



UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS



FONDAZIONE ISTITUTO ITALIANO DI TECNOLOGIA

**Methods to improve the coping capacities
of whole-body controllers for
humanoid robots**

by

Marie Charbonneau

Thesis submitted for the degree of *Doctor of Philosophy* (31° cycle)

March 2019

Francesco Nori, *Fondazione Istituto Italiano di Tecnologia*

Supervisor

Daniele Pucci, *Fondazione Istituto Italiano di Tecnologia*

Supervisor

Giorgio Cannata, *University of Genova*

Head of the PhD program

Thesis Jury:

Adrea Del Prete, *Max Planck Institute for Intelligent Systems*

External examiner

Michael Mistry, *University of Edinburgh*

External examiner

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Marie Charbonneau
March 2019

Acknowledgements

The work of this thesis has received funding from the Marie Skłodowska-Curie Action European Training Network SECURE funded by the European Commission (grant agreement No.642667), as well as the European Union's Horizon 2020 research and innovation program project An.Dy (grant agreement No. 731540), in collaboration between the Dynamic Interaction and Control (DIC) laboratory of the iCub Facility, Fondazione Istituto Italiano di Tecnologia (IIT), Genoa, Italy and the LARSEN Team of INRIA Nancy - Grand Est, Villers-lès-Nancy, France.

First of all, thanks to Francesco Nori who made it possible for me to join the Dynamic Interaction Control lab at IIT, and Daniele Pucci for his supervision and getting me started with whole-body control. They both granted me the opportunity to learn, to work with the iCub, and to build collaborations beyond the lab, for instance with researchers from INRIA Nancy and Sapienza University of Rome.

Sincere acknowledgements are extended to the reviewers, Drs. Andrea Del Prete and Michael Mistry, who have freely given of their time and expertise.

I would like to sincerely thank Serena Ivaldi for receiving me in her team at INRIA, providing me with patient explanations, as well as the support and resources I needed. Through Serena, I got to collaborate with Valerio Modugno and Luigi Penco, who, on top of being great coworkers, have been precious sources of positivity. Their optimism turned out to be precisely what I needed in order to push my work further.

Many thanks also to my colleagues (both in IIT and INRIA) for the fruitful exchanges during the last few years. Many thanks to Francisco Andrade, Brice Clement, Stefano Dafarra, Augusto Francisco, Nuno Guedelha, Dorian Goepf, Joan Kangro and especially Gabriele Nava and Luigi Penco for lending a hand when doing experiments with the robot, and also for the cheerfulness. Thanks to Julien Jenvrin for fixing the robot when I would break it, and reminding me to keep smiling. Heartfelt thanks also to Aiko Dinale, who consistently took care to bake cakes which I could

eat, when she brought some to the lab. Finally, Silvio Traversaro has been a model on mastering deeply a field of work, and even though being always so busy, remaining incredibly available for clear, complete explanations and help when needed.

This thesis was characterized by a certain degree of mobility. I would like to thank all those who provided me with a place to work and write at one point or another, including Professor Giuseppe Oriolo at Sapienza University of Rome, Professors Alexandre Bergel and Jocelyn Simmonds at the University of Chile, and Dr. Andreea Radulescu, who lent me a corner of her dining table and also checked on my sanity during the writing.

Special mentions go to Augusto Francisco, Nico Huebel and Naveen Kuppuswamy, for the deep discussions we had on thorny subjects I encountered during my PhD, helping me to gain a clearer viewpoint.

Many thanks to my friends and close ones, for the caring support and reminding me of the importance of small things, helping to fix my headaches, taking care of my stomach, and bringing me ice cream on deadline nights. Although in the past I had generally overcome challenges by myself, for this work, my parents did help me with impactful advice that I am deeply grateful for.

Finally, I would like to extend special thanks to those who have supported me in harder times during the past three years. Without providing specific names here, if you suspect that this may be for you, even just a little, then be assured that it indeed is.

Abstract

Current applications for humanoid robotics require autonomy in an environment specifically adapted to humans, and safe coexistence with people. Whole-body control is promising in this sense, having shown to successfully achieve locomotion and manipulation tasks. However, robustness remains an issue: whole-body controllers can still hardly cope with unexpected disturbances, with changes in working conditions, or with performing a variety of tasks, without human intervention. In this thesis, we explore how whole-body control approaches can be designed to address these issues. Based on whole-body control, contributions have been developed along three main axes: joint limit avoidance, automatic parameter tuning, and generalizing whole-body motions achieved by a controller. We first establish a whole-body torque-controller for the iCub, based on the stack-of-tasks approach and proposed feedback control laws in $SE(3)$. From there, we develop a novel, theoretically guaranteed joint limit avoidance technique for torque-control, through a parametrization of the feasible joint space. This technique allows the robot to remain compliant, while resisting external perturbations that push joints closer to their limits, as demonstrated with experiments in simulation and with the real robot. Then, we focus on the issue of automatically tuning parameters of the controller, in order to improve its behavior across different situations. We show that our approach for learning task priorities, combining domain randomization and carefully selected fitness functions, allows the successful transfer of results between platforms subjected to different working conditions. Following these results, we then propose a controller which allows for generic, complex whole-body motions through real-time teleoperation. This approach is notably verified on the robot to follow generic movements of the teleoperator while in double support, as well as to follow the teleoperator's upper-body movements while walking with footsteps adapted from the teleoperator's footsteps. The approaches proposed in this thesis therefore improve the capability of whole-body controllers to cope with external disturbances, different working conditions and generic whole-body motions.

Table of contents

List of figures	ix
List of tables	xii
1 Introduction	1
1.1 Motivation	4
1.2 Related work	5
1.2.1 Whole-body motion control of humanoid robots	5
1.2.2 Tuning of motion controllers	8
1.2.3 Joint limit avoidance of torque-controllers	10
1.2.4 Whole-body teleoperation	12
1.3 Contributions	14
1.4 Notation	16
1.5 Thesis outline	17
2 Optimization-based whole-body torque-control for humanoid robots	18
2.1 Modelling of floating-base systems	19
2.2 Optimization-based whole-body torque-control framework	21
2.2.1 Control input	22
2.2.2 Stack-of-tasks for whole-body torque-control	23
2.2.3 Stabilization of tasks	25

2.2.4	Whole-body control optimization problem	32
2.3	Implementation of optimization-based whole-body torque-controllers . .	33
2.3.1	Implementation of a strict tasks controller	34
2.3.2	Implementation of the soft tasks controller #1	36
2.3.3	Implementation of the soft tasks controller #2	39
2.4	Application to walking in place	42
2.4.1	Walking in place with the soft tasks controller #1	42
2.4.2	Walking in place with the soft tasks controller #2	50
2.4.3	Discussion	53
2.5	Conclusion	55
3	Joint limit avoidance for torque-control	57
3.1	Modelling of fixed-base systems	58
3.2	Classical torque-control techniques for fixed-base systems	59
3.3	Joint space parametrization	59
3.4	Joint space control with joint limit avoidance	61
3.4.1	Joint limit avoidance for fixed-base systems	62
3.4.2	Joint limit avoidance within a whole-body torque-controller . . .	65
3.5	Implementation for a fixed-base manipulator	65
3.5.1	Application of joint limit avoidance for the iCub leg	65
3.5.2	Discussion	73
3.6	Implementation within a whole-body torque-controller	74
3.6.1	Optimization-based controller	74
3.6.2	Application to walking in place	75
3.6.3	Discussion	79
3.7	Conclusion	80
4	Survey on parameter tuning for QP-based controllers	81

4.1	Description of the survey	83
4.2	Results of the survey	84
4.3	Conclusion	98
5	Learning task priorities of whole-body controllers	99
5.1	Constrained stochastic optimization	101
5.2	Framework for learning task priorities	107
5.3	Implementation for whole-body torque-control	110
5.3.1	Control problem formulation	110
5.3.2	Constraints on stochastic optimization	111
5.3.3	Fitness	111
5.3.4	Randomized conditions	112
5.4	Application to learning task priorities for walking in place	113
5.4.1	Training with the tethered iCub model	114
5.4.2	Testing with the tethered iCub model	115
5.4.3	Testing with the backpacked iCub model	116
5.4.4	Discussion	122
5.5	Conclusion	123
6	Teleoperation of generic whole-body motions	126
6.1	Whole-body velocity-control framework for teleoperation	129
6.1.1	Retargeting module	129
6.1.2	Finite state machine	134
6.1.3	Stack-of-tasks for whole-body velocity-control	136
6.1.4	QP controller	139
6.2	Applications of the whole-body teleoperation framework	141
6.2.1	Application to walking	142
6.2.2	Application to whole-body teleoperation	146

6.3 Conclusion	151
7 Conclusion	154
7.1 Whole-body balancing torque-control	154
7.2 Joint limit avoidance	155
7.3 Tuning parameters of whole-body controllers	155
7.4 Whole-body control for generic motions	156
7.5 Closing remarks	157
References	158
Appendix A Additional material for the soft tasks controller #1	169
A.1 Finite state machine for the soft tasks controller #1	169
A.2 Parameter values for the implementation in simulation	174
A.3 Parameter values for the implementation on the iCub	180
Appendix B Additional material for the soft tasks controller #2	186
B.1 Finite state machine for the soft tasks controller #2	186
B.2 Parameter values for the implementation in simulation	189
Appendix C Additional material for joint limit avoidance	192
C.1 Proof of lemma 3	192
Appendix D Additional material for the survey on QP controllers	194
D.1 Invitation to participate to the survey on QP-based controllers	195
D.2 Questionnaire of the survey on tuning of QP-based controllers	196
D.3 Answers to open-ended questions	203

List of figures

1.1	Examples of state of the art humanoid robots	2
1.2	Two different models of the iCub	3
1.3	Typical whole-body control architecture	6
1.4	Control architecture used for whole-body control in this thesis	16
2.1	Overview of the proposed method	22
2.2	Frames considered in the control framework	24
2.3	Finite state machine for soft tasks controller #1	38
2.4	Finite state machine for soft tasks controller #2	41
2.5	Snapshots of walking in place with the soft tasks controller #1	46
2.6	Position task trajectories for 1 stride, simulation experiments	47
2.7	Position task trajectories for 1 stride, real-world experiments	48
2.8	Orientation task errors for 1 stride, real-world experiments	49
2.9	Evolution of contact forces for 1 stride, real-world experiments	49
2.10	Evolution of joint torques for 1 stride, real-world experiments	49
2.11	Evolution of joint torques for 1 stride, soft tasks controller #2	51
2.12	Evolution of contact forces, soft tasks controller #2	52
2.13	Snapshots of walking in place with the soft tasks controller #2	52
2.14	Position task trajectories for 1 stride, soft tasks controller #2	53
3.1	iCub leg setup used for the experiments	66

3.2	Hip and knee joint trajectories and torque for experiment 1	68
3.3	Hip and knee joint trajectories and torque for experiment 2	70
3.4	Pictures of experiment 3	71
3.5	Hip joint trajectories and torques for experiment 3	72
3.6	Overview of the whole-body controller with joint limit avoidance	74
3.7	Behavior achieved under increasing external perturbations	77
3.8	CoM, elbow and shoulder trajectories	78
4.1	Demographic profile of the 35 respondents	92
4.2	Types of robots used with QP controllers by the respondents	93
4.3	Use and formulation of QP controllers by the respondents	93
4.4	Respondents' evaluation of the importance of tuning parameters	94
4.5	Time spent tuning parameters of QP controllers	95
4.6	Use of tools for tuning	95
4.7	Respondents' evaluation of how tedious tuning is	96
4.8	Importance of easing parameter tuning, according to respondents	96
4.9	Respondents' interest in eventual tools for tuning	96
5.1	Different iCub models performing the same whole-body motion	101
5.2	Effect of reducing the variance of the offspring distribution	104
5.3	Overview of the proposed method for learning task priorities	108
5.4	Typical ZMP trajectories obtained with the tethered iCub	117
5.5	Typical CoM and feet trajectories obtained for 6 strides	118
5.6	Typical base velocities obtained with the tethered iCub	119
5.7	Typical ground reaction at the right foot of the tethered iCub	119
5.8	Typical joint torques obtained with the tethered iCub	120
6.1	Typical whole-body control architecture	126

6.2	Control architecture used for whole-body teleoperation	127
6.3	Retargeting of human whole-body movements on the robot	128
6.4	Overview of the proposed method for whole-body teleoperation	130
6.5	Mapping between motion capture and robot joints	131
6.6	Hierarchical finite state machine for whole-body teleoperation	134
6.7	Snapshots of walking experiment 1	144
6.8	Snapshots of walking experiment 2	144
6.9	Snapshots of walking experiment 3	145
6.10	Teleoperation experiment 1: results of retargeting on the real robot . . .	147
6.11	Snapshots of teleoperation experiment 1	148
6.12	Feet motions achieved in simulated teleoperation experiments 1 and 2 .	148
6.13	Snapshots of teleoperation experiment 2	149
6.14	Snapshots of teleoperation experiment 3	151
D.1	Survey page 1	196
D.2	Survey page 2	197
D.3	Survey page 3	198
D.4	Survey page 4, part 1	199
D.5	Survey page 4, part 2	200
D.6	Survey page 4, part 3	201
D.7	Survey page 5	202

List of tables

3.1	Joint limits of the iCub	77
3.2	Forces applied on the right hand of the robot	77
4.3	References and links to source code indicated by respondents	97
5.1	Randomized set of conditions j	113
5.2	Summary of performed experiments and achieved results	121
6.1	Task weights used with the whole-body teloperation controller	141
6.2	Parameters of the finite state machine	142
A.1	Parameters for the stabilization of contact tasks	174
A.2	Task weights of the soft tasks controller #1 for the simulated robot	174
A.3	Parameters of the finite state machine	175
A.4	Proportional feedback gains for Cartesian tasks	176
A.5	Proportional feedback gains for the postural task	177
A.6	User-defined displacement of the Cartesian tasks for a stepping motion	178
A.7	User-defined joint positions for the postural task	179
A.8	Parameters for the stabilization of contact tasks	180
A.9	Task weights of the soft tasks controller #1 for the real robot	180
A.10	Parameters of the finite state machine	181
A.11	Proportional feedback gains for Cartesian tasks	182

A.12 Proportional feedback gains for the postural task	183
A.13 User-defined displacement of the Cartesian tasks for a stepping motion	184
A.14 User-defined joint positions for the postural task	185
B.1 Parameters for the stabilization of contact tasks	189
B.2 Parameters of the finite state machine	190
B.3 Task weights of the soft tasks controller #2 for the simulated robot . .	190
B.4 Proportional and derivative feedback gains for Cartesian tasks	191
B.5 Proportional and derivative feedback gains for the postural task	191

Chapter 1

Introduction

Research in humanoid robotics has seen massive advances in recent years, having received significant attention both from society and the research sector. Humanoid robots are expected to be highly involved in many unprecedented applications over the next decades, and to strongly impact everyday life. They shall find a place not only in research laboratories, but at home, in the workplace, as well as in environments unsuitable for humans. In any application, we anticipate humanoid robots to have the ability to autonomously function in the real-world, safely interacting with complex dynamic environments and coexisting with its inhabitants.

