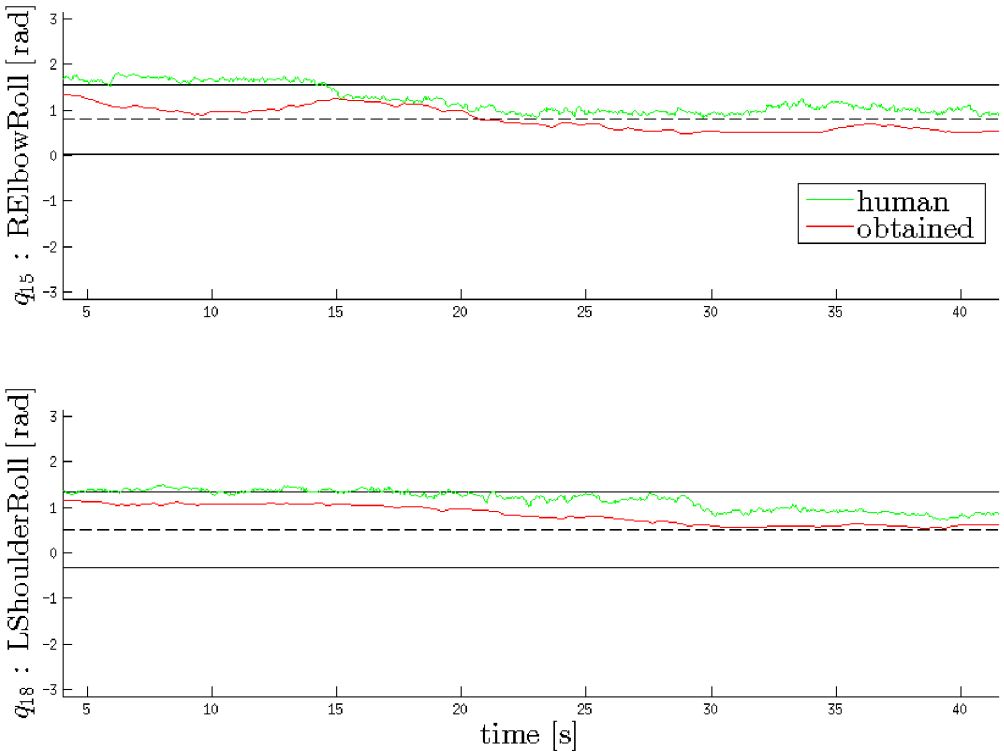


Figure 5.24: Human joint angles beyond robot limits.

As seen in Figure 5.24 above, at times, robot joints move quite differently from the human ones, which is to be expected of the lowest priority task. However, whenever is possible, the joints follow the human quite closely, as seen on Figure 5.25. This task will also be analysed in more depth on the following chapter.

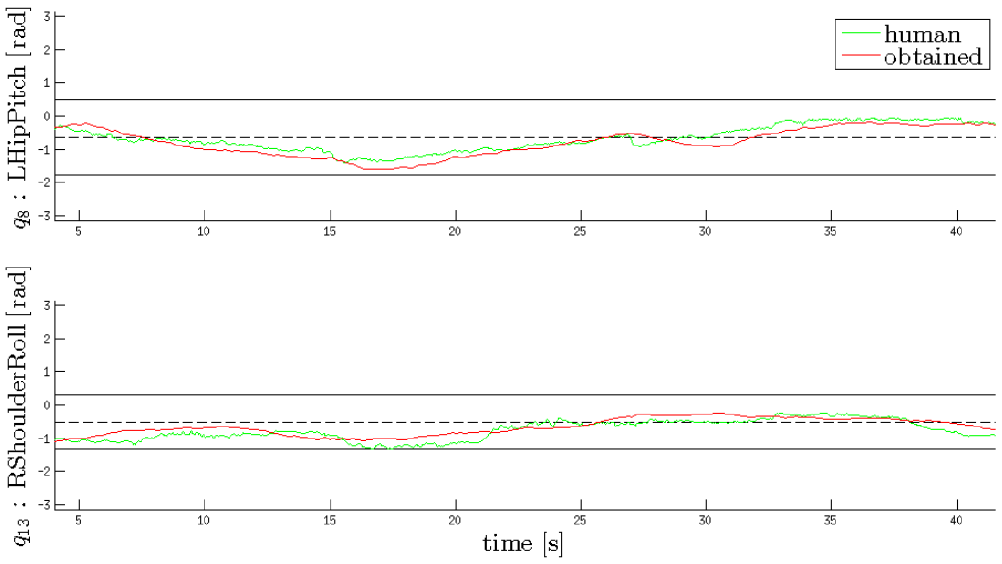


Figure 5.25: Human joint angles tracking.

5.3.4 Motor speed

Human beings are capable of moving very fast while keeping balance and good control of their motion. Reproducing human motions with the same quality and speed is a very difficult task. In this work, only quasi-static motion is being imitated, which is just an approximation. In reality, the human actors are moving slowly, but how slow is slow enough? It is interesting to push the speed barrier to as fast as the kinematic simplifications can handle.

NAO's *API* allows the choice of what fraction of the motor's maximum speed will be used. In order to test the effects of increasing motor speed, a motion was recorded online with 10% of the maximum speed and re-imitated offline with speeds from 10% to 100%. The motion analysed consists of lowering both arms and lifting the left ankle at the same time. The end-effector tracking for the motion in question for all speeds is shown on Figure 5.26. Three moments during the motion are shown on the right. Time zero represents the moment the imitation program begins running. Only the 300 samples corresponding to the motion in question are plotted.

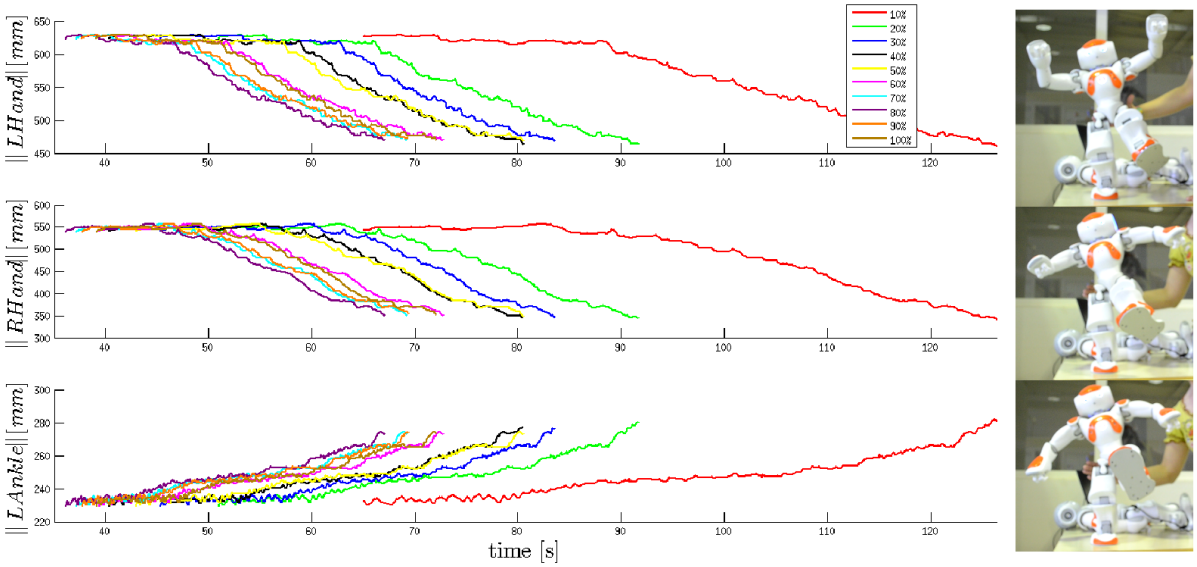


Figure 5.26: Same motion imitated offline with different speeds. On the left, end-effector tracking; on the right, three moments during the motion.

It was observed that in general, increasing the speed reflects in decreasing imitation time. However that is not always the case. For example, 80% speed took 30.9s to go through this motion (300 samples) while 100% speed took 33.0s. While the exactly same data was used for all experiments, since the solution is numerical and depends on the feedback from the robot, slightly different motions occur.

For all speeds, the robot jerked as it moved from sample to sample. This is mainly due to the noisy data from the *Kinect*, which doesn't provide a smooth motion. As the speed was increased, the jerking got stronger and the robot more unstable. Even though it did not lose fall this motion for any speed, the faster the speed, the more it wobbled on the

right ankle, sometimes the foot even lost flat contact with the ground.

Perhaps with smoother data, the speed can be increased without generating dangerous dynamic effects. However, for noisy data, it would be dangerous to have the robot shaking at high speeds. Besides, all the shaking looks bad. Therefore, it was decided to stick to 10% speed in further experiments.

5.3.5 Time constraint

During online imitation, the time taken to go from human motion until robot motion is of great importance for the quality of imitation. No time constraints were imposed on the program however, so the flow on Figure 5.18 runs until a solution is found or the maximum number of iterations is reached, taking a different time each loop. The period difference among time steps is mainly due to the different time required for the robot to move and the different number of iterations needed to solve each configuration.

Since there's no fixed frequency for the program, we'll be analysing the average time for each block in the program. The motion here analysed is the one on Figure 5.20. The imitation was performed at 10% the maximum speed, online, with the *Kinect* sensor.

The average time taken from the reception of one new human data to the next was of $410ms$ $(2.4Hz)$, with a minimum of $200ms$ and a maximum of $1041ms$. It is clear that the pose required from the robot greatly influences the resolution time.

But what is taking so long, the resolution of the *IK* or the robot motion? The chart on Figure 5.27 shows that, in average, 90% of the imitation time is spent by the robot moving, which takes $365.0ms$ in average. More specifically, this time corresponds to the instructions sent with *angleInterpolationWithSpeed*, one iteration at a time. This time can certainly be improved by increasing the robot speed, but as explained in the previous section, that would endanger balance.

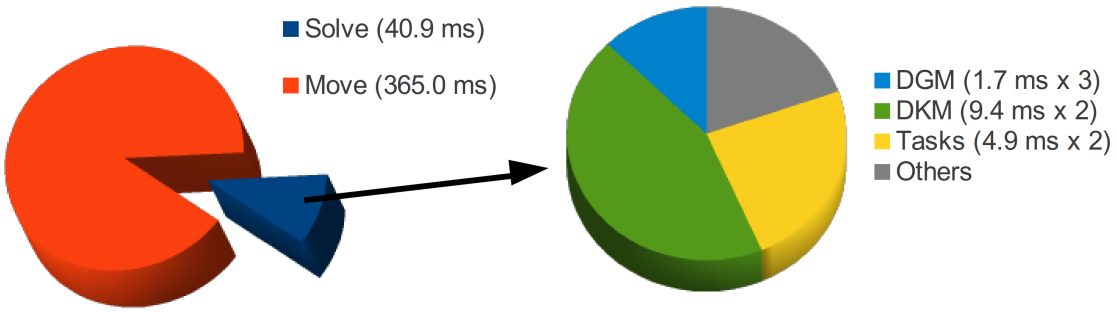


Figure 5.27: Average time slices for various parts of the program.

The *Solve* time given on the chart refers to the time between the moment the human data is received until either all constraints are satisfied or the maximum number of iterations

is reached. This takes 40.9ms in average. Most samples needed exactly 2 iterations to be solved (only 2/91 samples took 3 iterations).

The chart on the right shows what parts within solving take the longest. Each block's average time is multiplied by the average number of iterations. Most of the solving time is taken by the *DKM*. This corresponds to accessing the symbolic Jacobians for the 3 end-effectors and for the *CoM* projection. Next, the most time is taken by the task solving itself, which includes the computation of several matrix multiplications (including those to compute the pseudo-inverses), and the clamping loop.

5.4 Conclusions

In this chapter, task-based inverse kinematics were used to control a humanoid robot balanced on one foot using human motion online. A method of scaling the human motion segment by segment was introduced and its efficiency in reproducing whole-body poses verified.