Having said that, at this date, the capabilities of current robots are generally limited to basic tasks in controlled environments, for which physical interactions are restricted to the task being performed. For example, lifting and carrying objects, as well as dynamic walking have been achieved, albeit not exactly reproducing the autonomy with which humans can perform these actions. Much research is still required, in order to reach capacities approaching those of humans, with humanoid robots.

Related research problems then span over several different fields, such as (but not limited to) perception, motion control, motion planning, manipulation, cognition, as well as human-robot interaction, in order to deal with the real-world.

A significant number of outstanding humanoid robots have been developed thus far for research purposes, each one designed with the intention of tackling one or several of the research problems mentioned above. Without naming them all, famous state of the art examples include the Atlas robot [Boston Dynamics, 2018], which was developed for highly dynamic locomotion, as well as lifting and carrying objects. HRP series

robots [Kawada Industries, 2018] are designed to collaborate with humans, allowing for advanced research on motion control and manipulation. Darwin-OP [Robotis Co., 2018] is a small humanoid robot that allows research on motion control, planning and perception. Nao [SoftBank Robotics, 2018] is another small humanoid robot that offers similar capabilities, while being designed to interact with people. Each of them is illustrated in figure 1.1.



Figure 1.1 Examples of state of the art humanoid robots. From left to right: Boston Dynamic Atlas, Robotis Darwin-OP, SoftBank Robotics Nao, Kawada Industries HRP-4

Another noteworthy robot is the iCub [Metta et al., 2008], a child-sized humanoid robot developed at the *iCub Facility* of the *Fondazione Istituto Italiano di Tecnologia*. It is designed as a research platform for robotics, AI and cognitive science, which makes it a prime choice for carrying out research across many different fields, allowing to tackle each of the different research problems related to humanoid robotics mentioned above and more. It is notably used as the main robotic platform for the experiments carried out in this thesis, and for this reason deserves a more extensive description.

The iCub features a very high degree of mobility with its 53 degrees-of-freedom (DOFs), including 9 DOFs in each hand and 3 for the eyes. A total of 32 DOFs can be accounted for, to control the motion of the head, torso, arms and legs. Each of these DOFs is actuated with brushless DC electric motors, with harmonic drive transmission [Parmiggiani et al., 2012]. As such, these actuators are suitable for position-, velocity- and torque-control.

Beyond cameras in the eyes and microphones, the iCub also exhibits advanced sensing capabilities. Forces exchanged between the robot and the environment can be evaluated through the estimation of internal joint torques and external wrenches [Fu-

magalli et al., 2010; Traversaro et al., 2015]. This is made possible with a set of six-axis force-torque (F/T) sensors installed in each arm, leg and foot of the robot. In order to determine the location and distribution of contact forces applied on the robot, most of the body of the iCub is covered with tactile sensors enclosed within a fabric artificial “skin”. Also, an inertial measurement unit is installed in the head of the robot, equipped with magnetometer, accelerometer and gyroscope. This equipment provides inertial sensing, useful for calibration of the robot [Guedelha et al., 2016]. Additional sensors such as accelerometers and gyroscopes are distributed within its body.

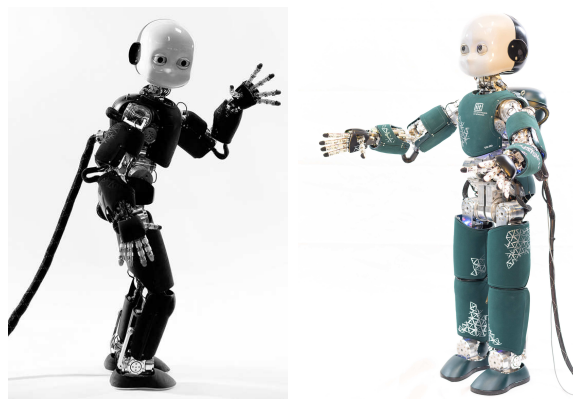


Figure 1.2 Two different models of the iCub. On the left: with wired power supply through the back. On the right: with a battery pack on the back (power supply can still be provided by a connection on the battery pack).

Thus, the iCub is highly suitable for complex tasks involving its entire body, such as walking. The development of control methods for achieving such motions, in a way which is safe for interaction with the real-world, and allows for an increased autonomy of the robot, are investigated in this thesis.

The motivations leading to the work presented in this thesis are described in the next section. An overview of the related work, in section 1.2, then allows to gain an understanding of the current state of the art, giving grounds for the contributions we propose in section 1.3. Finally, before moving from this chapter to the core of the thesis, notation is introduced, as well as an outline of the thesis, describing the organization of the remaining chapters.

1.1 Motivation

Robotic applications which are currently envisioned require not only that robots function autonomously in an environment specifically adapted to human capabilities, but also that robots safely coexist with humans.

Nevertheless, the design of useful, efficient and safe motion controllers for humanoid platforms is highly challenging, especially for applications involving interaction with the environment. As a matter of fact, physical interactions have a significant influence on the stability and balance of a humanoid robot. Furthermore, a great number of issues can stem from the complexity of the system and its control framework, in order to generate physically consistent behaviors. The development of robust compliant balancing controllers is therefore required.

In this respect, **whole-body control** is a promising research direction, aiming to define rules on robot motion that guarantee the execution of tasks involving the entire body of redundant, floating-base robots subjected to interactions with the environment. So far, whole-body controllers have shown to successfully achieve motion control of humanoid robots. In addition, whole-body torque-controllers can offer compliance and allow the control of physical interactions with the environment [Ott et al., 2011; Saab et al., 2013].

This thesis focuses on developing control methods for humanoid robots, within the frame of whole-body control. Although locomotion and manipulation tasks have shown to be successfully achieved using this approach, whole-body controllers can still hardly cope with unexpected disturbances, with performing a variety of tasks (as opposed to achieving a single task), or with changes in working conditions, without human intervention. In order to achieve a higher level of autonomy, developing controllers for **robustness** as we define it below, appears to be fundamental.

Definition 1 *In this thesis, we use the word robustness to indicate the ability of a controller to cope with perturbations, different tasks or changes in working conditions.*

It is considered an equivalent of coping capacities, the ability of a system to respond to and recover from the effects of stress or perturbations that have the potential to alter the function of the system. This expression is used in the title, to avoid any confusion of the term robustness in readers from different fields.

In this thesis, we explore how whole-body control approaches can be made more robust, to safely achieve complex motions of a humanoid robot. The controllers developed here shall be based on the concept of the **stack-of-tasks** [Mansard et al., 2009], which provides a hierarchical framework adapted for whole-body control, attempting the simultaneous stabilization of several elementary tasks. While it allows for flexibility in the definition of a controller, it also requires a certain amount of tuning, in a context where the controller is also sensitive to perturbations, as well as imprecisions in the model, estimation and measurements. As a result, transferring results from simulation to a real-world platform can be challenging. These issues can be addressed by developing methods to automatically tune controller parameters and to increase the robustness of controllers.

1.2 Related work

The field of humanoid robotics has seen impressive developments recently, although this section will concentrate only on a few chosen areas. More precisely, this thesis is rooted in whole-body motion control of humanoid robots, and branches off towards the subjects of joint limit avoidance, parameter tuning and teleoperation, in the search for measures which may increase the robustness of controllers. The following subsections shall review relevant literature on each of these subjects.

1.2.1 Whole-body motion control of humanoid robots

Whole-body control has been the subject of extensive research so far, and various types of controllers have been investigated specifically for biped robots. **Fixed-base** controllers considering a robot as a manipulator attached to the ground, either passivity-based [Hyon et al., 2007], bio-inspired [Heremans et al., 2016] or momentum-based [Ott et al., 2011], have shown to ensure stable behavior. However, in order to achieve higher mobility, modeling a robot as a **floating-base** system, in which no link is fixed with respect to an inertial frame, is generally more relevant.

Typically, the whole-body control problem is tackled following a control architecture such as discussed in [Romualdi et al., 2018]. It has been adapted here in figure 1.3, where it shows to be particularly targeted for walking. Given a desired task for the robot, reference trajectories of the robot are optimized, for example desired footsteps

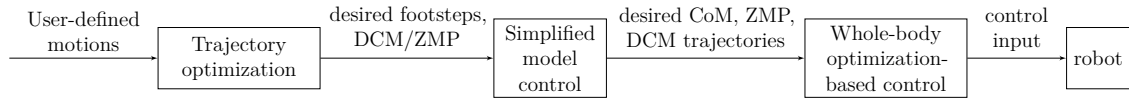


Figure 1.3 Typical whole-body control architecture for humanoid robots

are computed using a footstep planner. Taking the footsteps as input, kinematically feasible trajectories of the robot are computed, using a simplified robot model such as the linear inverted pendulum model, and optionally calling on the use of model predictive or receding horizon controllers. Finally, a whole-body controller based on optimization ensures the tracking of the computed trajectories on the robot.

A highly effective solution to the whole-body control problem is to decompose a complex behavior into several elementary tasks, typically framed as a stack-of-tasks [Ivaldi et al., 2016; Mansard et al., 2009]. The controller is then aiming at the achievement of elementary control objectives organized in a hierarchical structure.

In such a framework, some kind of **prioritization** among tasks must be provided, generally either a “strict” or a “soft” task hierarchy. With strict prioritization strategies (such as a traditional stack-of-tasks), a fixed task hierarchy is assured by geometrical conditions such as a null space task projector [Nava et al., 2016; Saab et al., 2013], or by the use of optimization strategies to compute control actions [Dafarra et al., 2018]. Conversely, prioritization in a soft task hierarchy (based on a weighted combination of tasks), can be achieved by assigning each task a weight defining its relative importance [Otani et al., 2018; Salini et al., 2011]. Since it involves concurrently solving a certain number of tasks on end-effectors, joints or relevant body parts, at each control step, the whole-body control problem is most often formulated as an optimization problem.

Quadratic programming (QP), the optimization problem of finding a vector that minimizes a quadratic function subject to bounds, linear equality, and inequality constraints, has shown to be advantageous in terms of flexibility, robustness and speed. For this reason, QP is often used to solve whole-body control optimization problems [Dafarra et al., 2018; Kuindersma et al., 2014; Nava et al., 2016; Otani et al., 2018; Righetti and Schaal, 2012].

From there, when compliance and physical interactions matter (e.g. safe human-robot physical interaction), torque-control, in opposition to position-control, is usually preferred. However, achieving precise motions with torque-control can be more challenging than with, for example, position-control. By extension, so is the problem

of achieving whole-body motions that are robust and compliant to interaction with the environment. For this purpose, suitable methods need to be developed in order to ensure balance and stability of the robot, ensuring not only that the robot keeps balance when external forces are applied, but also that it softly yields under forces, allowing for safer physical interaction.

Stability of a floating-base, torque-controlled humanoid robot is then typically ensured with momentum-based control strategies integrated within a stack-of-tasks [Herzog et al., 2014; Nava et al., 2016; Pucci et al., 2016b; Stephens and Atkeson, 2010], enabling the stabilization of desired center of mass (CoM) and contact wrenches. Control objectives are generally formulated into optimization problems, to be solved recursively for each layer of the hierarchy and requiring the projection of low-priority task Jacobians in the null-space of higher-priority task Jacobians. Such strategies have shown to allow specific motions to be achieved, through the definition of additional Cartesian or postural tasks. In [Lee and Goswami, 2012] for example, desired CoM and feet trajectories are taken into account for walking.

In the case of humanoid robots, the generation of a centroidal momentum often relies on applying a torque about the center of mass, for instance by swinging the arms or bending quickly at the hips [Stephens and Atkeson, 2010]. Although such behavior may enable a robot to maintain balance, the quick motions may not be ideal when interacting with humans. Moreover, joint limits of humanoid robots impose restrictions on movement. As a result, a desired angular momentum can typically be generated for short periods only, which may leave something to be desired, for example when subjected to continuous pushes. Alternate formulations of whole-body controllers may then be needed.

For instance, a framework enabling sequences of dynamic tasks for a floating-base torque-controlled humanoid robot has been developed in [Salini et al., 2011], using a soft hierarchy of Cartesian and postural tasks. In this particular implementation, an impedance controller is used to induce a desired behavior with respect to contacts with the environment, while a controller based on the approximation of the zero moment point is used to perform balancing and walking tasks. Ultimately, with this method, important discontinuities are introduced when switching between states, for which coping methods have to be proposed. Also, while stack-of-tasks hierarchization strategies are shown to allow for flexibility in view of achieving tasks, defining the priorities of each task can be considered a complex problem.

1.2.2 Tuning of motion controllers

When working with controllers based on a stack-of-tasks, the priorities of tasks are usually designed *a priori* by experts, then manually tuned to adjust the task ordering, timing, transitions, *etc.* However, this can be a relatively tedious operation, depending on the complexity of the system and the tasks.

As a matter of fact, coordination of multiple tasks for whole-body control of floating-base platforms can be particularly hard when it involves keeping balance and navigating an environment, while fulfilling other desired activities such as torso and upper limb movements for manipulation or obstacle avoidance. The level of complexity that characterizes some humanoid application scenarios can deeply hinder the experts' capability to provide an effective solution for the multi-task coordination problem.

A recent line of research seeks to tackle the issue, aiming to automatically learn whole-body task priorities, while satisfying a problem's constraints [Dehio et al., 2015; Ha and Liu, 2016; Modugno et al., 2016a; Su et al., 2018]. For instance in [Dehio et al., 2015], task coefficients are learned for controlling a single arm, allowing the transfer of acquired knowledge to different tasks. In this work however, the balance of the humanoid platform is ignored. In [Ha and Liu, 2016], an evolutionary algorithm, using parametrized motor skills to learn policies, achieves task generalization. Then, in [Modugno et al., 2016a; Su et al., 2018], task priorities guaranteeing the non violation of system constraints are learned for bimanual motions.

Since these methods require several repetitions of the same experiments (rollouts) in order to find a viable solution, a feasible way to speed up the tuning process of whole-body controllers is to optimize parameters through learning, in simulation. This approach is particularly advantageous, since the random exploration used by learning algorithms could bring disastrous results if used directly on the hardware, and the considerable number of iterations required to find an optimal solution can be problematic if performed on the real robot. For these reasons, training is preferably performed in simulation. However, inherent differences between simulated and real robots can render an optimal solution *untransferable* from one to the other. Researchers usually refer to this issue as the **reality gap** problem, which needs to be accounted for, in order to achieve automatic parameter tuning.

Solutions to this problem have recently been addressed by trial-and-error algorithms [Cully et al., 2015; Spitz et al., 2017]. In [Cully et al., 2015], prior knowledge from

simulated robot behaviors is exploited to find acceptable behaviors on the real robot, in few trials. In [Spitz et al., 2017], a trial-and-error learning algorithm is used to encourage exploration of the task space of the QP controller, allowing adaptation to inaccurate models, also in few trials.

Since QP solvers often allow for constraints relaxation, strict constraints satisfaction is not always ensured by the frameworks presented above, even though such a guarantee could potentially allow for a better generalization of learned solutions. In [Modugno et al., 2016a], strict constraints fulfillment is guaranteed by the use of a constrained extension of (1+1)CMA-ES [Arnold and Hansen, 2012]. Nonetheless, transferring acquired knowledge from simulation to the real robot (closely similar to its simulation), have not shown to fully achieve the desired behavior. This result implies that constraints satisfaction is beneficial, but not enough to achieve transferability: solutions which are *robust* rather than *optimal* are needed, in order to achieve a better generalization.

As a way to achieve this, [Del Prete and Mansard, 2016] have proposed to improve the robustness of task space inverse dynamics by modeling uncertainties in the joint torques of humanoid robots.

Looking for robust solutions is also central to **transfer learning** and **domain adaptation**, which seek to exploit previously acquired knowledge to allow a model trained in one task or domain to be re-purposed on a second related one [Pan and Yang, 2010]. It can for instance take the shape of learning from simulation. In fact, data gathering can be significantly simplified when performing a learning procedure in simulation, but there is no guarantee that the acquired knowledge will be effective in the real world. For example, optimal control and movement primitives are combined in [Clever et al., 2017] to find solutions which can be easily deployed on the real robot, but they strongly rely on the accuracy of the simulated model in order to ensure the transferability of solutions. In [Silvério et al., 2018], operational and configuration space control task priorities are learned from suitable robot configurations provided by an expert, given possible task hierarchies. This approach allows the transfer of results between different robots models in simulation, but relies on significant input from an expert when several tasks are involved.

Automatic tuning of floating-base, whole-body controllers, allowing an easier transfer of results, is therefore still an open issue.

Furthermore, in the case of torque-controllers, even optimally adjusted parameters may not ensure constraints on joint limits to be satisfied at all times. In case of external perturbations, for instance, joint limits may not be successfully avoided, and this problem therefore remains to be addressed.

1.2.3 Joint limit avoidance of torque-controllers

Nonlinear control of unconstrained fully-actuated manipulators is no longer a theoretically challenging problem for the control community. Algorithms based on position-, velocity-, and torque-control have long been analysed with back-stepping and feedback linearization tools, and have proved to be effective in numerous applications, e.g. [Isenberg et al., 2010; Luo et al., 2013; Mason et al., 2014]. The control problem associated with robotic manipulators, however, rapidly becomes challenging when motion and actuation constraints must be satisfied.

As a matter of fact, the problem of ensuring joint limit avoidance is not new to the robotics community. For instance, a variety of methods have been developed for avoiding joint limits in path planning, such as weighted least norm solutions [Chan and Dubey, 1995], damped least square solution of inverse kinematics [Na et al., 2008], Lyapunov-based methods [Chen and Guo, 2006], neural networks [Assal et al., 2005; Zhang et al., 2003], or using a time-varying weight matrix in the inverse kinematics [Kermorgant and Chaumette, 2011]. Nonetheless, generating reference trajectories that satisfy the physical limits does not imply that the joint positions will evolve within these limits.

In the case of redundant manipulators, on-line joint limit avoidance may be attempted by using the stack-of-tasks approach. In fact, the control objective associated with redundant manipulators is usually the stabilisation of the robot end-effector, and the solutions associated with this task may not be unique.

One can exploit redundancy by defining a secondary low-priority task, in charge of keeping the joints away from limits and acting onto the null space of the main task [Fiacco and Luca, 2013; Fukumoto et al., 2004; Jamone et al., 2013]. One of the main drawbacks of this approach is that there is no theoretical guarantee that the joint evolutions always belong to the feasible domain. Also, the two-layer prioritization may lead to undesired robot behavior, due to the projection onto the null space of the control action in charge of ensuring joint limit avoidance.

Among the most widely used methods for joint limit avoidance of redundant manipulators is the gradient projection-based technique [Chen and Liu, 2002; Ito et al., 2010; Liegeois, 1977; Marey and Chaumette, 2010]. This approach defines a criterion, such as a function maximizing the distance between joint positions and their limits. The gradient of this function is then projected onto the null space projection matrix of the Jacobian, allowing to move the joints away from limits without affecting the end-effector position. This method has a few drawbacks, as it does not guarantee minimization of the criterion for each individual joint, and additional coefficients need to be used to properly tune the self-motion magnitude.

Control approaches based on a barrier function [Ngo and Mahony, 2006; Prajna and Jadbabaie, 2004; Tee et al., 2009; Wieland and Allgöwer, 2007] may also allow for joint limit avoidance of redundant position-controlled manipulators, as proposed in [Atawneh et al., 2016].

Instead, [Prete, 2018] presents an algorithm to estimate bounds on joint accelerations, that, if respected, ensure that joint position, velocity and torque limits can be avoided in future joint trajectories of a torque-controlled manipulator.

In humanoid whole-body motion control, unilateral virtual springs and spring-dampers have been implemented around joint limits to generate torques repelling from the bounds [Dietrich et al., 2011; Moro et al., 2013]. Another possibility for humanoid robots is to solve whole-body motion as an optimization problem with inequality constraints corresponding to joint limits [Feng et al., 2014; Herzog et al., 2014; Hopkins et al., 2015; Tassa et al., 2014]. In all of these works, however, the theoretical guarantee of the stability and convergence properties associated with the evolution of the system is still missing.

Furthermore, in the case of robots co-existing and collaborating with humans, it is not only important to produce motions which avoid joint limits, but controllers that can actually avoid joint limits while coping with perturbations, such as unknown external forces applied on the robot.

What can also be highly beneficial for human-robot collaboration, is to produce motions of a humanoid robot which are easily understandable to humans. For this purpose, controllers that allow to keep balance while producing human-like whole-body motions also need to be investigated.

1.2.4 Whole-body teleoperation

To this day, the design of efficient and safe controllers for humanoid robots remains challenging [Ivaldi et al., 2016], especially for applications involving locomotion and manipulation, as well as interaction with the environment [Kuindersma et al., 2016] and with human partners [Otani et al., 2018; Romano et al., 2018].

As mentioned above, the whole-body control problem is most often formulated as QP [Otani et al., 2018]. It involves concurrently solving a certain number of tasks on end-effectors, joints or relevant body parts, at each control step. Therefore, one has to specify the target trajectories to be performed for each task. The design of such trajectories is often time-consuming. In many situations, for instance when complex manipulations and locomotions are involved, trajectories are manually tuned by an expert [Norton et al., 2017], or may be optimized offline [Modugno et al., 2017].

However, in the case of robots co-existing and collaborating with humans, it is not only important to produce motions which are feasible and efficient, but they should also be **legible**, i.e., easily understandable for the people working with the robot [Dragan et al., 2013]. This requirement is particularly important for humanoid robots, and can be translated into producing human-like motions, either by design, through optimization, or by resorting to whole-body human imitation [Ott et al., 2008].

The latter solution consists in reproducing the motion of a human operator onto the robot. Such techniques, known as **motion retargeting** (or **teleoperation**, in the real-time case) could then be used to demonstrate human-like movements to the robot, as a way to achieve complex behaviors such as loco-manipulation and collaborative policies. The ability to retarget complex human movements in real-time by imitating the operator has countless applications, ranging from enabling human demonstrations for complex tasks (eliminating the manual tuning or expert tuning problem), to enabling remote control of robots in dangerous, extreme or disaster response scenarios.

Several motion retargeting approaches proposed in the literature are implemented offline, allowing to retarget challenging multi-contact motions that affect the distribution of the weight of the robot. For instance, [Yamane and Nakamura, 2003] propose a filter that converts motion capture data into feasible joint trajectories for a human figure, provided a list of constraints. This approach allows for successful contact switching, but involves extensive parameter tuning for each desired behavior of the robot. In later work [Yamane and Hodgins, 2009], a framework combining a balance controller and a

controller tracking joint angles from motion capture data, has shown to allow offline whole-body motion retargeting onto a humanoid robot in double support conditions, but again requiring careful parameter tuning.

Instead, in [Ayusawa and Yoshida, 2017], an optimization problem is defined in order to find optimal generalized coordinates and properties of a human model and of a robot model, under constraints that ensure balance and limitations in the movement of the robot. This method is demonstrated for retargeting whole-body motions from offline motion capture in double support conditions. In [Kanajar et al., 2017; Otani and Bouyarmane, 2017], QP frameworks are used for retargeting complex motions such as climbing over an obstacle or box lifting. In these works, multi-contact motions are retargeted offline onto the robot, given the pre-processed sequence of contact events.

Notably, the offline formulation makes it possible to process and clean up the motion capture data, as well as to specifically tune the retargeted robot trajectories [Fava et al., 2016; Otani and Bouyarmane, 2017; Yamane and Hodgins, 2009; Yamane and Nakamura, 2003], as a way to fit the demonstrated motions more closely. However, such facilitations cannot be used in real-time. For this reason, walking represents one of the most highly challenging multi-contact motions to have been addressed so far in the teleoperation scenario.

Therefore, first teleoperation works have been concerned only with upper-body movements [Brygo et al., 2014; Dariush et al., 2009; Fritsche et al., 2015; Shin and Kim, 2010]. Whole-body movements then started to be included in teleoperation frameworks, with joysticks being used to control walking and end effector motions, such as in [Sian et al., 2002; Stilman et al., 2008]. Motion capture systems, having the potential to allow for more complex motions, have then been used for teleoperation. For example, [Koenemann et al., 2014] proposes a method based on a simplified model of the human, to allow the retargeting of transitions between double and single stance phases onto the NAO robot. In [Elobaid et al., 2018], joy-pads are used to teleoperate the arms of the iCub robot. Furthermore, the walking of the robot is teleoperated separately (but not simultaneously) with an immersive scenario using a virtual reality headset and a walking platform. Instead, in [Hu et al., 2014], the authors teleoperate exclusively the walking onto TORO, considering the human footsteps and leg joints configuration.

However, in such frameworks, few parts of the robot body are directly taken into consideration for the retargeting, in contrast to whole-body retargeting. Typically, the position of the end effectors (EE) of the human operator are properly scaled to

match the dimensions of robot links, then retargeted onto the robot by using an inverse kinematics (IK) formulation which generates feasible configurations within joints limits.