The quality of motion was analysed by looking at the hard constraints: end-effector tracking, keeping balance, avoiding joint limits; and soft-constraints: joint tracking and imitation time. The results show that the robot successfully tracked the human performer's end-effectors (hands and left foot) while keeping the projection of its center of mass under the right foot's ankle. The tracking is good even when the sensor misplaces a limb for a few seconds.

It was shown that the clamping loop together with the limit avoidance optimization task are able to keep the joints within their limits and the human joint values are tracked whenever is possible. Is was also explained that for stability reasons, the chosen robot velocity was of 10% its maximum speed. As for the imitation time, it was seen that about 90% of each time step is spent by the robot moving while only 10% are used to solve the *IK*. The average frequency of imitation was of 2.4Hz.

Chapter 6

Imitation including support changes

In the previous chapter it was explained how imitation in constant single support on the right foot was performed. That implementation was very limited however, since it corresponds to only a fraction of all the possible human motions. In this chapter, the human ability of standing on either foot, two feet at the same time and transitioning between these behaviours will be addressed. The method previously used is extended to include double support and support on the left foot, as well as the transitions between them. Figure 6.1 captures several moments during online imitation including support changes for different actors.

Figure 6.1: Example of online imitation including support changes.

6.1 Main differences from single support imitation

Up to now, the robot was continuously on right support (RS). Now, double support (DS) and left support (LS) are included in the imitation.

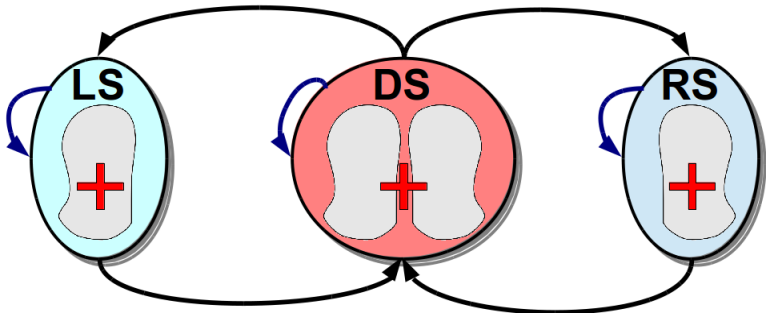


Figure 6.2: Transitions between supports.

On the scheme above, blue arrows represent a continuation of the support from the previous time step, while black arrows represent a change in support, *i.e.* a transition. The red cross represents the desired position for the projection of the CoM in relation to the robot's footprints. Note that it is not allowed for the robot to transition directly from RS to LS and vice-versa.

Unlike before, the goal point for the projection of the CoM changes according to the support. It is therefore very important to make sure the CoM is within the current support polygon while it is moved from one goal to the other in order to ensure balance during transition. Another difference is that now the right foot is not fixed to the floor any longer, so the robot is able to locomote in the world frame. That calls for an absolute frame external to the robot.

6.2 Implementation

The tools used for this experiment were the same as those in the previous chapter. Like before, two programs were developed, one which handles the motion capture and another which controls the robot. Several elements were added both to the robot model and to the human motion scaling in order to manage the new motions.

6.2.1 Robot model

Now that the robot's right foot is no longer fixed to the world frame, the modelling has to be rethought in a way that it takes support changes into account. For instance, when the robot is in left support, the projection of the CoM has to be taken with respect to the left foot's frame.

This could be solved in several ways. For instance, calculating a transform ${}^{LFoot}\mathbf{T}_{RFoot}$ which transforms points expressed on the right foot frame into the left foot frame. Another possibility would be to add 6 *DOF* to the model in order to position and orient the robot in the world frame. Both these solutions proved to be unfit though. The former was too heavy for *Matlab* to calculate symbolically and the latter would over-constrain the 23 *DOF* robot. Face these difficulties, a different approach was taken.

Instead of using the *RFoot* model and transforming points to the *LFoot* frame, a new model based on the *LFoot* was developed. So whenever the robot is in *LS*, the *IK* can be solved with this model, and while in *RS*, the model from the previous chapter can be used. The M-DH parameters for this model and the resulting tree structure are shown on Table 6.1 and Figure 6.3 respectively. From the torso upwards, the model is the same as before. The masses were placed on the new kinematic chain as previously done.

During *RS*, the model based on the right foot (Figure 5.9, Table 5.1) will be used, while the left foot model just described will be used during *LS*. As for *DS*, the right foot model will be used together with constraints that assure that the left foot is flat on the floor as will be explained shortly.

Table 6.1: *NAO*'s M-DH parameters based on the left foot.

ℓ	a_ℓ	σ_ℓ	γ_ℓ	b_ℓ	α_ℓ	d_ℓ	θ_ℓ	r_ℓ
1	0	0	$-\frac{\pi}{2}$	$FootHeight$	$-\frac{\pi}{2}$	0	$-\frac{\pi}{2} - RAnkleRoll$	0
2	1	0			$\frac{\pi}{2}$	0	$-RAnklePitch$	0
3	2	0			$\frac{\pi}{2}$	$TibiaLength$	$-RKneePitch$	0
4	3	0			0	$ThighLength$	$-RHipPitch$	0
5	4	0			$-\frac{\pi}{2}$	0	$-\frac{\pi}{4} - RHipRoll$	0
6	5	0			$-\frac{\pi}{2}$	0	$(L/R)HipYawPitch$	0
7	6	$2(Torso)$	$-\frac{\pi}{2}$	$\sqrt{HipOffsetY^2 + HipOffsetY^2}$	$-\frac{\pi}{4}$	0	0	$HipOffsetZ - HipOffsetY$
8	7	0	0	$-(HipOffsetZ - HipOffsetY)$	$\frac{3\pi}{4}$	0	$\frac{\pi}{2} - (L/R)HipYawPitch$	$\sqrt{HipOffsetY^2 + HipOffsetY^2}$
9	8	0			$\frac{\pi}{2}$	0	$-\frac{\pi}{4} + LHipRoll$	0
10	9	0			$\frac{\pi}{2}$	0	$LHipPitch$	0
11	10	0			0	$-ThighLength$	$LKneePitch$	0
12	11	0			0	$-TibiaLength$	$LAnklePitch$	0
13	12	0			$-\frac{\pi}{2}$	0	$\frac{\pi}{2} + LAnkleRoll$	0
14	13	$2(RFoot)$			$\frac{\pi}{2}$	0	$\frac{\pi}{2}$	$-FootHeight$
15	7	0	0	$ShoulderOffsetZ$	$-\frac{\pi}{2}$	0	$RShoulderPitch$	$-ShoulderOffsetY$
16	15	0			$\frac{\pi}{2}$	0	$\frac{\pi}{2} + RShoulderRoll$	0
17	16	0			$\frac{\pi}{2}$	$-ElbowOffsetY$	$RElbowYaw$	$UpperArmLength$
18	17	0			$-\frac{\pi}{2}$	0	$RElbowRoll$	0
19	18	0			$\frac{\pi}{2}$	0	$RWristYaw$	$LowerArmLength$
20	19	$2(RHand)$	0	$HandOffsetX$	$-\frac{\pi}{2}$	0	$-\frac{\pi}{2}$	$-HandOffsetZ$
21	7	0	0	$ShoulderOffsetZ$	$-\frac{\pi}{2}$	0	$LShoulderPitch$	$ShoulderOffsetY$
22	21	0			$\frac{\pi}{2}$	0	$\frac{\pi}{2} + LShoulderRoll$	0
23	22	0			$\frac{\pi}{2}$	$ElbowOffsetY$	$LElbowYaw$	$UpperArmLength$
24	23	0			$-\frac{\pi}{2}$	0	$LElbowRoll$	0
25	24	0			$\frac{\pi}{2}$	0	$LWristYaw$	$LowerArmLength$
26	25	$2(LHand)$	0	$HandOffsetX$	$-\frac{\pi}{2}$	0	$-\frac{\pi}{2}$	$-HandOffsetZ$
27	7	0			0	0	$HeadYaw$	$NeckOffsetZ$
28	27	0			$-\frac{\pi}{2}$	0	$-\frac{\pi}{2} + HeadPitch$	0

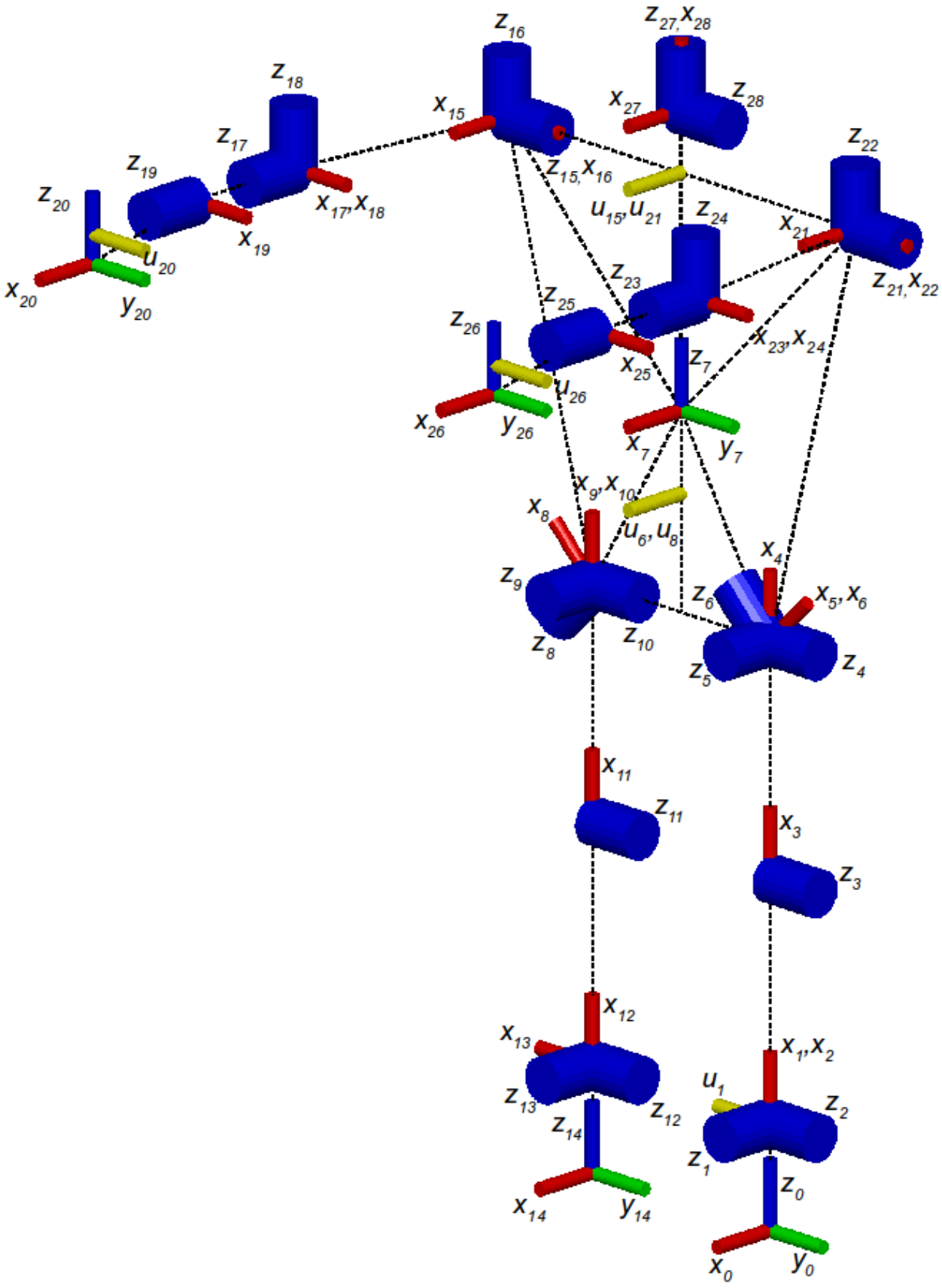


Figure 6.3: *NAO*'s kinematic chain based on the left foot, modelled with the *M-DH* parameters on Table 6.1.