Indeed in the literature, work addressing whole-body motion retargeting is generally based on the consideration that human motions are assigned locally and independently. Hence, the retargeting of leg motions, which is more involved with balance, is treated separately from upper-body motion retargeting, which is more involved with manipulation tasks. For instance, tracking leg motions may be neglected, in order to use the lower-body specifically for keeping balance, as in [Ott et al., 2008], where the teleoperation problem is split between retargeting the position of human motion capture markers on the upper-body joints, while a separate balancing algorithm controls the lower-body (legs and hips). This concept is also applied in [Kim et al., 2013], in order to separately retarget upper-body motions and walking.

In these works, the classical motion retargeting approaches are not applicable to the robot body parts which are the most involved in a dynamic movement, e.g. legs and feet, particularly when footsteps and contact transitions are involved. For this reason, so far, the teleoperation of upper-body manipulation tasks has generally been dissociated from challenging lower-body motions such as walking.

Conversely, real-time teleoperation of whole-body movements is achieved in [Ishiguro et al., 2017, 2018; Penco et al., 2018]. In [Ishiguro et al., 2017, 2018], reference joint trajectories are computed from EE motion capture data, using IK and a stabilizer to ensure balance. As noted by the authors however, IK can limit the achievement of movements approaching human speed. In [Penco et al., 2018] instead, a retargeting framework using a dynamic filter and a QP-based controller is proposed, in order to maintain balance while retargeting whole-body motions, but this method has been verified only for double support conditions.

1.3 Contributions

In response to the gaps in the literature evidenced in section 1.2, the present thesis offers the contributions introduced in the following paragraphs.

Contributions are tested using an optimization-based whole-body control framework, developed as the base of this work. The framework established in this thesis is formulated to be readily available for eventual dynamic locomotion tasks, and is rather

standard with respect to state of the art whole-body controllers. In spite of that, it notably has the modest contribution of defining **control laws in $SE(3)$** , which circumvent the singularities typically introduced by Euler parameters in the control of orientation tasks. It is used in [Dafarra et al., 2018], as part of a control architecture developed for torque-controlled walking.

We then investigate a **joint limit avoidance approach** which can cope with external perturbations, and for which convergence and stability can be theoretically proven. A novel solution is proposed in the form of feedback control laws based on a parametrization of the feasible joint space, as published in [Charbonneau et al., 2016].

Following this work, we dive into the subject of tuning for controllers based on quadratic programming. Since information on the subject is currently limited in the literature, we have collected **data regarding the effort generally spent tuning QP-based controllers** through a survey distributed to researchers working in related fields. This information brings to light an interest of the community in the development of tools for tuning QP-based controllers.

As a response, tuning methods which allow for an easier deployment of controllers, and for transferability of the results achieved in simulation, are proposed. They consist in a framework developed for **automatically learning task priorities** while encouraging a higher robustness of the controller towards external perturbations and different working conditions, as published in [Charbonneau et al., 2018].

Finally, a **whole-body control framework for teleoperation** is proposed, in order to allow simultaneous upper-body and lower-body movements. A controller is developed to ensure robustness to generic reference trajectories, allowing the robot to keep balance while performing whole-body movements retargeted from a human operator to the robot. This work has been submitted for publication at a peer reviewed international conference as [Charbonneau et al., 2019].

Note that for our purpose, in this thesis we simplify the architecture of figure 1.3, concentrating on the last block, as shown in figure 1.4.

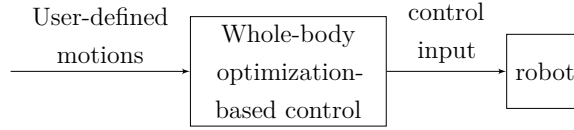


Figure 1.4 The control architecture used for whole-body control in this thesis. The user defines trajectories for the robot (e.g. feet, CoM, joint trajectories), which are directly used by the whole-body optimization-based controller to compute the control input (e.g. contact forces and joint torques) that allows the robot to achieve the desired motion.

1.4 Notation

The following notation is used throughout the thesis.

- The set of real numbers is denoted by \mathbb{R} .
- $1_n \in \mathbb{R}^{n \times n}$ denotes the identity matrix of dimension n .
- $0_{n \times m} \in \mathbb{R}^{n \times m}$ is the zero matrix of dimension $n \times m$.
- The \cdot operator denotes the inner product of vectors in \mathbb{R}^3 .
- The transpose operator is denoted by $(\cdot)^\top$.
- The hat operator denoted by $(\cdot)^\wedge$ is the skew-symmetric operator associated with the cross product in \mathbb{R}^3 .
- The vee operator denoted by $(\cdot)^\vee$ is the inverse of the $(\cdot)^\wedge$ operation, transforming a skew-symmetric matrix in $\mathbb{R}^{3 \times 3}$ into a vector in \mathbb{R}^3 .
- The skew(\cdot) operator extracts the skew-symmetric matrix $\text{skew}(A) = \frac{1}{2}(A - A^\top)$ for any matrix $A \in \mathbb{R}^3$.
- The $\text{tr}(\cdot)$ operator denotes the trace operator over square matrices.
- The Euclidean norm of a vector of coordinates $v \in \mathbb{R}^n$ is denoted by $|v|$.
- Given a time function $f(t) \in \mathbb{R}^n$, its first- and second-order time derivatives are denoted by $\dot{f}(t)$ and $\ddot{f}(t)$, respectively.
- ${}^A R_B \in SO(3)$ and ${}^A T_B \in SE(3)$ denote the rotation and transformation matrices which transform a vector expressed in the B frame into a vector expressed in the A frame.

- The Moore-Penrose pseudoinverse is denoted by $(\cdot)^\dagger$.
- The tilde operator denoted by $(\tilde{\cdot})$ denotes the expression $(\cdot) - (\cdot)_d$, i.e. the difference between the value of a variable (\cdot) and its desired value $(\cdot)_d$.

1.5 Thesis outline

The remaining of the thesis is organized as follows.

Chapter 2 introduces whole-body control frameworks for torque-controlled humanoid robots, based on quadratic programming. The controllers are based on the stack-of-tasks approach, with an emphasis on Cartesian and postural tasks. This chapter serves as the foundation of the thesis, on top of which the rest of the chapters are built.

Then, chapter 3 introduces novel control laws to ensure joint limit avoidance, adapted for torque-control. These control laws are based on parametrization of the feasible joint space, making joint limit avoidance an intrinsic property of the controller. The proposed approach is shown to be successfully applied to the control of a fixed-base manipulator, as well as the whole-body control of a humanoid robot.

Following the observation that tuning takes an important role in the development of our whole-body controllers, a survey on the subject of parameter tuning for QP-based controllers has been developed in order to investigate if this is a general issue in the community. Chapter 4 presents the survey and its results, which encourage us to propose, in chapter 5, a new automatic tuning method for adjusting task priorities of whole-body controllers. We demonstrate the effectiveness of the approach by transferring results between different robot models, without re-tuning parameters.

Next, building on the results obtained in the previous chapter, chapter 6 presents a whole-body control framework targeting the achievement of generic whole-body trajectories. A retargeting module is proposed, in order to define robot trajectories from human motions. When coupled to a finite state machine to define center of mass and feet trajectories, the proposed method is shown to allow the achievement of generic footsteps, as well as simultaneous real-time upper-body teleoperation and walking of the robot.

Finally, chapter 7 wraps up the thesis and provides some closing remarks.

Chapter 2

Optimization-based whole-body torque-control for humanoid robots

As this thesis is concerned with developing whole-body control methods to increase the autonomy of humanoid robots and their capacity to safely interact with people and the environment, the present chapter introduces the foundation on which the rest of the thesis is built.

Section 2.1 first acts as a recall on the modeling of floating-base robots for whole-body control. Section 2.2 then lays out a general framework for optimization-based whole-body torque-control of humanoid robots. It is based on the stack-of-tasks approach, and formulated to be readily available for eventual dynamic locomotion tasks. While recalling techniques for task stabilization, the formulation of a control policy in $SE(3)$ is notably proposed, as an alternative to the typical control policies which rely on Euler parametrization for orientation tasks.

Defined as an optimization problem, the controller can be solved through QP in order to obtain control inputs for the robot. Cost functions are defined such that a single optimization problem is solved, for all tasks of the stack-of-tasks. Therefore, with this formulation, there is no need to solve lower-priority tasks projected onto the null-space of higher priority tasks through a series of optimization problems.

With this framework, various formulations of the control problem may be defined, using different combinations of tasks and hierarchization strategies. Two main ways to define hierarchies between tasks can be defined: (i) “strict” prioritization, which takes higher priority tasks as constraints to lower priority tasks, and (ii) “soft” prioritization,

for which a weight is attributed to each task according to its priority, allowing to achieve a tradeoff between tasks. From there, strict and soft prioritization strategies of the different tasks can be combined together, in order to formulate the control problem. Furthermore, various definitions of a finite state machine used with the controller and the associated parameters may also be chosen, according to the application.

A few main iterations of this framework are presented in section 2.3, defined with different tasks, finite state machine and parameters. Discussing their differences allows to explore how changes may affect the results achieved with the control framework, as presented in section 2.4. The whole-body control framework has been implemented for the iCub, allowing to experimentally validate its performance.

2.1 Modelling of floating-base systems

For the purpose of whole-body control, it is assumed that the robot is a multibody system composed of $n + 1$ rigid bodies, called links, connected by n joints with 1 DOF each. If one of the links has a constant pose with respect to an inertial frame \mathcal{I} , then this fixed link is referred to as the base, and the multibody system is considered as *fixed-base*. However, if none of the links has an *a priori* constant pose with respect to the inertial frame \mathcal{I} , the system is considered free-floating, or *floating-base*. In this case, the system is subjected to contact constraints, and it can be modelled as follows.

The configuration space of the robot can be characterized by the position and orientation of a frame attached to a link of the robot (called base frame \mathcal{B}) and the joint configurations. Thus, the configuration space is defined by

$$\mathbb{Q} = \mathbb{R}^3 \times SO(3) \times \mathbb{R}^n \quad (2.1)$$

An element of \mathbb{Q} is then a triplet

$$q = (\mathcal{I}p_{\mathcal{B}}, \mathcal{I}R_{\mathcal{B}}, s) \quad (2.2)$$

where $(\mathcal{I}p_{\mathcal{B}}, \mathcal{I}R_{\mathcal{B}})$ denotes the origin and orientation of the base frame expressed in the inertial frame, and s denotes the joint angles.

It is possible to define an operation associated with the set \mathbb{Q} such that this set is a group. Given two elements q and ρ of the configuration space, the set \mathbb{Q} is a group under the following operation.

$$q \cdot \rho = (p_q + p_\rho, R_q R_\rho, s_q + s_\rho) \quad (2.3)$$

Furthermore, one easily shows that \mathbb{Q} is a Lie group. Then, the velocity of the multibody system can be characterized by the algebra \mathbb{V} of \mathbb{Q} defined by $\mathbb{V} = \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^n$. An element of \mathbb{V} is then a triplet

$$\nu = (\mathcal{I}\dot{p}_{\mathcal{B}}, \mathcal{I}\omega_{\mathcal{B}}, \dot{s}) = (\mathbf{v}_{\mathcal{B}}, \dot{s}) \quad (2.4)$$

where $\mathcal{I}\omega_{\mathcal{B}}$ is the angular velocity of the base frame expressed with respect to the inertial frame, i.e. $\mathcal{I}\dot{R}_{\mathcal{B}} = S(\mathcal{I}\omega_{\mathcal{B}})\mathcal{I}R_{\mathcal{B}}$.

We also assume that the robot is interacting with the environment, exchanging n_c distinct wrenches¹. The application of the Euler-Poincaré formalism [Marsden and Ratiu, 2010, chapter 13.5] to the multibody system yields the following equations of motion:

$$M(q)\dot{\nu} + h(q, \nu) = \zeta\tau + \sum_{k=1}^{n_c} J_{c_k}^\top f_k \quad (2.5)$$

where $M \in \mathbb{R}^{n+6 \times n+6}$ is the mass matrix, $h \in \mathbb{R}^{n+6}$ is the bias vector of Coriolis and gravity terms, $\tau \in \mathbb{R}^n$ is a vector representing the actuation joint torques, $\zeta = (0_{n \times 6}, 1_n)^\top$ is a selector matrix, and $f_k \in \mathbb{R}^6$ denotes the k -th external wrench applied by the environment on the robot. We assume that the application point of the external wrench is associated with a frame C_k , attached to the link on which the wrench acts, and has its z axis pointing in the direction of the normal of the contact plane. The external wrench f_k is expressed in a frame which has the same orientation as the inertial frame, and has its origin in C_k , i.e. the application point of the external wrench f_k .

The Jacobian $J_{c_k} = J_{c_k}(q)$ is the map between the robot velocity ν and the linear and angular velocities $\mathcal{I}v_{C_k} := (\mathcal{I}\dot{p}_{C_k}, \mathcal{I}\omega_{C_k})$ of the frame C_k , i.e. $\mathcal{I}v_{C_k} = J_{c_k}(q)\nu$.

¹As an abuse of notation, we define as *wrench* a quantity that is not the dual of a *twist*.

The Jacobian has the following structure:

$$J_{C_k}(q) = \begin{bmatrix} J_{C_k}^b(q) & J_{C_k}^j(q) \end{bmatrix} \in \mathbb{R}^{6 \times n+6} \quad (2.5a)$$

$$J_{C_k}^b(q) = \begin{bmatrix} 1_3 & -S(\mathcal{I}p_{C_k} - \mathcal{I}p_B) \\ 0_{3 \times 3} & 1_3 \end{bmatrix} \in \mathbb{R}^{6 \times 6} \quad (2.5b)$$

Lastly, it is assumed that holonomic constraints act on system (2.5). These constraints are of the form $c(q) = 0$ and may represent, for instance, a frame having a constant pose with respect to the inertial frame. In the case where this frame corresponds to the location at which a contact occurs on a link, we represent the holonomic constraint as:

$$J_{C_k}(q)\dot{\nu} + \dot{J}_{C_k}(q, \nu)\nu = 0 \quad (2.6)$$

2.2 Optimization-based whole-body torque-control framework for floating-base systems

A general framework for optimization-based whole-body control may be defined as in figure 2.1, where a state machine defines desired robot trajectories, in accordance with a user-defined stack-of-tasks. Using these trajectories, feedback control policies compute desired accelerations of the robot. The whole-body torque-control problem is then formulated as an optimization problem based on the stack-of-tasks. It is solved in the form of a QP, to obtain a control input for the robot.

In order to describe in detail the whole-body torque-control problem, the following subsections first define the control input for the robot and elementary tasks to be performed by the robot as a stack-of-tasks. An optimization problem is then defined in order to compute the control input that stabilizes the desired tasks.

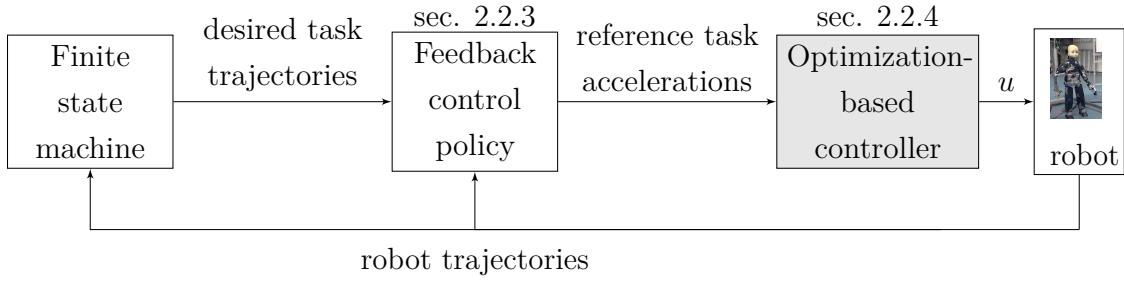


Figure 2.1 Overview of the proposed method. A whole-body controller computes the control input achieving a combination of task trajectories as defined in the finite state machine.

2.2.1 Control input

With floating base systems, the trajectory of the base must be accounted for, as well as the exchange of forces between the robot and the environment. In particular, the robot can generally exchange only pressure and friction forces with the environment. Also, since the base is free to move around, not all degrees of freedom of a floating-base system may be fully controlled, which makes it an underactuated system. The 6 DOFs associated to the base pose therefore are not directly controllable, but controlling the contact forces exchanged with the environment allow to stabilize the motion of the floating base.

The problem of whole-body torque-control is generally achieved through inverse dynamics computations, and it can be set in different ways, e.g. finding the joint torques τ that are consistent with desired ground reaction forces f_c , or finding the τ and f_c that are consistent with the desired robot acceleration $\dot{\nu}$. In other words, for the latter case, one seeks to control the right hand side of equation (2.5).

Therefore, the control input u , composed of joint torques and contact forces, may be defined as follows.

$$u = \begin{bmatrix} \tau \\ f_c \end{bmatrix} \quad (2.7)$$

where $f_c \in \mathbb{R}^{6n_c}$ is a stacked vector of contact wrenches, of which the associated contact Jacobian J_c is stacked in the same way. With this definition, the system dynamics of equation (2.5) can be reformulated as the following.

$$M(q)\dot{\nu} + h = Bu \quad (2.8)$$

where

$$B = \begin{bmatrix} \zeta & J_c^\top \end{bmatrix} \quad (2.9)$$

The control input u which achieves a desired motion of the robot is however not unique, and to deal with this redundancy, one may be concerned with defining a certain number of quantities, or tasks, related to the movement of the robot.

2.2.2 Stack-of-tasks for whole-body torque-control of a humanoid robot

For realizing whole-body motions, a number of elementary quantities related to the movement of the robot need to be controlled. The realization, or stabilization, of each one of these quantities shall be referred to as a **task**.

For example, a **Cartesian** task would concern the movement of a frame attached to the robot, while a **postural** task the movement of joints of the robot. Additionally, a **contact** task would concern the forces applied at contact points with the environment, in order to ensure that a contact remains fixed through the use of Coulomb friction cones. Thus, a stack-of-tasks represents the hierarchy of tasks which the robot must achieve simultaneously.

For the particular case of humanoid robots, the following examples of tasks can be proposed. The movement of the CoM ideally needs to be stabilized in order to keep balance, while moving the feet allows to perform walking motions. Furthermore, in order to prevent the feet from slipping on the ground, it may be important to keep the contact forces exchanged with the environment within the associated contact constraints.

However, these few tasks are not sufficient to define the movement of all DOFs composing the robot, as the upper-body also needs to be taken into account. For instance, when walking, the body may need to be kept upright and facing in the walking direction. Also, redundancy of the robot needs to be accounted for. Indeed, while a redundant robot may be able to achieve desired movements of an end effector through different trajectories of the joints, some trajectories may be more desirable than others. A postural task may be used in order to encourage the generation of more desirable trajectories, and minimizing the effort extended by the robot may allow to produce more regular movements.

Note however that of the control input, only joint torques are to be minimized. Indeed, as explained in [Romano et al., 2017] minimizing contact wrenches would have the undesirable effect of enforcing an almost constant vertical force at the contacts, independently of the center of mass position.

This list of simple tasks can be formalized as a stack-of-tasks, for instance with the following list of objectives:

- Stabilize the contact of the feet with the ground
- Stabilize the center of mass position $p_{CoM} \in \mathbb{R}^3$
- Stabilize the left foot pose $T_{Lfoot} \in SE(3)$
- Stabilize the right foot pose $T_{Rfoot} \in SE(3)$
- Stabilize the orientation of a frame on the upper-body $R_{ub} \in SO(3)$
- Track joint positions (postural task) s
- Minimize joint torques τ

Figure 2.2 illustrates the position on the iCub of each of the four frames concerned by the Cartesian tasks: CoM, feet and a frame on the upper-body. In this case, the upper-body frame is chosen on the root link, but it could just as well be on the torso or the head.

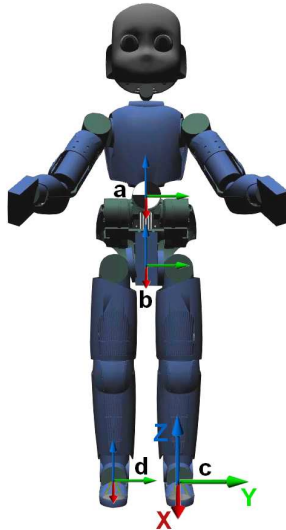


Figure 2.2 Frames considered in the control framework of section 2.3.2: **a** for the center of mass position, **b** for the root link orientation, **c** for the left foot pose and **d** for the right foot pose; **c** also shows the x-y-z convention.

Other combinations of tasks can eventually be used in order to control whole-body movements through a stack-of-tasks. For example, if one is interested in having a more precise control of the hands, stabilization of the hand poses can be used as well.

Nonetheless, recall that the configuration space of the robot evolves in a group of dimension¹ $n + 6$. Hence, besides pathological cases, when the system is subject to a set of holonomic constraints of dimension k , the configuration space shrinks into a space of dimension $n + 6 - k$. Similarly, when the combined tasks amount to a dimension larger than $n + 6$, it may not be possible to achieve all of the desired task motions simultaneously.

2.2.3 Stabilization of tasks

Given a certain number of Cartesian, postural and contact tasks for a controller, each of them needs to be stabilized in some way. It can be done for instance as detailed in the following paragraphs.

Stabilization of Cartesian tasks

In order to stabilize a Cartesian task, i.e. the pose, position or orientation of a frame \mathcal{T} attached to the robot, the velocity $v_{\mathcal{T}}$ of this frame can be computed from the associated Jacobian $J_{\mathcal{T}}$ and the robot velocity ν with the following expression.

$$v_{\mathcal{T}} = J_{\mathcal{T}}\nu \quad (2.10)$$

Deriving this expression with respect to time yields the acceleration of the frame associated to the task:

$$\dot{v}_{\mathcal{T}} = \dot{J}_{\mathcal{T}}\nu + J_{\mathcal{T}}\dot{\nu} \quad (2.11)$$

In view of equation (2.8) and (2.11), task frame accelerations $\dot{v}_{\mathcal{T}}$ can be formulated as a function of the control input u :

$$\dot{v}_{\mathcal{T}}(u) = \dot{J}_{\mathcal{T}}\nu + J_{\mathcal{T}}M^{-1}(Bu - h) \quad (2.12)$$

¹We refer here to the dimension of the associated algebra \mathbb{V} as introduced in section 2.1

Stabilization of a frame associated to a Cartesian task may then be attempted by minimizing the error on its acceleration $\tilde{\dot{v}}_{\mathcal{T}}(u)$, given a feedback term $\dot{v}_{\mathcal{T}}^*$, as follows.

$$\tilde{\dot{v}}_{\mathcal{T}}(u) = \dot{v}_{\mathcal{T}}(u) - \dot{v}_{\mathcal{T}}^* \quad (2.13)$$

Typically, the feedback term $\dot{v}_{\mathcal{T}}^*$ is composed of reference linear accelerations obtained with a proportional-derivative (PD) feedback control policy, and reference angular accelerations obtained through Euler parametrization.

Here, instead, we propose to compute reference accelerations with a proportional-derivative feedback control policy in $SE(3)$. What is particular in this method is the computation of reference angular accelerations as proposed in [Olfati-Saber, 2001, section 5.11.6, p.173], which has the advantage to avoid the creation of artificial singularities caused by the use of Euler parametrization. This is a novel application of such a control policy, in the field of humanoid robotics.

The following paragraphs describe the proposed policies for stabilizing a desired position and orientation (p_d, R_d) of a frame B , given its configuration $(p, R) := (\mathcal{I}p_B, \mathcal{I}R_B)$.

The position and orientation problems are treated separately, in order to define desired linear and angular accelerations $(\ddot{p}^*, \dot{\omega}^*) := (\mathcal{I}\ddot{p}_B^*, \mathcal{I}\dot{\omega}_B^*)$. As such, control of a frame in $SE(3)$ can be attempted using the control laws (2.15) and (2.26) yielding the desired frame acceleration as follows:

$$\dot{v}^* = \begin{bmatrix} \ddot{p}^* \\ \dot{\omega}^* \end{bmatrix} \quad (2.14)$$

Stabilization of a desired position

The desired linear acceleration is computed using the following PD feedback control policy:

$$\ddot{p}^* = \ddot{p}_d - K_{P_l}(p - p_d) - K_{D_l}(\dot{p} - \dot{p}_d) \quad (2.15)$$

where $K_{P_l} > 0$ and $K_{D_l} > 0$ are the linear proportional and derivative gains, and the subscript d indicates desired values.

Stabilization of a desired orientation

The problem of stabilizing a desired orientation $R \in SO(3)$, on the other hand, may not be straightforward. For instance, the topology of $SO(3)$ forbids the design of smooth controllers that globally asymptotically stabilize a reference orientation [Bhat and Bernstein, 2000]. In consequence, quasi-global asymptotic stability is commonly guaranteed by orientation controllers.

As a way to achieve it, the distance between a desired attitude R_d and the current R can be measured using the following function.

$$\delta(R, R_d) = \frac{1}{2} \text{tr}(1_3 - RR_d^\top) \quad (2.16)$$

It has been shown in [Olfati-Saber, 2001, section 5.11.6] to be equivalent, up to a constant multiplicative factor, to the Euclidean distance measured using a 3D orthonormal basis.

Note that as discussed in [Olfati-Saber, 2001, section 5.11.6], the rotation θ around a unit vector k can be expressed using Rodrigues' formula as follows.

$$R(K, \theta) = 1_3 + \sin\theta k^\wedge + (1 - \cos\theta)k^\wedge^2 \quad (2.17)$$

As a result,

$$\frac{1}{2} \text{tr}(1_3 - R(K, \theta)) = 1 - \cos\theta \leq 2 \quad (2.18)$$

Thus, $\frac{1}{2} \text{tr}(1_3 - R) : SO(3) \rightarrow [0, 2]$.

Furthermore, letting $\omega_d = (R_d^\top \dot{R}_d)^\vee$, an important property of $\delta(R, R_d)$ is that the time derivative of $\delta(R, R_d)$ yields the following expression.

$$\dot{\delta}(R, R_d) = \text{skew}(RR_d^\top) \cdot ({}^B\omega - {}^B\omega_d) \quad (2.19)$$

As a result, the following lemma is proposed in [Olfati-Saber, 2001, section 5.11.6].