6.2.2 Human motion

For single support imitation, the captured skeleton was always translated to the right foot and then scaled down to the robot's size (Section 5.2.2.1). Here, the scaling including support changes is significantly more complicated. Now there's the need for an absolute frame which is not attached to the robot. Also, there's the need to detect the current support type and transform the frames accordingly. Finally, special care has to be given to the *DS* phase so the scaled motion keeps both feet on the floor. A summary of all frames of reference used is shown on Figure 6.4.

All these transformations occur in the program responsible for the motion capture. As before, for the *Kinect* sensor, the program can run either online or offline. For the *ART-track* system, the data is captured offline, transformed with *Matlab* and sent to the *IK* program via a *CSV* file.

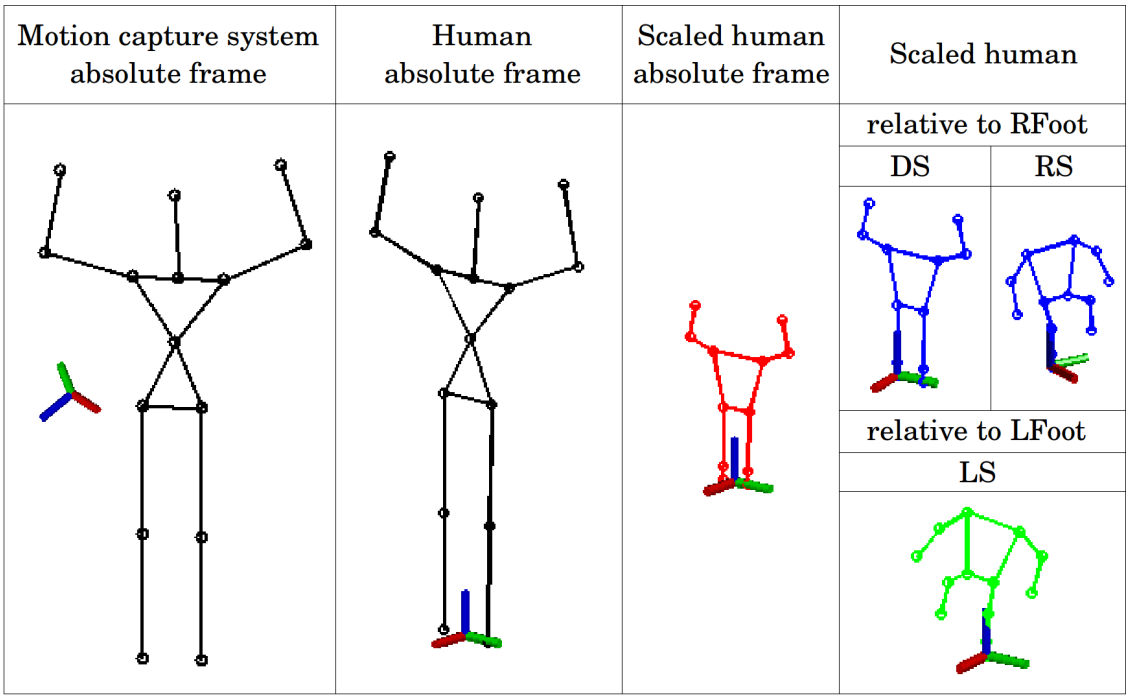


Figure 6.4: Frames of reference for imitation including support changes.

6.2.2.1 Human absolute frame

Each motion capture system has its own frame of reference. For consistency among different systems, the first step is to define our own absolute frame of reference. When only *RS* was imitated, the absolute frame coincided with *RFoot*. Now, the absolute frame is placed on the point in between the two feet. As before, the calibration pose (*psi pose*, with both arms bent about 90°) is used to help define the absolute frame: the frame's origin is placed between the two feet with its Y-axis pointing towards the left foot, and Z-axis parallel to the legs. A transform from the motion capture system's absolute frame to this frame is calculated once in the beginning and used at each time step. Therefore,

unlike in the previous chapter, the human can move around, but the sensor still cannot be moved during imitation.

6.2.2.2 Support detection

Before scaling the data to the robot's size, it is necessary to detect in which kind of support the human is. A foot is considered to be support if its z coordinate is lower than a given threshold. The threshold chosen is quite high ($200mm$) to avoid false positives due to jittering. Even so, depending on lighting conditions, the quality of the *Kinect* data can be so bad that false positives occur. Such a high threshold means that the human actor must lift the leg very high in order to cause a support change in the robot.

6.2.2.3 Scaling in the absolute frame

The data is scaled to the robot's size with respect to an absolute frame placed between the robot's feet. It will be with respect to this absolute frame that the robot will locomote in space. This data will be later transformed to a relative frame (*RFoot* or *LFoot*) only for the *IK* resolution, according to the current support. When scaling the data, important decisions must be taken regarding how to choose where to place the feet.

Continuous Double Support

The imitation always starts after calibration with the *psi pose*, which is in *DS*. From there, the robot remains in *DS* for a while until the actor decides to lift a leg. Due to jittering, the human data received may vary a lot from one time step to the next. However, in double support it is specially not desirable that the robot's legs keep on shaking as it jeopardizes stability. Thus, it was decided that when the robot is continuously in *DS*, both feet's positions are kept the same as in the previous time step. That is, the scaling forces them to stay in place, ignoring the received data for the feet. This comes with one great limitation though: if the human slides the feet on the floor, the robot won't follow.

Since we're tracking ankle frames, these frames will be fixed at *FootHeight* (refer to Figure 5.8). Fixing both ankles means that the legs are forming a closed loop. It would be very complicated to scale the knees and hips in order to fit the robot's geometry while following the human directions as previously done. For offline motion, optimization techniques could be used, but during online imitation, this might take precious time for something that isn't very important (remember that we're only interested in the end-effectors positions and other joints are only scaled together in order to maintain proportions close to the human).

To cope with the closed loop, it was decided to simplify the legs during *DS*. Instead of scaling each body segment separately, only for the legs, the vector from the mid-ankles

to the mid-hips was scaled at once. From the mid-hips onwards however, the previous method was maintained (Section 5.2.2.1). Figure 6.5 shows a human pose and its corresponding scaling. Note that after scaling, no knee joints can be seen, as they were not calculated.

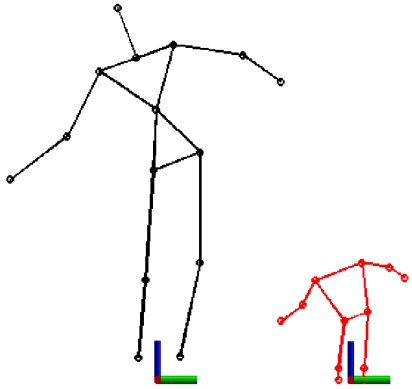
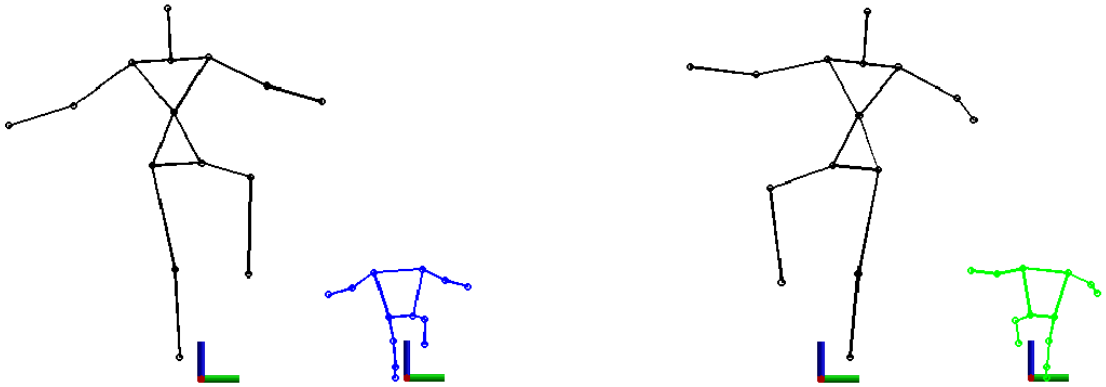


Figure 6.5: Scaling in double support.

Transition into single support and its continuation

When either foot is lifted beyond the threshold height, single support begins. The scaling stops forcing both feet on the floor and instead fixes only the one which remains as support. That is, the support foot and ankle keep their positions from the previous time step. This works both for a continuous single support or for a *DS-SS* transition. From the ankle upwards, the rest of the body is scaled as in the previous chapter, starting from either the right ankle or the left ankle, as seen on Figure 6.6.



(a) Right support (b) Left support

Figure 6.6: Scaling in single support.

There's a large discontinuity when transitioning from *DS* to *SS*, because the foot which was forced to be on the ground is suddenly lifted beyond the threshold. This will be dealt with on the robot control side.

Transition into Double Support

Finally, when transitioning back into *DS*, the swing foot should be fixed to the floor again. Once more, there will be a discontinuity which will be dealt with later. The choice to be made here is where to place the foot - what will be the position on the floor for the foot to track?

Human beings make great use of their dynamics when placing feet on the floor. When walking, for example, we "fall" onto double support because we know that the swing foot will be placed in time to absorb the fall. Since we're dealing with static imitation, this kind of dynamic movement is not taken into account.

The range of positions on the floor where the robot can place the swing foot while in static balance is quite narrow since the *CoM* projection must remain in the support foot. Thus, the human foot position received has to be scaled down to the robot's reach while making sure that the feet are far enough from each other, because *NAO*'s feet are proportionally larger than the human's.

Several approaches were experimented with. Fixing the foot onto the projection of its previous *SS* pose, for example, often results in poses which cannot be reached by the robot (the *IK* doesn't converge to a solution). An approach which worked reasonably well was to scale the human foot position to fit within an area where it is known that *NAO* can reach with a flat foot in static balance. This area was found by trial and error by making the robot trace a rectangle on the floor with flat feet.

Different rectangles were experimented with until one was found where the robot can comfortably reach all edges. The chosen rectangle is shown on Figure 6.7 with respect to *RFoot* ($x_{min} = -100mm, x_{max} = 100mm, y_{min} = 100mm, y_{max} = 150mm$). The rectangle corresponds to the allowed positions for the swing foot origin, which is under the ankle, so the rest of the foot can fall outside this area. Note that the foot orientation w.r.t. the vertical axis is not being controlled, the reason will be explained shortly. Since this area was chosen by trial and error, a more rigorous kinematic analysis of the foot landing area should be done in the future.

The acquired human foot position is first scaled down to the robot's size. If it is within the rectangle, the foot is fixed in that position. If it is outside, it is scaled to the closest edge.