Lemma 1 *The state feedback law*

$$\omega = \omega_d - c_0 \text{skew}(R_d^\top R)^\vee \quad (2.20)$$

where $c_0 > 0$, (almost) globally asymptotically stabilizes the equilibrium $\delta = 0$ of

$$\dot{\delta} = \text{skew}(R_d^\top R)^\vee \cdot (\omega - \omega_d) \quad (2.21)$$

for all $\delta(0) \neq 2$.

The proof considers the closed-loop system

$$\dot{\delta} = -c_0 \left| \text{skew}(R_d^\top R)^\vee \right|^2 \quad (2.22)$$

where $\dot{\delta} < 0$ for $\delta > 0$. Then, in order to show that $\text{skew}(R_d^\top R)^\vee = 0$ implies $\delta = 0$, one must have that $R_d^\top R$ is a symmetric matrix. In this case, $SO(3)$ allows for two possibilities of symmetric matrices, as can be deduced from Rodrigues' formula: 1_3 , and $1_3 + 2k^\wedge^2$, with $k \in \mathbb{R}^3$ an arbitrary vector. $R_d^\top R = 1_3$ means that $\delta(R, R_d) = 0$, while $R_d^\top R = 1_3 + 2K^\wedge^2$ means that $\delta(R, R_d) = 2$. The assumption that $\delta(0) \neq 2$ allows to avoid having more than a single equilibrium point, by making $R_d^\top R = 1_3$ the only acceptable solution for $\text{skew}(R_d^\top R)^\vee = 0$, which implies $R = R_d$, making $\delta = 0$ the only invariant equilibrium that proves (almost) global asymptotic stability of $\delta = 0$ for $\delta(0) < 2$.

Stabilizing the orientation of a frame can then be achieved through the following lemma, obtained from [Olfati-Saber, 2001, section 5.11.6].

Lemma 2 *Consider the following orientation dynamics:*

$$\begin{cases} \dot{R} = \omega^\wedge R \\ \dot{\omega} = \dot{\omega}^* \end{cases} \quad (2.23)$$

where $\dot{\omega}^* \in \mathbb{R}^3$ is considered as control input. Assume that the control objective is the asymptotic stabilization of a desired attitude $(R_d(t), \omega_d(t)) \in SO(3) \times \mathbb{R}^3$.

Let the term λ be defined as follows

$$\lambda = K_{0\omega} \text{skew}(R_d^\top R)^\vee \quad (2.24)$$

which yields the following time derivative.

$$\dot{\lambda} = K_{0\omega} \text{skew}(R_d^\top R^B \omega^\wedge - {}^B \omega_d^\wedge R_d^\top R)^\vee \quad (2.25)$$

Then,

$${}^B\omega_d = R_d^\top \omega_d \quad (2.26a)$$

$$\begin{aligned} {}^B\dot{\omega}^* &= -K_{0\omega} K_{1\omega} \text{skew}(R_d^\top R)^\vee \\ &\quad - K_{1\omega} ({}^B\omega - {}^B\omega_d) \\ &\quad - \dot{\lambda} \end{aligned} \quad (2.26b)$$

$$\dot{\omega}^* = R {}^B\dot{\omega}^* + \dot{\omega}_d \quad (2.26c)$$

renders the equilibrium point $(R, \omega) = (R_d, \omega_d)$ quasi-globally stable, for all $\delta(0) \neq 2$. $K_{0\omega} > 0$ and $K_{1\omega} > 0$ are gains of the rotational control laws.

The rotational control laws proposed above have been proven in [Olfati-Saber, 2001, section 5.11.6] for (almost) global exponential stability, using Lyapunov analysis based on the following candidate Lyapunov function.

$$V = 2K_{0\omega} K_{1\omega} \delta(R, R_d) + \frac{1}{2} |{}^B\omega - \lambda|^2 > 0 \quad (2.27)$$

Its time derivative yields the following result, using the closed-loop dynamics of the system.

$$\dot{V} = 2K_{0\omega} K_{1\omega} \text{skew}(RR_d^\top) \cdot ({}^B\omega - {}^B\omega_d) + ({}^B\omega - \lambda) \cdot ({}^B\dot{\omega} - \dot{\lambda}) \quad (2.28a)$$

$$= -K_{1\omega} |({}^B\omega - {}^B\omega_d)|^2 - 2K_{0\omega} K_{1\omega} |\text{skew}(R_d^\top R)^\vee|^2 \quad (2.28b)$$

The term “(almost)” is therefore used due to the fact that $\dot{V} < 0$ for $(R, \omega) \neq (R_d, \omega_d)$. Note that, as explained for lemma 1, the control policy assumes $\delta(0) \neq 2$.

Stabilization of postural tasks

In order to stabilize a postural task, the process is similar to the one presented for Cartesian tasks in section 2.2.3. In view of equation (2.8), one can obtain $\ddot{s}(u)$, a formulation of the joint accelerations \ddot{s} as a function of the control input u :

$$\ddot{s}(u) = \zeta^\top [M^{-1}(Bu - h)] \quad (2.29)$$

Stabilization of the postural task may then be attempted by minimizing the error on the joint accelerations $\tilde{\ddot{s}}(u)$, given a feedback term \ddot{s}^* .

$$\tilde{\ddot{s}}(u) = \ddot{s}(u) - \ddot{s}^* \quad (2.30)$$

Typically, the feedback term \ddot{s}^* consists in reference accelerations obtained with a PD feedback control policy.

$$\ddot{s}^* = \ddot{s}_d - K_{D_s}(\dot{s} - \dot{s}_d) - K_{P_s}(s - s_d) \quad (2.31)$$

where the subscript d indicates desired values, and K_{P_s} , K_{D_s} are feedback gain matrices with different gain values for each joint. Alternatively, \ddot{s}^* can eventually be obtained through inverse kinematics by taking into account Cartesian tasks.

Stabilization of contact tasks

In order to stabilize a contact, forces exchanged between the robot and the environment at the contact must be constrained in some way, for example as proposed in [Caron et al., 2015].

First, in order for the contact to exist, the normal component of the reaction force f_{k_z} (e.g. along the z -axis, when considering the contact plane to be on the $x - y$ plane) should remain positive, or larger than a threshold value ($f_{k_z} > f_{k_z_{min}} > 0$).

$$f_{k_z} > f_{k_z_{min}} \quad (2.32)$$

Then, in order to prevent slipping of the robot on the contact surface, the tangential component of the ground reaction force f_{k_t} (e.g. the norm of the forces along the x - and y -axes) should not be larger than f_{k_z} multiplied by the Coulomb static friction coefficient associated to the contact surfaces μ_c :

$$|f_{k_t}| < \mu_c f_{k_z} \quad (2.33)$$

The projection of the constraint (2.33) on the contact plane shows that the tangential forces must remain within a circle of radius $\mu_c f_{k_z}$. Thus, the total reaction force, when considered along the x -, y - and z -axes, is constrained within the shape of a cone, which explains why it is usually described as the “friction cone”. In order to make

its computation easier, the circular shape can be approximated with straight lines intersecting in a given number n_v of vertices, and n_v constraints are then defined (one for each vertex).

To prevent also slipping due to torsion, the torsional friction about the z -axis, given a static torsional friction coefficient μ_t , is also considered with the following constraint.

$$-\mu_t f_{k_z} < \tau_z < \mu_t f_{k_z} \quad (2.34)$$

Note that this constraint neglects the torsional effect of forces and moments acting along and about the x - and y -axes over the surface of the support polygon. A formulation of the constraint on τ_z considering those can be found in [Caron et al., 2015]. Nonetheless, in this work we prefer to use the approximation of equation (2.34), as it only relies on the precise estimation of f_{k_z} .

Additionally, in order to ensure stationary contacts, the center of pressure (CoP) needs to remain within the support polygon. Constraints on the position of the CoP can then be expressed as a linear inequality, given the expression of the reaction force at the zero moment point.

$$-d_x < \frac{\tau_y}{f_{k_z}} < d_x \quad (2.35a)$$

$$-d_y < \frac{\tau_x}{f_{k_z}} < d_y \quad (2.35b)$$

where the reaction force is assumed to be measured in the middle of the contact, and the contact is a rectangle of dimensions $2d_x \times 2d_y$.

Rearranging the equations (2.32) to (2.35) and combining them together, the contact constraints can be formulated as linear inequality constraints of the following form.

$$Cu \leq b \quad (2.36)$$

Note that besides the inequality constraint of equation (2.36), a rigid contact with the environment should keep a constant pose of the frame attached to the robot at the location where contact occurs, with respect to the inertial frame. This can be represented as the holonomic constraint of equation (2.6), and can be achieved by tracking the frame acceleration to zero, as described in section 2.2.3.

2.2.4 Whole-body control optimization problem

The complexity of floating-base systems, such as humanoid robots, often makes it easier to solve inverse dynamics through optimization. In particular, the whole-body control problem shows to be well posed for quadratic programming [Escande et al., 2010; Righetti et al., 2013; Righetti and Schaal, 2012]. In this case, a QP solver is used as a block box which, given a quadratic objective function with linear equality and inequality functions, returns the optimal control input. The whole-body control problem formulation is then often of the shape described in [Del Prete and Mansard, 2016]:

$$u^* = \arg \min_u |Xu - x|^2 \quad (2.37a)$$

$$\text{subject to } D_{eq}u + d_{eq} = 0 \quad (2.37b)$$

$$Du + d \leq 0 \quad (2.37c)$$

where the cost function represents the squared error on the task accelerations, which for torque-controllers is generally defined to stabilize centroidal momentum dynamics, Cartesian or postural tasks [Herzog et al., 2014; Lee and Goswami, 2012; Nava et al., 2016; Salini et al., 2011; Stephens and Atkeson, 2010]. The equality constraint defined by D_{eq} and d_{eq} can be used to represent the system dynamics (2.5) and contact acceleration constraints (2.6). The inequality constraints defined by D and d can be used, among other things, to define joint limits or contact constraints.

The optimized quantity u , for its part, may be defined from different combinations of the robot acceleration, joint torques and contact forces, depending on the formulation of the control problem. For instance, assuming that any control objective can be expressed as a linear combination of $\dot{\nu}$, f_c and τ , one can define $u = (\dot{\nu}, f_c, \tau)$ [Del Prete and Mansard, 2016]. As proposed in [Herzog et al., 2016; Pucci et al., 2016a], the system dynamics (2.5) can be decomposed in such a way as to decouple joint and base frame accelerations, allowing to express joint torques τ as a linear combination of joint accelerations \ddot{s} and contact forces f_c . It results that, for increased computational efficiency, the optimization problem can be solved in terms of $u = (\dot{\nu}, f_c)$.

In a similar fashion, the robot acceleration $\dot{\nu}$ can be obtained as a linear combination of τ and f_c , under the assumption that M is well posed, by reorganizing equation (2.8) as follows.

$$\dot{\nu} = M^{-1}(Bu - h) \quad (2.38)$$

Task accelerations can then be formulated in function of τ and f_c , as shown in equations (2.11) and (2.29). This formulation therefore integrates system dynamics into the objective function of the optimization problem, which then does not require the use of constraints representing system dynamics. Such a procedure then makes it advantageous to obtain desired joint torques directly from the solution of the optimization problem, formulated in terms of $u = [\tau^\top \ f_c^\top]^\top$.

From there, algebraic manipulation allows to reformulate the problem (2.37) as QP:

$$u^* = \arg \min_u \frac{1}{2} u^\top H u + u^\top g \quad (2.39a)$$

$$\text{subject to } \underline{b} \leq A u \leq \bar{b} \quad (2.39b)$$

where the Hessian matrix H is symmetric and positive definite and g is the gradient vector. A is the constraint matrix, with \underline{b} and \bar{b} the associated lower and upper constraint vectors.

In order to define the controller as an optimization problem, a hierarchy of tasks needs to be defined. The interest of the stack-of-tasks approach is that it allows the definition of various hierarchization strategies, which may be a combination of “strict” and “soft” task priorities. Section 2.3 shall explore the development of different tasks and hierarchization strategies for the framework we just defined, while section 2.4 compares the results they can achieve.

2.3 Implementation of optimization-based whole-body torque-controllers

The flexibility offered by the framework defined in section 2.2 allows to define a controller using several different configurations (i.e. different tasks, task hierarchy, finite state machine and controller parameters), in order to achieve a desired behavior of the robot. Therefore, through an iterative process, we have defined a series of whole-body torque-controllers for the iCub. The following subsections discuss three main implementations of the controller.

1. Contact and Cartesian tasks are strictly hierarchized with respect to other tasks.
2. A soft tasks hierarchy is defined between Cartesian and postural tasks, while the contact task keeps a strict higher priority. A finite state machine consisting of 11 states is defined for stepping motions.
3. A soft tasks hierarchy, similar to the one in the second implementation, is defined, but with different tasks, and a new state machine is defined with only 5 states for stepping motions.

The results achieved with each of these controllers can then be confronted, allowing to gain some insight on the formulation of controllers based on the proposed framework.

2.3.1 Implementation of a strict tasks controller

A first implementation of a whole-body controller is presented in this subsection. It is based on a stack-of-tasks, including Cartesian, postural and contact tasks.

The controller is used to stabilize the contact of the feet on the ground, the position of the center of mass, the orientation of a frame attached to the pelvis, as well as the position and orientation of the feet frames, while a low-priority postural task provides reference positions for each joint of the robot. Additional tasks are defined for the regularization of the joint torques, and their rate of change. Joint torques and contact forces allowing to stabilize tasks are obtained through quadratic programming optimization, and fed to the robot as control input.

Stack-of-tasks

The stack-of-tasks used for this controller is defined from the following tasks:

- Stabilize the contact of the feet with the ground
- Stabilize the center of mass position $p_{CoM} \in \mathbb{R}^3$
- Stabilize the left foot pose $T_{left} \in SE(3)$
- Stabilize the right foot pose $T_{right} \in SE(3)$
- Stabilize the root link (pelvis) orientation $R_{root} \in SO(3)$

- Track joint positions (postural task) s
- Minimize joint torques τ
- Keep the rate of change of joint torques smaller than $\dot{\tau}_{max} \in \mathbb{R}$

The last item on the list (keep the rate of change of joint torques smaller than a given threshold) was added in order to ensure a measure of continuity in the joint torques.

Optimization problem formulation

The initial implementation of this stack-of-tasks considers a formulation with strict tasks priorities. The Cartesian tasks are all given the highest priority together with contact tasks, by being set as constraints to the control optimization problem. The postural task, for its part, is used in the objective function of the controller, along with regularization on joint torques. Applying the formulation of (2.37), we define the controller as follows.

$$u^* = \arg \min_u \frac{1}{2} |\tilde{s}(u)|^2 + \lambda_\tau \tau(u) \quad (2.40a)$$

$$\text{subject to } Cu \leq b \quad (2.40b)$$

$$\dot{\Upsilon}^* = \dot{\Upsilon}(u) \quad (2.40c)$$

$$(\zeta_\tau u_{-1} - \dot{\tau}_{max} t_s) \leq \zeta_\tau u \leq (\zeta_\tau u_{-1} + \dot{\tau}_{max} t_s) \quad (2.40d)$$

where λ_τ is a parameter that controls the importance of the regularization on τ , in order to encourage solutions with lower torque amplitudes.

The accelerations associated to Cartesian tasks on the CoM, root link, left foot and right foot are stacked together into $\dot{\Upsilon}$, defined as follows.

$$\dot{\Upsilon} = \begin{bmatrix} \dot{v}_{CoM} \\ \dot{v}_{root} \\ \dot{v}_{left} \\ \dot{v}_{right} \end{bmatrix} \quad (2.41)$$

The torque continuity constraint of equation (2.40d) constrains the rate of change of torque values, by ensuring that the difference between optimized torques $\tau(u) = \zeta_\tau u$ and torques applied at the previous time step τ_{-1} is smaller than a threshold, for all joints. In this equation, t_s is the time step duration, $\dot{\tau}_{max}$ is the maximum desired torque rate of change, and $\zeta_\tau = [1_n, 0_{n \times 6}]$ is a selector matrix.

During early experimentation, this formulation of the Cartesian tasks as constraints has turned out to produce a relatively stiff behavior of the robot, which is not so advantageous for stability. In fact, in the present context, tasks may have relative priorities, such that allowing a trade-off between weighted tasks may improve results. For this reason, results achieved with this strict tasks controller will not be shown. Instead, efforts have been re-directed towards a soft task formulation, as shall be presented in the following section.

2.3.2 Implementation of the soft tasks controller #1

A second whole-body controller is presented in this section. It is based on the same stack-of-tasks as the controller of subsection 2.3.1: the controller is used to stabilize the contact of the feet on the ground, the position of the center of mass, the orientation of a frame attached to the pelvis, as well as the position and orientation of the feet frames, while a low-priority postural task provides reference positions for each joint of the robot. Additional tasks are defined for the regularization of the joint torques, and their rate of change. Joint torques and contact forces allowing to stabilize tasks are obtained through QP optimization, and fed to the robot as control input.

Stack-of-tasks

The stack-of-tasks used for this controller is again defined from the following tasks:

- Stabilize the contact of the feet with the ground
- Stabilize the center of mass position $p_{CoM} \in \mathbb{R}^3$
- Stabilize the left foot pose $T_{left} \in SE(3)$
- Stabilize the right foot pose $T_{right} \in SE(3)$
- Stabilize the root link (pelvis) orientation $R_{root} \in SO(3)$

- Track joint positions (postural task) s
- Minimize joint torques τ
- Keep the rate of change of joint torques smaller than $\dot{\tau}_{max} \in \mathbb{R}$

In this new formulation, contact stabilization remains to be achieved with inequality constraints, as well as keeping the rate of change of joint within limits. For the rest of the tasks, each one is attributed a priority within the following set of task weights.

$$\mathbf{w} = \{w_{\Upsilon}, w_s, w_{\tau}\} \quad (2.42)$$

where the terms $w_{\Upsilon}, w_s, w_{\tau} \in \mathbb{R}$ refer to weights associated to the Cartesian tasks (CoM, left foot, right foot and root link), postural task and joint torque regularization, respectively.

Optimization problem formulation

The control problem is formulated with soft task priorities, by using a weighted sum of task errors in the cost function:

$$u^* = \arg \min_u w_{\Upsilon} |\dot{\Upsilon}(u) - \dot{\Upsilon}^*|^2 + w_s |\ddot{s}(u) - \ddot{s}^*|^2 + w_{\tau} \tau(u) \quad (2.43a)$$

$$\text{subject to } Cu \leq b \quad (2.43b)$$

$$(\zeta_{\tau} u_{-1} - \dot{\tau}_{max} t_s) \leq \zeta_{\tau} u \leq (\zeta_{\tau} u_{-1} + \dot{\tau}_{max} t_s) \quad (2.43c)$$

In this formulation, $w_{\Upsilon}, w_s, w_{\tau}$ are the weights associated to Cartesian, postural and effort minimization tasks, respectively. Since task tracking is of high priority in the case of the balancing controller, w_{Υ} shall be attributed the highest value. Furthermore, task weights are kept at constant values over time.

The optimization problem obtained in equation (2.43) can easily be transformed into a QP formulation of the form of equation (2.39). The Hessian matrix H and gradient vector g can be obtained from equations (2.9), (2.12), (2.29) and the previously formulated optimization of equation (2.43). The constraint matrix A is then obtained from C in the contact constraints (2.43b), with upper bound b , and no specific lower bound. Note that the size of contact constraints are adjusted each time step, depending on the number of feet which are currently in contact with the ground.

Finite state machine

A finite state machine is used, in order to define desired setpoints for Cartesian and postural tasks, in function of the state of the robot. It is also used for gain scheduling, outputting PD gains used in the computation of feedback control policies for the Cartesian and postural tasks. To define its output, the finite state machine takes as input some user-defined positions and PD gains for the Cartesian and postural tasks, for each state.

In this case, the state machine has been applied to a scenario in which the robot performs the whole-body motion of stepping in place, and is divided into 11 states for this purpose, as illustrated in figure 2.3. The formulation of the state machine, desired trajectories defined for each task and state, as well as transitions between states, are described in detail in appendix A.

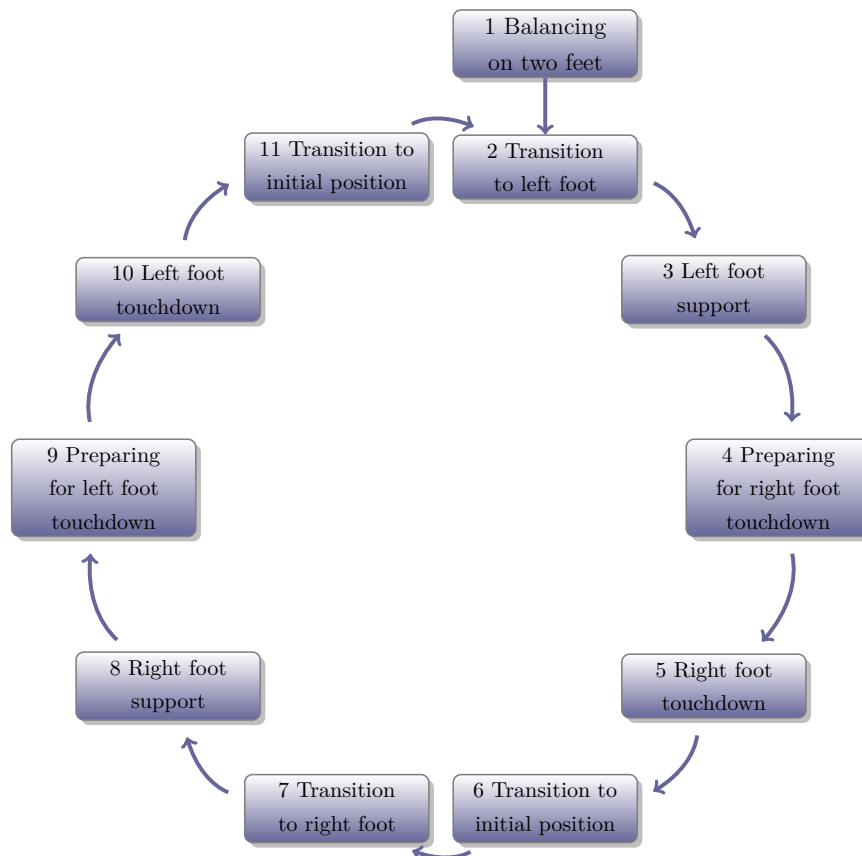


Figure 2.3 States of the finite state machine introduced in section 2.3.2 for generating walking in place motion

2.3.3 Implementation of the soft tasks controller #2

Following the definition of the controller presented in subsection 2.3.2, a new, modified version of the controller has been developed in order to improve its ease of use.

The most important differences are the following:

- The orientation task is applied to the neck frame instead of the root link frame, which allows a higher compliance of the hips and torso. This is helpful for absorbing the impact at foot touchdown, by dissipating energy through hips and torso motion.
- Cartesian tasks are defined for swing foot and stance foot, instead of left and right feet. This allows to take into account variations in task priorities depending on the phase of walking: the pose of the foot when it is on the ground may not need to be tracked with the same priority as when performing a step¹.
- Each Cartesian task is given a specific weight, which allows to prioritize Cartesian tasks with respect to each other. For instance, the neck frame orientation task may not need to be achieved with the same priority as the CoM task.
- The use of the postural task is limited to stabilizing redundant degrees of freedom; desired joint positions are not defined for each state anymore, but represent the initial posture of the robot.
- The torque continuity constraint is discarded, since its use was not beneficial when switching contact conditions.
- The state machine is simplified to 5 states, taking advantage of the symmetry of the stepping motion. Conditions on the postural task are removed, a maximal state duration is set to prevent that the robot remains stuck in a state, and a significantly smaller number of user-defined parameters is required.
- Gains used by the feedback control policies are equal across all states.

The modified whole-body control framework is then used to stabilize the position of the center of mass, the pose of frames attached to the swing and stance feet, as well as the orientation of a frame attached to the neck, while a postural task allows to

¹Recall that in the formulation of the controller, the pose of each foot is tracked at all times, allowing to move the foot as desired when in swing phase, and to ensure rigid contact by keeping the foot in place when in stance phase.

stabilize the motion. Joint torques and contact forces are again obtained through QP optimization, and sent to the robot as control input.

Stack-of-tasks

The stack-of-tasks used for this controller is defined from the following tasks:

- Stabilize the contact of the feet with the ground
- Stabilize the center of mass position $p_{CoM} \in \mathbb{R}^3$
- Stabilize the stance foot pose $T_{stance} \in SE(3)$
- Stabilize the swing foot pose $T_{swing} \in SE(3)$
- Stabilize the neck orientation $R_{neck} \in SO(3)$
- Track joint positions (postural task) s
- Minimize joint torques τ

Contact stabilization is achieved with inequality constraints, as presented in section 2.2.3. For the rest, soft tasks are used as prioritization scheme. For this purpose, each task is attributed a priority within the following set of task weights.

$$\mathbf{w} = \{w_{CoM}, w_{stance}, w_{swing}, w_{neck}, w_s, w_\tau\} \quad (2.44)$$

where the terms $w_{CoM}, w_{stance}, w_{swing}, w_{neck} \in \mathbb{R}$ refer to weights associated to the CoM, stance foot, swing foot and neck Cartesian tasks, and $w_s, w_\tau \in \mathbb{R}$ to the weights associated to the postural task and joint torque regularization, respectively.

Optimization problem formulation

The control problem is formulated as the following optimization problem.

$$u^* = \arg \min_u w_\tau |\tau(u)|^2 + w_s |\tilde{s}(u)|^2 + \sum_{\mathcal{T}} w_{\mathcal{T}} |\tilde{v}_{\mathcal{T}}(u)|^2 \quad (2.45a)$$

$$\text{subject to } Cu \leq b \quad (2.45b)$$

where the cost function (2.45a) is computed as the weighted sum of all task objectives, in which $\mathcal{T} \in \{CoM, stance, swing, neck\}$ refers to Cartesian tasks on CoM, feet and

neck. The constraint equation (2.45b) ensures that the contact forces remain within the associated contact constraints.

Reorganizing the terms in the cost function, one can easily verify that it has the form of a QP problem following the same procedure as in 2.3.2.

Finite state machine

A finite state machine is used in order to output desired setpoints for Cartesian tasks, in function of the state of the robot. To define its output, the finite state machine takes as input some user-defined positions for the Cartesian tasks.

The state machine is applied to a scenario in which the robot performs the whole-body motion of stepping in place, and is divided into 5 states for this purpose, as illustrated in figure 2.4. The formulation of the state machine, desired trajectories defined for each task and state, as well as transitions between states are described in detail in appendix B.