6.2.2.4 Transforming to the support foot frame

In the previous section, the data was scaled with respect to an absolute frame. That will be the space in which the robot will locomote. However, since our kinematic models assume one of the feet to be fixed to the ground, it is necessary to translate the scaled

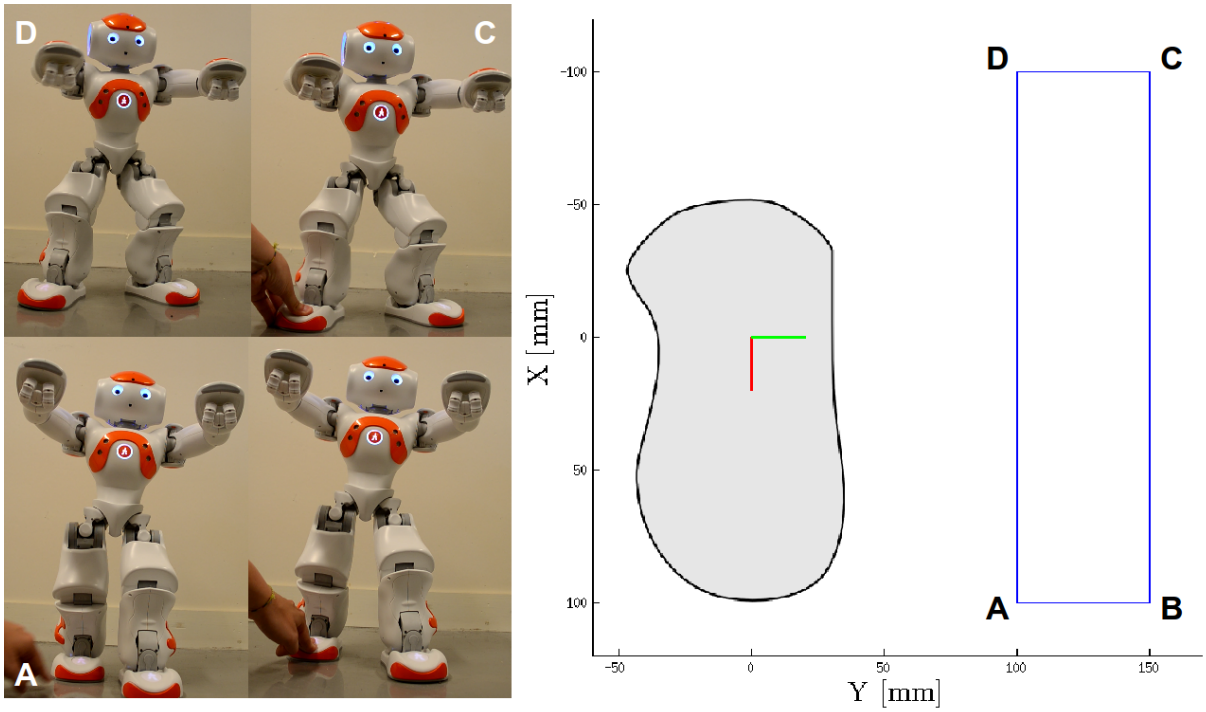


Figure 6.7: Mapping area which can be reached by the ankle of swing foot while flat. Note that the rest of the foot might fall outside the area. Here, depicting the case where *RFoot* is the support and *LFoot* is the swing foot.

skeleton to the foot frame.

According to the support, the robot control program will use a different model to solve the *IK*. When in *RS*, the right foot model will be used, and when in *LS*, the left foot model. When in *DS*, the right foot model will be used and the *IK* will include extra constraints to guarantee that the left foot is flat on the ground.

We need to find a transform from the absolute frame to the foot frame. However, recalling that we're only working with joint positions and ignoring orientations, we don't have a foot frame. So this frame will have to be constructed making use of other joint positions. For either foot, the 3 axes are chosen as follows:

- Origin: The origin is taken either as the right foot or the left foot.
- Z axis: The same as it is in the absolute frame [0,0,1].
- Y' axis: First it is taken as the direction from $\mathbf{P}_{RHip}$ to $\mathbf{P}_{LHip}$. It will be modified soon.
- X axis: The cross product $Y \times Z$. It points to the front of the body and is on the ground plane.
- Y axis: Finally, the Y axis is taken as the cross product $Z \times X$. This fixes the frame's orthogonality and places the Y axis on the floor plane.

After translating the data, the result is a scaled human like those previously shown on

Figure 6.4. From these joints, the 3 end-effector positions are sent to the robot control program, as well as the human joint angles. The end-effectors for each model are:

- *RFoot* model: *LHand*, *RHand*, *LAnkle*
- *LFoot* model: *LHand*, *RHand*, *RAnkle*

In the end, the data that is sent to the robot control program includes the Cartesian position for 3 end-effectors ($3 \times 3 = 9$), 12 joint angles, the current support (*RS*, *LS* or *DS*), and the support of the previous time step. The transformation matrix from the absolute frame to the current model is also sent, not to be used in the resolution of the *IK*, but so that the obtained results can be transformed back to the absolute frame later.

6.2.3 Robot Control

A program in *C++* is responsible for receiving data from the motion capture system, computing the *IK* and sending the results to the robot. This program is considerably more complicated than that of the continuous single support, because different robot models are used and transitions have to receive special care. A general flow of the program is shown in Figure 6.8.

Now the initial configuration is in *DS* with the arms positioned similarly to the *psi pose* and all the leg joints equal zero. As before, the data can come either from the shared memory segment or from a *CSV* file. This time, 35 values are being passed (9 coordinates, 12 joint angles, current support, previous support, 12 elements of the transformation matrix).

This time, the *IK* is solved slightly differently according to the support. Each continuous support (current support the same as the previous one) has a different set of goals and constraints. For support transitions, the solution is divided in 2 steps: placing *CoM* and placing swing foot.

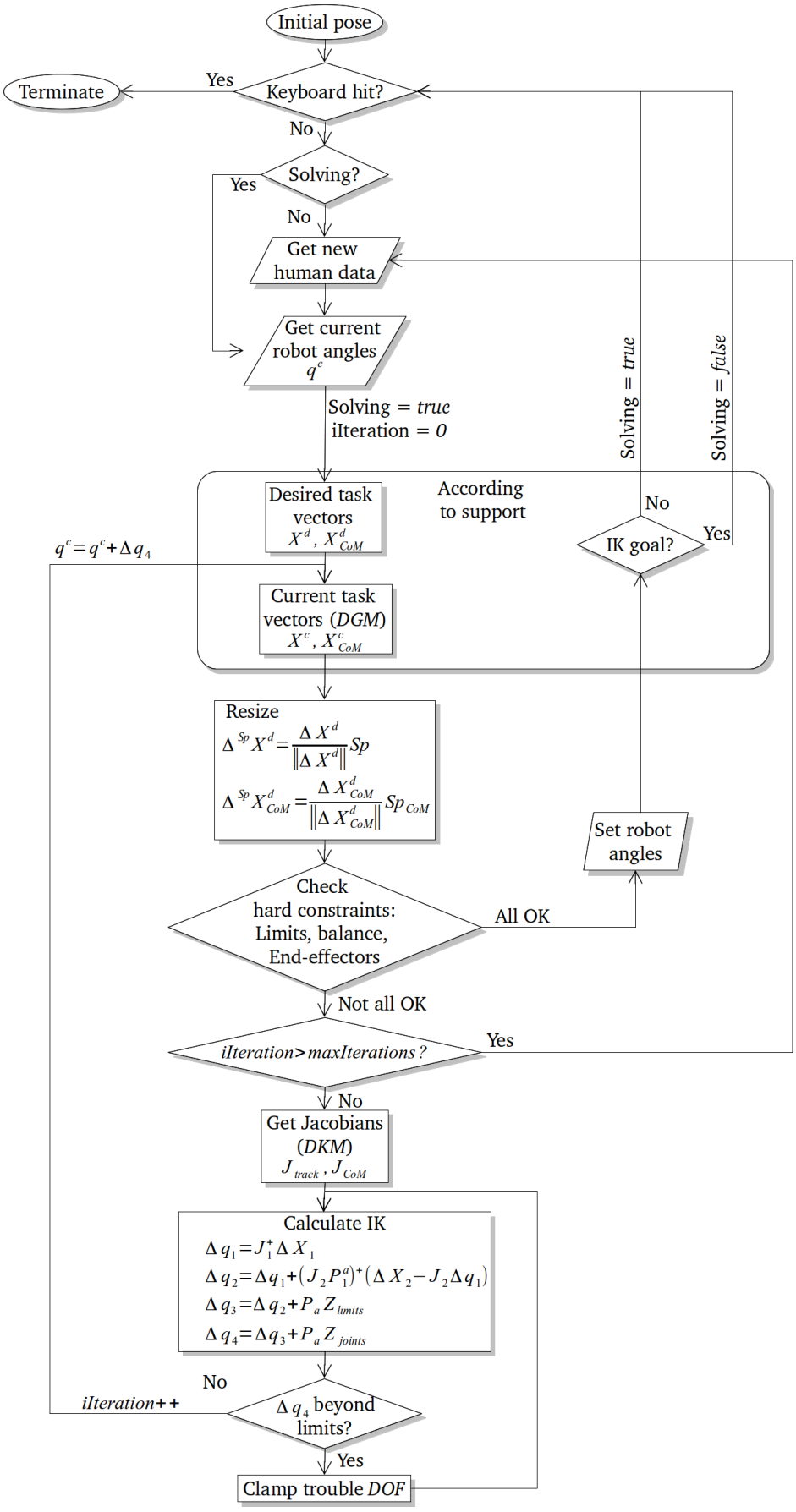


Figure 6.8: Flow chart of the robot control program including support changes.

6.2.3.1 Continuous support

The 3 types of continuous support are: *RS* to *RS*, *DS* to *DS* and *LS* to *LS*. A summary of their tracked coordinates is found on Figure 6.9.

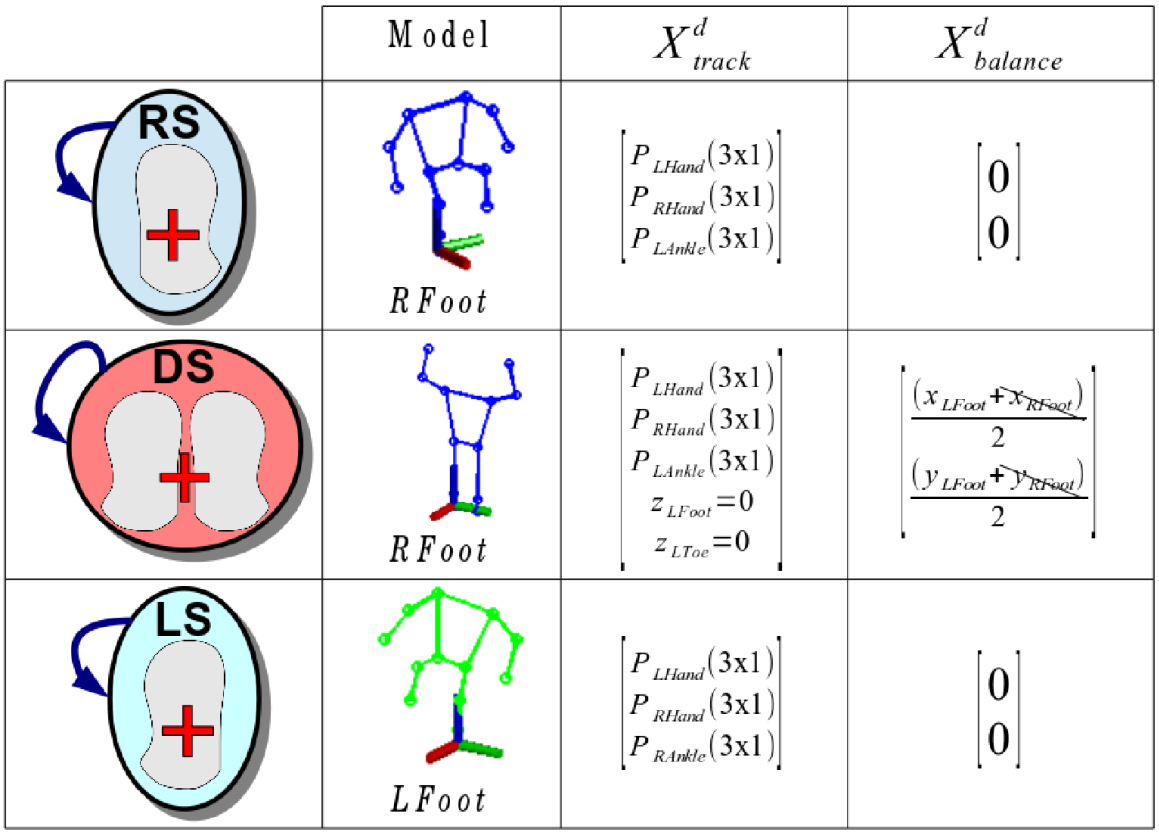


Figure 6.9: Tracked coordinates for continuous support.

Each continuous single support (*RS* to *RS* and *LS* to *LS*) uses the model referent to its support foot. Their desired end-effector positions $\mathbf{X}^d_{track}$ contain 9 values: the 3 Cartesian coordinates for each hand and for the swing ankle. And the desired *CoM* projection is $\mathbf{X}^d_{balance} = [0, 0]$ for both cases, which corresponds to the x, y coordinates of the support foot for the current model.

While the single supports are similar to what was done in the previous chapter, the *DS* to *DS* motion requires some modifications. To begin with, the desired *CoM* projection no longer falls onto the origin. Now it falls between the feet. Considering that *RFoot* is the origin, $\mathbf{X}^d_{balance}$ ends up being half the position of *LFoot*.

In *DS*, it is also important to keep the swing foot (*LFoot*) flat on the same plane as the support foot (*RFoot*). As before, the 3 Cartesian coordinates for the ankle are tracked (note that the z coordinate was fixed to *FootHeight* by the scaling). However, more constraints are needed in order to keep the foot flat. To keep the foot fixed on a same position on the floor, it is necessary to track 6 *DOF* (3 translations and 3 rotations).

Several attempts were made to fix 6 *DOF* of the foot, such as tracking 2 points among *LFoot*, *LAnkle* and a newly defined *LToe*, or tracking a point and its orientation. However, the *IK* wasn't able to converge for these constraints. Whenever the balance was satisfied, the foot couldn't be fixed. This is believed to be due to the robot's kinematic structure. The pelvis joint $(R/L)HipYawPitch$ is responsible for both feet's yaw, which are coupled to each other. Moreover, this joint greatly influences the position of the torso, which is the heaviest mass in the body, thus, *HipYawPitch* is very important for balance. When asking to place the *CoM* projection onto a specific point and at the same time fix the feet with a certain yaw angle between them, this *DOF* often cannot handle both tasks and the *IK* doesn't converge.

This poses a big problem to the imitation using this specific robotic platform. Either the balance constraint must be loosened in order to give more room for the feet yaw, or the foot constraints must be loosened. The balance constraint could be loosened to cover the whole support area instead of a specific point, but ultimately we'd still have 1 joint controlling 2 important tasks.

Thus, it was decided to loosen the constraints on the foot placement. For balance, it is very important that the foot remains flat on the floor, and that can be assured independently of the foot orientation about the vertical axis (yaw). If the foot rotation about the vertical axis is allowed, there are only 5 constraints left to keep the foot flat, leaving *HipYawPitch* to do mainly the torso positioning. This allows the feet to slide on the floor rotating about the vertical axis, which can be dangerous in case of high friction. However, most of the time, allowing yaw only influences the visual quality of the motion and does not endanger balance. Given the kinematic limitations of the *NAO* robot, this was considered an acceptable compromise.

After several attempts on combinations of 5 *DOF* to track, it was observed that tracking 3 ankle coordinates and setting the z coordinate of *LFoot* and of *LToe* to zero provided conversion. Tracking orientations about the horizontal axes, for example, rarely converged. It would be interesting to investigate the reasons for that in a future work. Note that *LToe* can be found from the *DGM* given $\mathbf{q}^c$, but not from the *Kinect*. However, we're only interested in setting its vertical coordinate to zero, so we don't need to obtain it from the human.

It's also worth noting that geometrically, tracking only 3 *DOF* for the ankle and the z coordinate of the foot should be enough to constrain 5 *DOF* since the fixed distance between these points acts as an extra constraint. However, due to precision errors, this was generating little shaking of *RAnklePitch* which made the whole robot very unstable. Tracking *LToe*'s vertical position decreased shaking on that joint, but still allows some shaking of the *RAnkleRoll* joint, which at times increases the robot's instability.

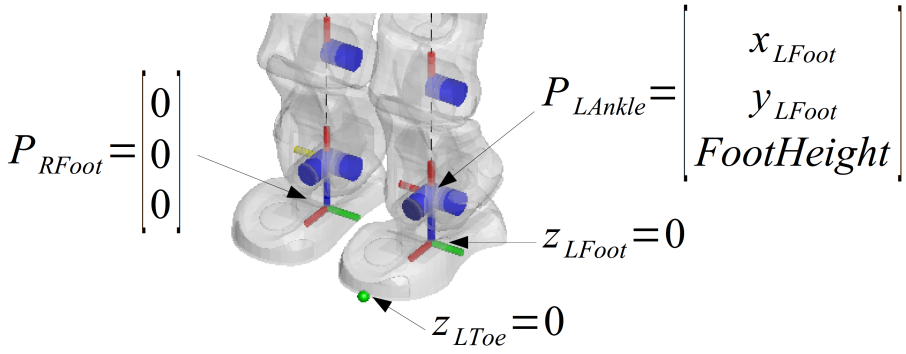


Figure 6.10: Tracked coordinates so as to keep *LFoot* flat on the ground during *DS*.

6.2.3.2 Solving transitions

For single support, if the goal $\mathbf{X}^d$ was further than *Sp*, only ${}^{Sp}\mathbf{X}^d$ would be covered, $\mathbf{X}^d$ was thrown away and a new goal was asked from the motion capture system. Now, the same is done for continuous support phases but not during transition.

For transitions, it is of utmost importance that the goal is reached, as we will see below. Thus, even after an intermediate goal ${}^{Sp}\mathbf{X}^d$ is completed, the final goal $\mathbf{X}^d$ must be fulfilled before new data is acquired. While the final goal hasn't been reached, the program is said to still be "solving" for the previous data, so no new data is acquired until the final goal has been reached.

6.2.3.3 Transition from *DS* to *SS*

When taking off a foot from the ground, the projection of the *CoM* must be moved to the support foot. Also, the swing foot must take off perfectly parallel to the floor, so it doesn't "kick" the ground pushing the whole body upwards. In order to avoid a loss of balance, these transitions have to be performed smoothly. While transitioning, it is of utmost importance that the feet are placed as desired. The locus of attention is set to the feet, so the hands' constraints were released.

Figure 6.11 shows the tasks for the *DS* to *RS* transition. The first step is to move the *CoM* projection from the middle of the feet to *RFoot*. The effector vector $\mathbf{X}^d_{track}$ remains the same as for the continuous *DS* on Figure 6.9. To increase stability during support change, the *CoM* goal $\mathbf{X}^d_{balance}$ is set not to the projection of the ankle $[0, 0]$, but to a point closer to the center of the foot with an added offset on the X axis $[CoMoffset, 0]$. The offset was set to $15mm$. Once the continuous support *RS* to *RS* begins, the *CoM* goal goes back to the ankle projection in order to minimize the torque needed from the ankle motors.

The second step of the transition is to detach *LFoot* from the ground while keeping it parallel. This is done by incrementing the z coordinates of *LAnkle*, *LFoot* and *LToe* little by little until a predetermined *LiftHeight* is reached (using $LiftHeight = 30mm$).

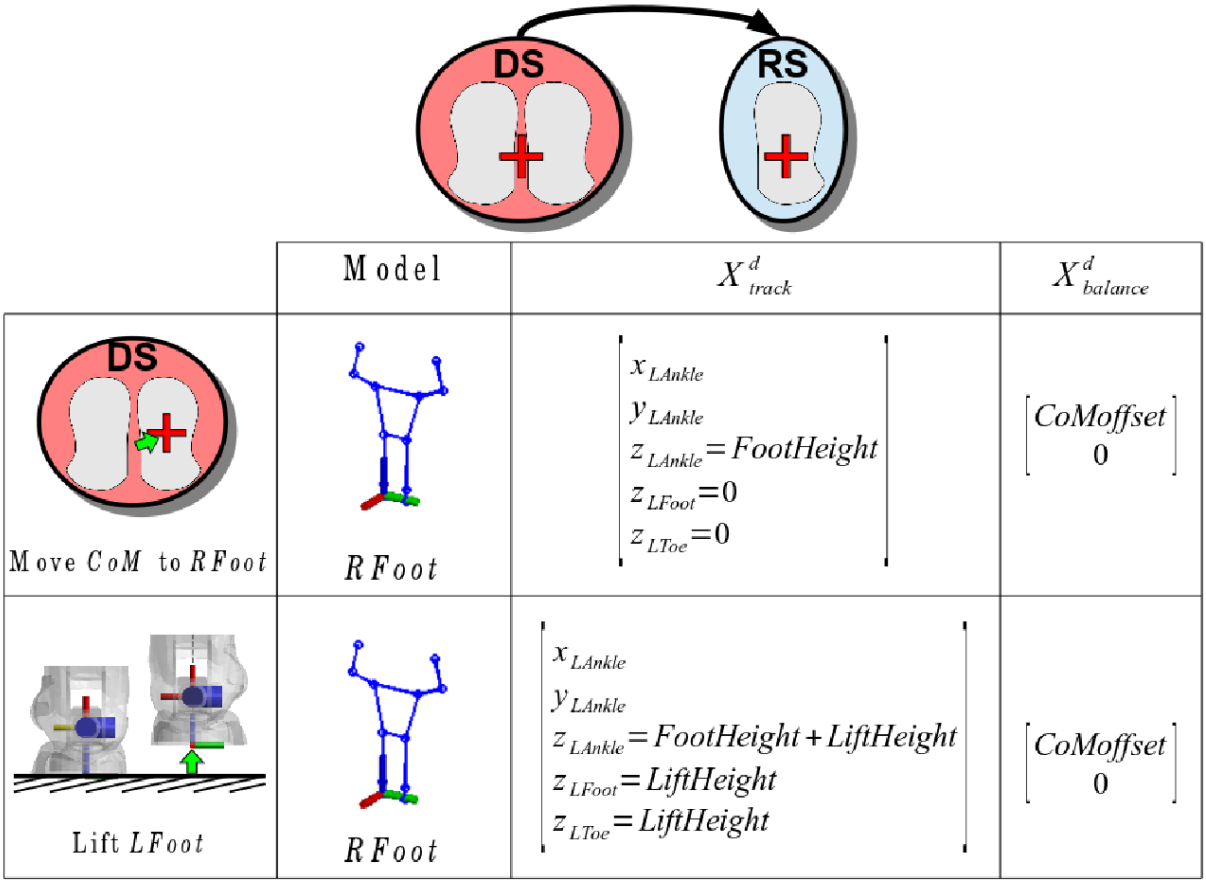


Figure 6.11: Tracked coordinates during support transition, *DS* to *RS* example.

The *DS* to *LS* transition occurs in a similar manner. Using *RFoot* model, the *CoM* is placed onto *LFoot*. Then the *LFoot* model is used with constraints to keep *RFoot* parallel to the ground while it is lifted to *LiftHeight*.

6.2.3.4 Transition from *SS* to *DS*

The steps of *SS* to *DS* transitions go on a reverse order: first, the swing foot must be lowered to the ground while parallel, and only after that the *CoM* is placed between the feet. A scheme is seen on Figure 6.12.

Even after mapping the flat foot area and not tracking the hands, there are times when the *IK* cannot find a solution to place the foot on the desired position. That's because the conversion depends not only on the goal but also on the starting point. Thus, as a last resort to try to ensure convergence, in case tracking the 5 *DOF* is not possible, only 2 *DOF* are tracked, namely z_{LFoot}, z_{LToe}. This is dangerous, as the foot could be placed anywhere, including on top of the other foot. However, it is one more attempt to insure conversion which works most of the time, as will be shown on the results. If not even this is possible, the imitation fails.

For *LS* to *DS*, the process is similar, but the *RFoot* is lowered using the *LFoot* model,

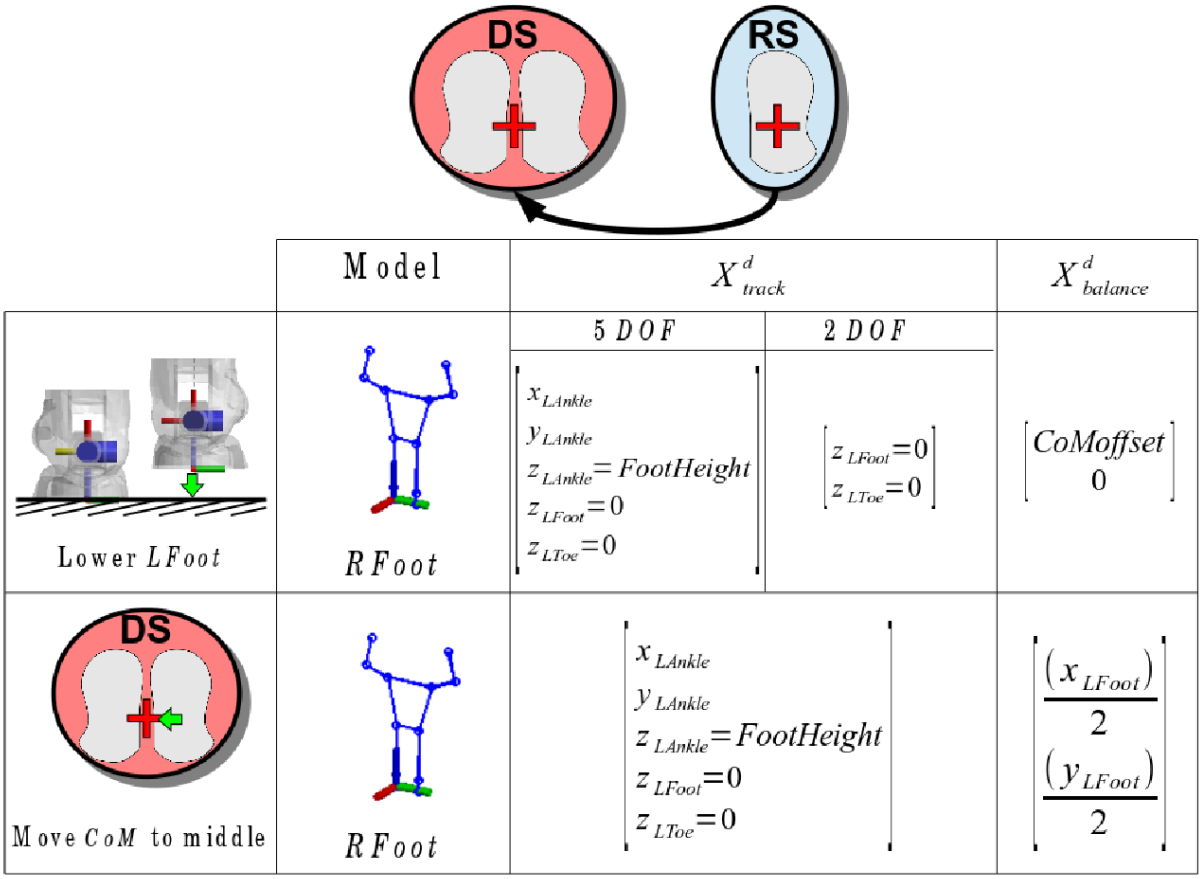


Figure 6.12: Tracked coordinates during support transition, *RS* to *DS* example.

and the *CoM* is placed the exact same way, with the *RFoot* model.

6.3 Results

6.3.1 End-effector tracking

Figure 6.13 shows the imitation online for a movement that starts with both feet on the floor, lifts the left foot, then places the foot again on the floor on another position. In other words, a step forward. The tracking of end-effectors with respect to the absolute frame is shown on Figure 6.14.

For the *DS* to *RS* transition, the robot takes a while to finally lift the foot from the floor, since it has to carefully move the *CoM* and lift the foot parallel to the ground. After that, it is seen that the robot catches up with the human's movement in *RS*.

Note that even though the hand's constraints were released, the hands didn't move much away from the previous position. That's thanks to the pseudo-inverse, which finds the solution which requires the minimum joint movement possible. During other experiments

Figure 6.13: Moments during support transition, here showing *DS-RS-DS*.

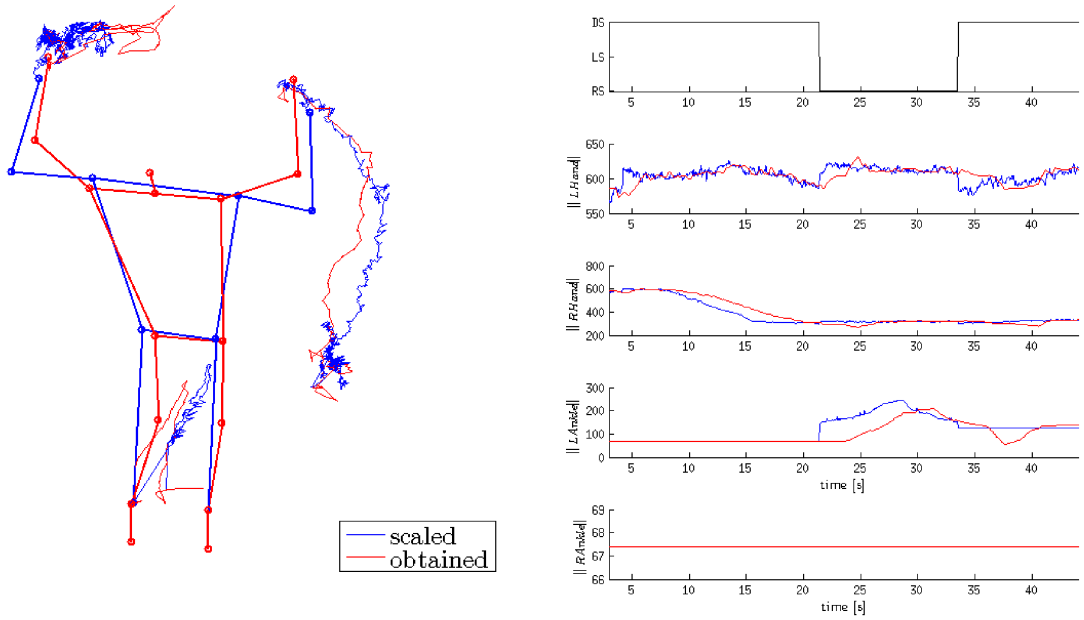


Figure 6.14: End-effector tracking including support changes: *DS-RS-DS*.

however, it was observed that the hands might move away during transition. They quickly catch up with the new goal after transition though.

During the *RS* to *DS* transition, it can be seen that the foot wasn't initially placed on its final goal. In fact, what happened was that the *IK* couldn't be solved with the 5 *DOF* constraint, and instead was solved while just keeping the left foot parallel to the ground, which was enough to insure balance. Once the foot was already on the ground, during the following time steps in *DS*, the foot was correctly slid to the position it should be on the ground. This position can for sure be reached sliding the feet since the rectangle area was previously delimited. The detour and correction described in this paragraph occurred between $35s$ and $40s$. The foot placement seems to happen as often with 2 *DOF* as with 5 *DOF*, but more rigorous experiments must be conducted to verify this.

6.3.2 Balance

For the previous movement, the projection of the *CoM* on the floor and the position of the support feet are plotted on Figure 6.15. The position of the feet on the floor is taken from the *DGM* used on the joint values returned by the robot. However, this model doesn't take slipping into account, and due to the constant yaw rotation of the feet, both feet were slipping during the motion. Therefore, the results are illustrative of how the *IK* works rather than the actual position of the robot in space. For an accurate reading of the robot's position in space, sensors external to the robot should be used.

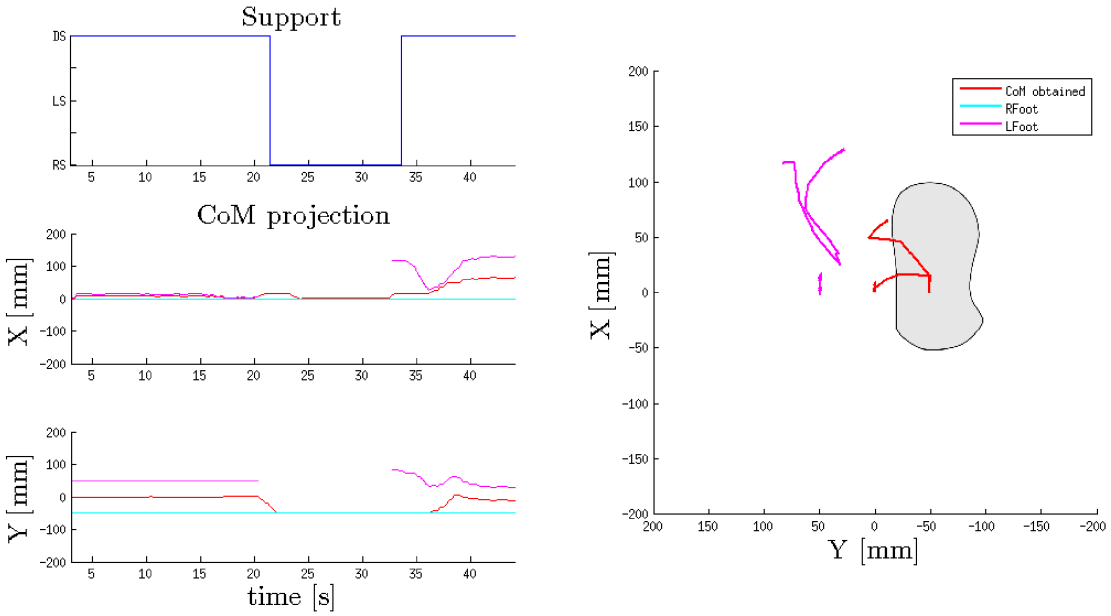


Figure 6.15: CoM projection with foot placement.

Up to 20*s*, the robot was in continuous *DS*. Note that even though the feet are supposed to be fixed, *LFoot* moves. In reality both feet were sliding with respect to the absolute frame, but since our model is based on *RFoot*, only their relative positions are taken and it seems like *LFoot* was the only one sliding.

The moment the transition to *RS* starts, the *CoM* projection begins moving from the point in-between the feet towards *RFoot*. *LFoot* actually only separates from the floor once the *CoM* projection has reached *RFoot*. Throughout *RS*, the *CoM* projection remains in the vicinity of *RFoot*.

As explained in the previous section, during this movement, the *IK* wasn't able to place *LFoot* directly on the desired position, so it was placed wherever it was the most convenient for the *IK* while staying parallel to the ground. The movement of *LFoot* can be clearly seen here. It landed onto a random position and when the continuous *DS* phase began it slid to the originally intended position. Note that the projection of the *CoM* remains between both feet even during the sliding.

6.3.3 Joint constraints

In the previous chapter, the joint limit avoidance and human joint tracking were validated (Section 5.3.3). In this chapter, the tuning of the minimization tasks' weights and the need for the clamping loop will be explained in more detail.

We saw that for $\kappa_{limits} = -0.1$ and $\kappa_{limits} = -0.02$, used together with a clamping loop, the imitation has a good convergence rate and results in movements close to the human's whenever it is possible. Now, a smooth motion captured with the marker-based system was imitated offline in 10 experiments varying the joint-related parameters. The movement consists of waving both hands bending the elbows similarly to the *psi pose*, always in double support (good motion example shown on Figure 6.16, top). The results are summarized on Table 6.2.

Table 6.2: Offline experiments varying the joint constraints for the same motion.

Exp.	κ_{joints}	κ_{limits}	Clamping	Converges?	Conversion fail	Similar to human?
1	-0.02	-0.1	On	Almost always	×	Yes
2	0	-0.1	On	Sometimes	Balance and tracking	Yes
3	-0.1	-0.1	On	Sometimes	Balance and tracking	Yes
4	-0.5	-0.1	On	Rarely	Balance and tracking	×
5	-0.1	-0.02	On	Never	Balance and tracking	×
6	-0.02	0	On	Sometimes	Balance and tracking	Somewhat
7	0	0	On	Almost always	×	No
8	-0.02	-0.1	Off	Sometimes	Limits	Yes
9	-0.1	-0.1	Off	Rarely	Limits	×
10	-0.02	0	Off	Never	Limits	×

Whenever the clamping loop is turned off (experiments 8,9,10), the convergence begins to fail when it comes to the constraint responsible for keeping the joint values within their limits. This proves that just the minimization task using κ_{limits} is not enough to keep the joints within their limits. Notably, on the last experiment, when both methods of joint avoidance are turned off, no conversion is possible since there's always a joint outside limits.

On the other hand, when the clamping loop is used without the minimization, some joints were clamped to their limits, most notably *(R/L)ElbowYaw*, resulting in a motion that looks quite different from the human's. It can be seen on Figure 6.16(middle) that the hands were turned backwards for experiment 6. This reinforces the idea of the clamping loop as a last resort and not a main joint avoidance strategy. Ideally, joints are clamped to their limits only when there's no alternative. To this end, the minimization task helps

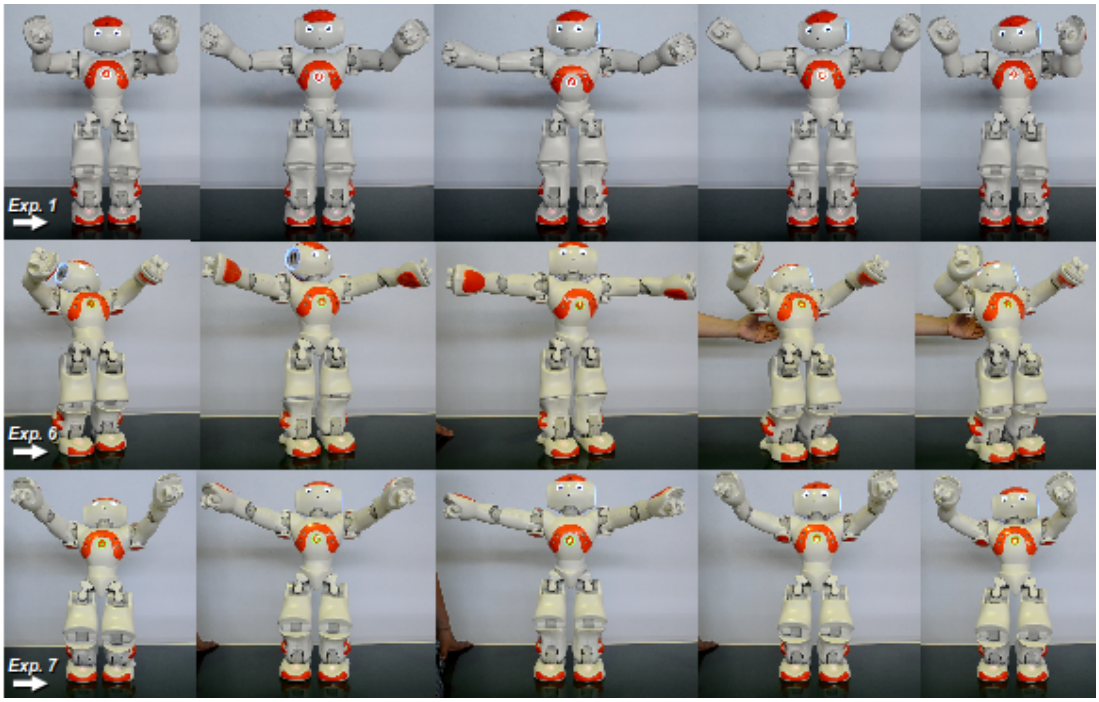


Figure 6.16: Waving motion results for different joint constraints.

the loop not to be overused.

To see the influence of κ_{joints}, it is better to set $\kappa_{limits} = 0$, so the limit avoidance minimization didn't influence the joint values. For example, let's compare experiments 6 and 7. For the motion studied, the main human angles being tracked are the *ElbowRoll*s. On experiment 6, the robot's elbows bent similarly to the human's, although the clamped *ElbowYaw*s made the hand orientation look unnatural. On experiment 7 however (see Figure 6.16, bottom), the absence of the joint tracking minimization made the robot "wave" with the arms completely stretched, which was very different from the human's.

The influence of κ_{joints} for experiments including κ_{limits} is not so clear for this motion. For example, experiment 2 presented a motion similar to the performer's even though the joint tracking was off. That's because the limit avoidance minimization was keeping the elbows far from their limits and bent.

It was also understood that high weights disturb the equality tasks (besides experiment 4, non-listed experiments showed that weights of -0.5 and -1 hardly converge). Also, the relation between the weights must be $\kappa_{limits} > \kappa_{joints}$: experiments 3, 5 and 9, which had larger or equal κ_{joints} proved that tracking the human joints too much disrupts other tasks. This is expected, since the limits avoidance task is a higher priority task and the human tracking is the task with the lowest importance.

6.3.4 Time constraint

For imitation with supports, the blocks which were the same as before took similar average time as shown in Section 5.3.5. In this chapter, it is interesting to notice the different times to solve each kind of support for several motions by several actors.

Continuous supports took an average of about $0.7s$ to be solved, including times when no solution was found and 20 iterations were calculated, which can take more than $3s$.

The transitions however took much longer. Since their goal is not being resized, the final goal is pursued no matter how far it is. Plus, the solution is divided in 2 phases. The observed times are shown on Table 6.3, all times given are in seconds. At least 4 experiments for each transition were performed.

Table 6.3: Time taken to perform support transitions.

Support	**Fastest**	**Slowest**	**Average**
RS to *DS*	5.1700	12.5670	8.2080
LS to *DS*	4.5090	7.6510	6.6444
DS to *RS*	2.8100	5.0170	3.9759
DS to *LS*	3.0480	7.3690	5.3636

In general, transitions into *DS* took longer than transitions to *SS*. This is expected, as it is more difficult to find a good position to place the foot on the floor than it is to lift it. The transitions are clearly the most difficult moments during imitation, and ways of obtaining faster transitions are necessary.

6.4 Conclusions

In this chapter, task-based inverse kinematics were extended to imitate human motion with support changes. This allowed for the human and robot to lift and land feet and locomote in space. The human motion was scaled taking into account support changes and the inverse kinematics had different constraints and different robot models were used according to the support.

Only static balance was considered, so the motions being imitated have to be slow. In particular, the support transition times can take more than $10s$ for the robot while it only takes a fraction of a second for a human. It was shown that the end-effectors are properly tracked with respect to the world frame and that the center of mass projection moves properly according to the support. It was also shown that the joint limit clamping loop is necessary and that the joint tracking minimization indeed improves the quality of imitation.

The implementation was tested with several actors, including people who hadn't seen the system being used before. For all experiments, the results were satisfactory, which showed that the method implemented is intuitive and works well.

There are still several limitations, besides those inherent to a kinematic approach, to be overcome. The most limiting factor for the support changes was the particular kinematic arrangement of the NAO robot's legs, which makes it difficult to freely place feet on the floor while maintaining static balance.

Chapter 7

Conclusions

7.1 Report conclusions

Humanoid robots are redundant open serial chains not fixed to their environment with a few dozen *DOF*. In order to cope with this complexity, the approach of imitation uses captured human motion as the basis for motion generation. This approach takes much of the burden of motion prediction away from the robot.

Due to the kinematic and dynamic differences between human beings and humanoid robots, motion transfer is not a straightforward matter. Besides taking into account the physical limitations of the robot, issues such as kinematic singularities, self-collision and balance must be addressed. Moreover, for real-time imitation, time constraints must be enforced.

In this research, whole-body kinematic imitation of human motion by the humanoid robot *NAO* using task specification inverse kinematics was implemented with four tasks: tracking end-effectors, keeping static balance, avoiding joint limits and tracking human joint values. Mainly a markerless motion capture systems was used online, and a marker-based system with better accuracy was used for some offline experiments.

A method to scale the human motion to the robot's dimensions one body segment at a time was introduced. The scaled end-effector positions in Cartesian space were tracked while keeping static balance for single support on either foot, double support and transition between supports. The joint limits were avoided by a combination of a clamping loop and a task which minimizes the distance from the average joint value. The joint values were kept as close as possible to the performer's whenever that didn't affect the higher priority tasks to imitate the manner as well as possible.

A graphical user interface which can be easily used online for the *Kinect* sensor and the *NAO* robot was developed. Experiments with several performers showed satisfactory results for the imitation of a wide variety of slow movements. Compared to other methods in the literature, the present approach better preserves the nuances of human motion

during whole-body online teleoperation while working with an uncluttered motion capture system.

7.2 Perspectives

The developed method is a step towards more accurate and robust online teleoperation of humanoid robots using human motion. There are several paths which follow-up research could take. Throughout the report some recommendations for future research can be seen. Some other suggestions follow:

- Currently the head is not being tracked either as an end-effector or through direct kinematics. Since it is heavy, it was left free to move as it might aid the balance task. It would be interesting however to add head tracking as well, preferably as a minimization task or even directly sending the human angles and removing the head joints from the inverse kinematics.

- Applying the current algorithm to a robot with more *DOF* such as *Romeo* or *HRP-2* could provide further validations and verify whether the kinematic limitations of the *NAO* robot were really the main constraints to a better support change.

- Tracking the *ZMP*, rather than the projection of the *CoM* could extend the method to fast dynamic motions. This would require a program running in parallel to the *IK* so the acceleration of the robot can be better estimated even during blocking calls.

- The balance task could be extended to an inequality task to place the *CoM* projection anywhere in the support polygon, loosening balance constraints.

- A smoother way of sending results to the robot would greatly improve the quality of motion. A delay of a few samples could be introduced and these samples interpolated. It would also be nice to have better control over the velocity at each time-step, unlike the current functions in the robot's *API*, which slow down to a halt at each instruction sent, making the robot jerk.

- The time constraint could be reinforced in the same way as the number of iterations is being limited. That is, if the solution is taking more than the allowed period, the current goal can be abandoned and new data from the human obtained or a new starting point for the *IK* can be chosen. This would allow for the program to run in a fixed frequency.

Bibliography

[1] K. Čapek and P. Selver, *R.U.R. (Rossum's Universal Robots): a play in three acts and an epilogue*. H. Milford, Oxford University Press, 1928.

[2] C. C. Kemp, P. M. Fitzpatrick, H. Hirukawa, K. Yokoi, K. Harada, and Y. Matsumoto, "Humanoids," in *Springer Handbook of Robotics*, pp. 1307–1333, 2008.

[3] M. Akhratuzzaman and A. A. Shafie, "Advancement of androids and contribution of various countries in the research and development of the humanoid platform.," *IJRA the International Journal of Robotics and Automation*, vol. 1, Issue 2, pp. 43–57, 2010.

[4] P. Bakker and Y. Kuniyoshi, "Robot see, robot do : An overview of robot imitation," in *In AISB96 Workshop on Learning in Robots and Animals*, pp. 3–11, 1996.

[5] E. Hanavan, "A mathematical model of the human body," Final Report AMRL-TR-64-102, Aerospace Medical Research Laboratories, Oct. 1964.

[6] C. Breazeal and B. Scassellati, "Robots that imitate humans," *Trends in Cognitive Sciences*, vol. 6, no. 11, pp. 481 – 487, 2002.

[7] M. J. Mataric, "Getting humanoids to move and imitate," *IEEE INTELLIGENT SYSTEMS*, vol. 15, pp. 18–24, 2000.

[8] M. Do, P. Azad, T. Asfour, and R. Dillmann, "Imitation of human motion on a humanoid robot using non-linear optimization.," in *Humanoids*, pp. 545–552, IEEE, 2008.

[9] R. O. Ambrose, H. Aldridge, R. S. Askew, R. R. Burridge, W. Bluethmann, M. Diftler, C. Lovchik, D. Magruder, and F. Rehnmark, "Robonaut: Nasa's space humanoid," *Intelligent Systems and Their Applications, IEEE*, vol. 15, no. 4, pp. 57–63, 2000.

[10] S. Nakaoka, A. Nakazawa, F. Kanehiro, K. Kaneko, M. Morisawa, H. Hirusawa, and K. Ikeuchi, "Leg task models for reproducing human dance motions on biped humanoid robots [in japanese]," *Journal of the Robotics Society of Japan*, vol. 24, pp. 388–399, apr 2006.

[11] P. Morasso and V. Sanguineti, "Human motion analysis," *Wiley Encyclopedia of Biomedical Engineering*, vol. 17, no. 1, pp. 12–5, 2006.

[12] R. Slyper and J. K. Hodgins, "Action capture with accelerometers," in *Symposium on Computer Animation*, pp. 193–199, 2008.

[13] F. Montecillo P., M. Sreenivasa, and J.-p. Laumond, "On real-time whole-body human to humanoid motion transfer," *International Conference on Informatics in Control, Automation and Robotics (ICINCO 2010)*, pp. 22–31, 2010.

[14] L. Cheze, B. Fregly, and J. Dimnet, "A solidification procedure to facilitate kinematic analyses based on video system data," *Journal of Biomechanics*, vol. 28, no. 7, pp. 879–884, 1995.

[15] M. Riley, A. Ude, and C. G. Atkeson, "Methods for motion generation and interaction with a humanoid robot: Case studies of dancing and catching," in *Proc. 2000 Workshop on Interactive Robotics and Entertainment, Robotics Inst., Carnegie Mellon Univ*, pp. 35–42, 2000.

[16] "Kinect for windows sensor components and specifications." http://msdn.microsoft.com/en-us/library/jj131033.aspx, February 2013.

[17] V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast, "Dynamic response for motion capture animation," *ACM Trans. Graph.*, vol. 24, pp. 697–701, July 2005.

[18] R. Baker, "Isb recommendation on definition of joint coordinate systems for the reporting of human joint motionpart i: ankle, hip and spine," *Journal of Biomechanics*, vol. 36, no. 2, pp. 300 – 302, 2003.

[19] M. Yeadon, "The simulation of aerial movementii. a mathematical inertia model of the human body," *Journal of Biomechanics*, vol. 23, no. 1, pp. 67 – 74, 1990.

[20] Y. H. Kwon, "Modified hanavan model." http://www.kwon3d.com/theory/bspeq/hanavan.html, 1998. accessed: February 2013.

[21] W. Khalil and E. Dombre, *Modeling, Identification and Control of Robots*. Bristol, PA, USA: Taylor & Francis, Inc., 3rd ed., 2002.

[22] H. J. Shin, J. Lee, S. Y. Shin, and M. Gleicher, "Computer puppetry: An importance-based approach," *ACM Transactions on Graphics*, vol. 20, pp. 67–94, 2001.

[23] B. Dariush, M. Gienger, B. Jian, C. Goerick, and K. Fujimura, "Whole body humanoid control from human motion descriptors," *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 2677–2684, 2008.

[24] S. Nakaoka, A. Nakazawa, K. Yokoi, H. Hirukawa, and K. Ikeuchi, "Generating whole body motions for a biped humanoid robot from captured human dances," *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 3, pp. 3905–3910, 2003.

[25] E. S. Neo, K. Yokoi, S. Kajita, F. Kanehiro, and K. Tanie, "Whole body teleoperation of a humanoid robot -a method of integrating operator's intention and robot's autonomy," in *ICRA*, pp. 1613–1619, 2003.

[26] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Resolved momentum control: humanoid motion planning based on the linear and angular momentum," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, vol. 2, pp. 1644–1650, IEEE, 2003.

[27] N. Pollard, J. Hodgins, M. Riley, and C. Atkeson, "Adapting human motion for the control of a humanoid robot," in *Proceedings 2002 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1390–1397, IEEE, 2002.

[28] A. Safonova, N. Pollard, and J. Hodgins, "Optimizing human motion for the control of a humanoid robot," *2nd International Symposium on Adaptive Motion of Animals and Machines (AMAM2003)*, 2003.

[29] W. Suleiman, E. Yoshida, F. Kanehiro, and J.-P. Laumond, "On human motion imitation by humanoid robot," *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 2697–2704, 2008.

[30] M. Ruchanurucks, S. Nakaoka, S. Kudoh, and K. Ikeuchi, "Humanoid robot motion generation with sequential physical constraints," *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2649 – 2654, 2006.

[31] F. Kanehiro, E. Yoshida, F. Lamiraux, O. Kanoun, and J.-P. Laumond, "A local collision avoidance method for non-strictly convex objects," *Journal of the Robotics Society of Japan*, vol. 27, no. 1, pp. 63–70, 2009.

[32] J. Koenemann and M. Bennewitz, "Whole-body imitation of human motions with a nao humanoid," in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, HRI '12, (New York, NY, USA), pp. 425–426, ACM, 2012.

[33] M. Vukobratovic, "On the stability of anthropomorphic systems," *Mathematical Biosciences*, vol. 15, pp. 1–37, Oct. 1972.

[34] M. Vukobratovic and B. Borovac, "Zero-moment point - thirty five years of its life.," *I. J. Humanoid Robotics*, vol. 1, no. 1, pp. 157–173, 2004.

[35] A. Dasgupta and Y. Nakamura, "Making feasible walking motion of humanoid robots from human motion capture data," in *ICRA*, pp. 1044–1049, 1999.

[36] S. Kim, C. Kim, B. You, and S. Oh, "Stable whole-body motion generation for humanoid robots to imitate human motions," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2518–2524, Oct. 2009.

[37] A. Ude, C. G. Atkeson, and M. Riley, "Programming full-body movements for humanoid robots by observation.," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 93–108, 2004.

[38] M. Riley, A. Ude, K. Wade, and C. Atkeson, "Enabling real-time full-body imitation: A natural way of transferring human movement to humanoids," *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 2, pp. 2368–2374, 2003.

[39] E. Demircan, L. Sentis, V. Sapio, and O. Khatib, "Human motion reconstruction by direct control of marker trajectories," *Advances in Robot Kinematics: Motion in Man and Machine*, pp. 283–292, 2008.

[40] A. Billard and M. Mataric, "Learning human arm movements by imitation: Evaluation of a biologically-inspired connectionist architecture," *Robotics and Autonomous Systems*, vol. 37, no. 2-3, pp. 145–160, 2001.

[41] N. Naksuk, C. S. G. Lee, and S. Rietdyk, "Whole-body human-to-humanoid motion transfer.," in *Humanoids*, pp. 104–109, IEEE, 2005.

[42] N. Mansard, *Enchanement de tches robotiques*. PhD thesis, University of Rennes, December 2006.

[43] A. Liegeois, "Automatic supervisory control of the configuration and behavior of multibody mechanisms," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 7, no. 12, pp. 842–868, 1977.

[44] P. Baerlocher and R. Boulic, "An inverse kinematic architecture enforcing an arbitrary number of strict priority levels," *The Visual Computer*, vol. 20, pp. 402–417, 2004.

[45] A. Robotics, "Nao humanoid robot documentation 1.12.3," 2012.

[46] "Aldebaran robotics." http://www.aldebaran-robotics.com/, February 2013.

[47] "Openni." http://www.openni.org/, June 2013.

[48] ART GmbH, *System user manual: ARTtrack, TRACKPACK, DTrack*, 2012.

[49] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[50] "Qt project." http://qt-project.org/, July 2013.

[51] D. Cehajic, "Using human motion to control the humanoid robot nao," Master's thesis, Ecole Centrale de Nantes, 2012.

Appendices

Appendix A

Graphical interface

A *graphical user interface* (*GUI*) was made in *Qt* in order to aid in the development and analysis, as well as to make the program easily accessible to other users. The interface is seen on Figure A.1.

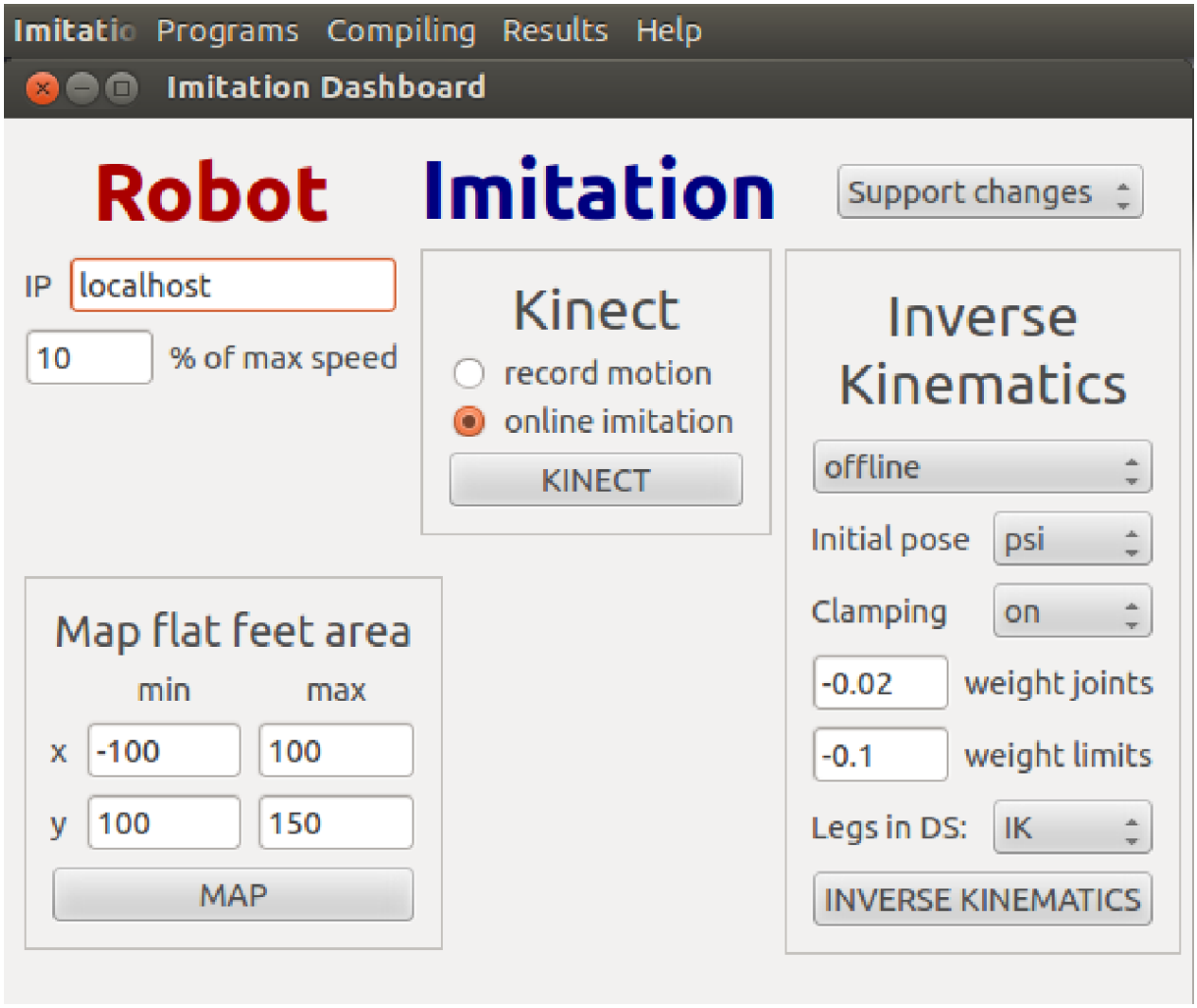


Figure A.1: Imitation dashboard *GUI*.

This *GUI* allows the user to run both the *Kinect* and the robot control programs. The *Kinect* program can either be used to record a motion to be used later offline or online

at the same time as the *IK* program.

Several parameters can be set for the *IK* program. It can be used online or offline, in the latter case after making sure the needed *CSV* file is in the right folder. This file may have come from any type of motion capture system, as long as it is in the desired format. The initial pose can be chosen between the *psi pose* and the *zero pose* - normally the *psi pose* should be used, but in case the recorded motion starts differently, the *zero pose* might be a closer start. The clamping loop can be turned off and the weights for minimization tasks can be chosen.

The option to fix legs during *DS* is there in case the user doesn't want the robot's legs to be rotating during *DS*. Instead, if *DK* is chosen, the legs are fixed and the arms are controlled directly with the human joint angles.

The map flat feet area program makes the robot trace the rectangle explained on Section 6.2.2.3. From the *GUI*, it is also possible to directly start related software, such as *Webots*, *Choreographe*, *NAOqi simulator* and *CMake*. The *Compiling* menu gives direct access to the place where each of the programs can be compiled and the *Result* menu to the folders where results are saved.