Note that with this state machine, the purpose of the postural task is solely to stabilize redundant degrees of freedom, and desired joint positions are simply defined as the initial posture of the robot. In order to encourage that the reference postural task accelerations input to the controller are consistent with reference Cartesian task accelerations, we attempt to take into account feedback terms on Cartesian tasks $\dot{v}_{\mathcal{T}}^*$ into the computation of the postural feedback term \ddot{s}^* , as exposed in appendix B.

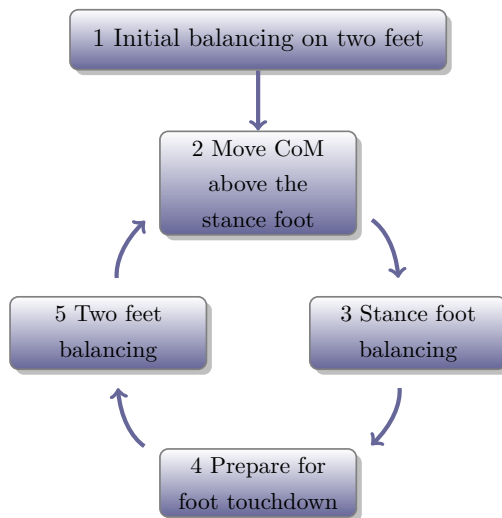


Figure 2.4 States of the finite state machine defined in section 2.3.3 for generating a stepping motion

2.4 Application to walking in place

The controllers presented in section 2.3 have been implemented in Matlab/Simulink using WBToolbox [Romano et al., 2017], and the open-source software package qpOASES [Ferreau et al., 2014] is used for QP solving. Each implemented controller runs in real-time, generating joint torque commands every hundredth of a second, such that the time step duration $t_s = 0.01$ s. Tests have been performed in simulation and real-world experiments with the iCub, using 23 DOFs on legs, arms and torso.

To validate the effectiveness of the proposed method, each controller is applied to a scenario in which the robot performs a stepping motion, as if walking in place: balancing on two feet, then repeatedly switching between double and single support by lifting one foot, placing it back on the ground, and repeating with the other foot.

The next subsections present the experiments performed with soft tasks controllers #1 and #2.

2.4.1 Walking in place with the soft tasks controller #1

Experiments with the soft tasks controller #1 have been conducted both in simulation, and with the real robot. In these experiments, the foot is lifted 5 cm above the ground, and then the foot is kept in its lifted position for a duration of 5 seconds in simulation experiments, and 15 seconds in real world experiments.

In total, the following number of parameters need to be adjusted for the controller:

- 4 parameters for the stabilization of contact tasks
- 3 task weights
- 11 parameters of the finite state machine
- 18 proportional gains of Cartesian tasks, for each of the 11 states of the finite state machine (in total over all states: 198)
- 18 derivative gains of Cartesian tasks, for each of the 11 states of the finite state machine (in total over all states: 198)
- 23 proportional gains of the postural task, for each of the 11 states of the finite state machine (in total over all states: 253)

- 23 derivative gains of the postural task, for each of the 11 states of the finite state machine (in total over all states: 253)
- 6 displacement values for the Cartesian tasks (movement of the CoM and feet), over 8 states of the finite state machine (in total: 48)
- 23 joint position values for the postural task, over 8 states of the finite state machine (in total: 184)

This list amounts to a total of 1152 parameters which may be tuned for the controller. Fortunately, all derivative gains can be set as a fixed value with respect to the proportional gains (twice the squareroot of the corresponding proportional gain for simulation experiments, or 0 for real-world experiments), allowing to reduce the number of parameters that are actively tuned to 699. Furthermore, symmetry could be accounted for in simulation experiments, further reducing the number of parameters to 485. However, for real-world experiments, the robot needs adjustments which are not always symmetric.

Parameters of the controller which allow to successfully achieve the desired motion have been obtained through manual tuning. They are set as described in appendix A, where task weight values are provided, as well as parameters used with the finite state machine, proportional and derivative gains defined for Cartesian and postural tasks of the controller, and desired Cartesian and postural task values.

Note that different values are used between simulation experiments and real-world experiments, as a separate tuning process needs to be performed on the robot. In particular, the leg joint positions for the postural task have been adjusted, in order to ensure that the knee joint remains away from its limit, and that the ankle of the support foot is in a position which improves balance. Also, proportional and derivative gains have been adjusted in function of the results obtained when experimenting with the robot: postural gains have been increased. On the other hand, derivative gains have been decreased, due to the damping effects already present on the robot. Finally, thresholds and displacements used in the finite state machine have been adjusted.

The robot behavior achieved in simulation is illustrated in figure 2.5a: the robot begins in double support, then transitions to single support on the left foot, before lifting the right foot and lowering it back to the ground and repeating the same process on the other side. It can be noted that the root link and feet remain horizontal, as required by the orientation tasks. The robot can repeat the process of lifting one foot

after the other practically indefinitely, without loss of balance: it has been validated in simulation for a minimum of 75 continuous cycles. However, the results presented here are limited to showing the first cycle.

The trajectories obtained in simulation for the center of mass and feet are shown in figure 2.6. The error on the center of mass is generally kept below 0.01 m at all times, but it can be observed that the tracking error is higher along the z -axis. As for the feet tasks, the tracking error is higher, being contained below 0.03 m before it is stabilized. These lesser tracking precisions are likely due to the combination of a few factors.

Firstly, since gains associated to the CoM task are given significantly higher values compared to the other tasks, its tracking tends to be more precise. Secondly, recall that all Cartesian tasks are attributed the same priority, while desired trajectories of the CoM, feet, root link and joints are defined independently. However, on the robot, these trajectories are in fact not independent, each one potentially affecting the others. For example, at foot liftoff, task trajectories are defined in order to lift the swing foot up, while the CoM position, stance foot pose and root link orientation remain the same. Nonetheless, lifting the left foot up would have the effect of moving the CoM up. Desired joint trajectories that are not exactly aligned with the foot lifting motion will also cause tracking errors. In order to minimize task tracking errors, the controller will then seek to achieve a trade-off between all tasks, allowing increased errors on the conflicting tasks.

As illustrated in figure 2.5b, the walking in place motion has also been achieved with the real robot. Contrarily to simulation experiments, only two consecutive strides have been performed with the robot: it could successfully lift its left foot and then its right, twice. Trajectories obtained for the center of mass and feet are shown for one stride in figure 2.7. The error on the center of mass has generally been kept below 0.02 m in each direction at all times, with the error on the x -axis being the largest. Again, feet tasks have been provided lower PD gains than the CoM. As a result, the foot is first moved of about 0.1 m forward (due to the hip bending faster than the knee) before being brought back by the bending knee. At foot touchdown, each foot is brought back to its initial position with an error below 0.02 m.

The error obtained on the orientation tasks for one stride is shown in figure 2.8, where the error e_R is represented with

$$e_R = \left| RR_d^T - 1 \right| \quad (2.46)$$

The obtained graphs show that the orientation error of the lifted feet is stabilized. The root orientation error is also stabilized, although it seems to increase for the second footstep; there may be a correlation with the increase of the CoM position error.

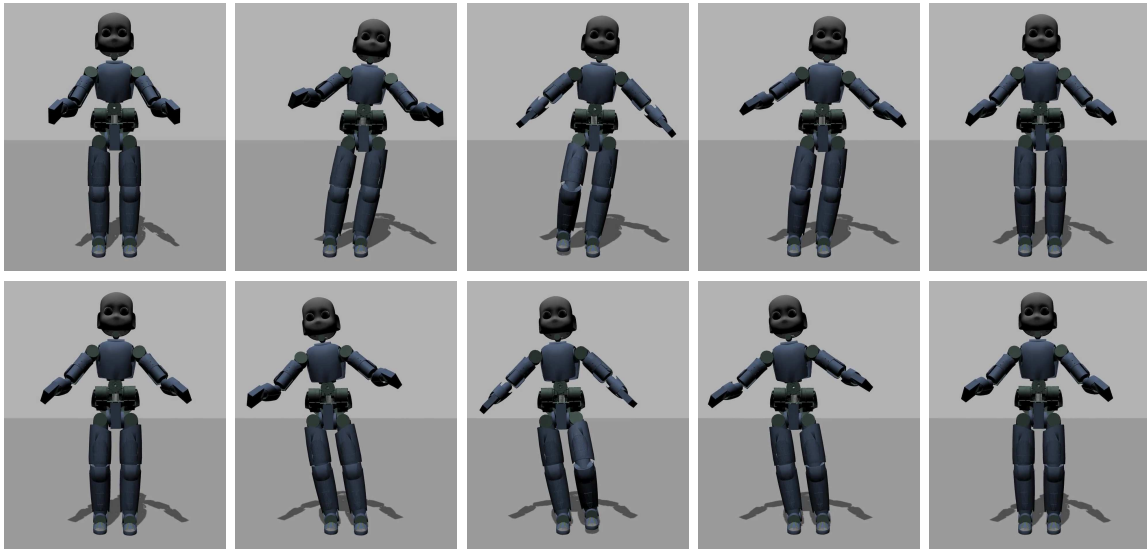
Exploiting embedded force-torque sensors in the legs of the robot allows to estimate contact forces [Traversaro, 2017; Traversaro et al., 2015]. Each leg can be virtually cut into an independant subsystem, at the location of the force-torque sensor. The position and velocity of the subsystem can then be measured or estimated from accelerometers, gyroscopes and joint encoders. We assume the contact position to be known, and the subsystem to be subject to the force measured by the force-torque sensor and a resultant external contact force. This allows to compute external forces through a classical recursive Newton-Euler algorithm over the subsystem, given its dynamics in the form of equation (2.5). The forward step of the recursive Newton-Euler algorithm also allows to obtain the position, velocity and acceleration of the links of the subsystem.

In turn, joint torques applied on the robot can be estimated as follows. A parametric representation of the system dynamics (2.5) is defined, such that torques are obtained as a linear function of a known set of parameters and a regressor matrix, that can be computed from the subsystem link positions, velocities and accelerations obtained above, as detailed in [Traversaro, 2017; Traversaro et al., 2015].

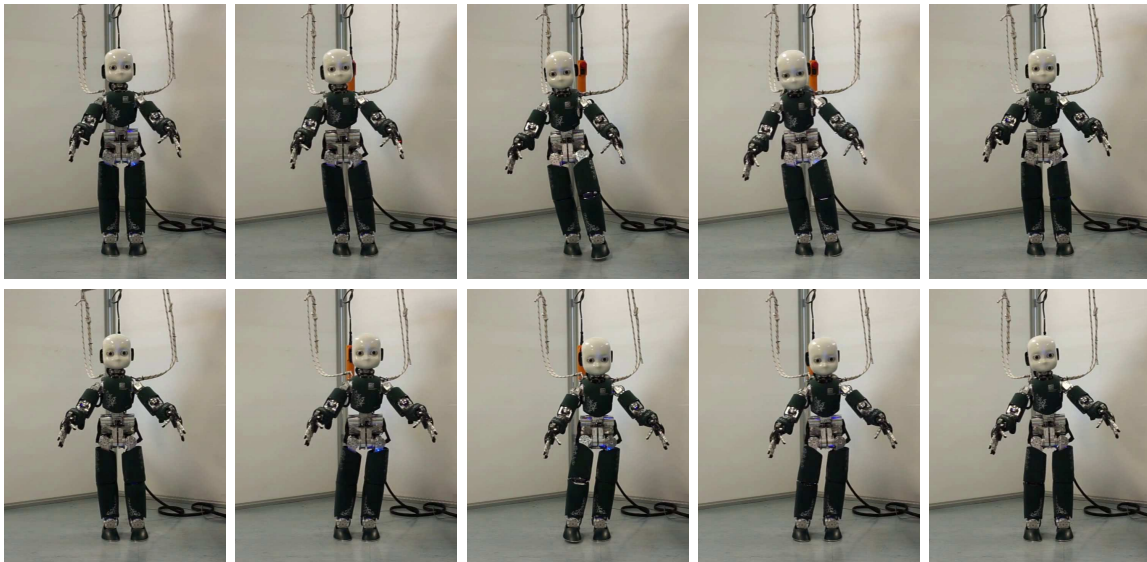
This allows us to show, in figure 2.9, the vertical component of the estimated resulting contact forces at the feet. At the moment of contact switching, although the magnitude of the forces vary rather rapidly, the forces do not show discontinuities which can not be handled by the robot. Furthermore, we have noted that at the moment of contact switching, small variations of conditions seem to cause variations in the solution of the controller. The use of the postural task then shows to be helpful in enforcing repeatability of the robot behavior.

The noise observed in the measured contact forces however do not appear to be discontinuities arising from the formulation of the controller. When tested in simulation, the desired contact wrenches computed by the QP solver show to be continuous, with minimal noise; the difference between measured and desired contact forces is contained within 50 N at most, except at contact switching. In this case, the error does not surpass half of the robot weight, and can be explained by the fact that the impact of the foot with the ground is not explicitly anticipated by the controller.

Control torques obtained from the QP solver are directly applied to the robot. They are shown in figure 2.10 for the joints which are most critical for balancing: the hips and ankles. The graphs show that torques are contained within a feasible range, and they are relatively smooth, apart from the peaks which can be observed at foot liftoff and touchdown.



(a) Simulation experiments



(b) Real-world experiments

Figure 2.5 Walking in place motion achieved with the **soft tasks controller #1** in simulation and real-world experiments: lifting one foot, then the other.

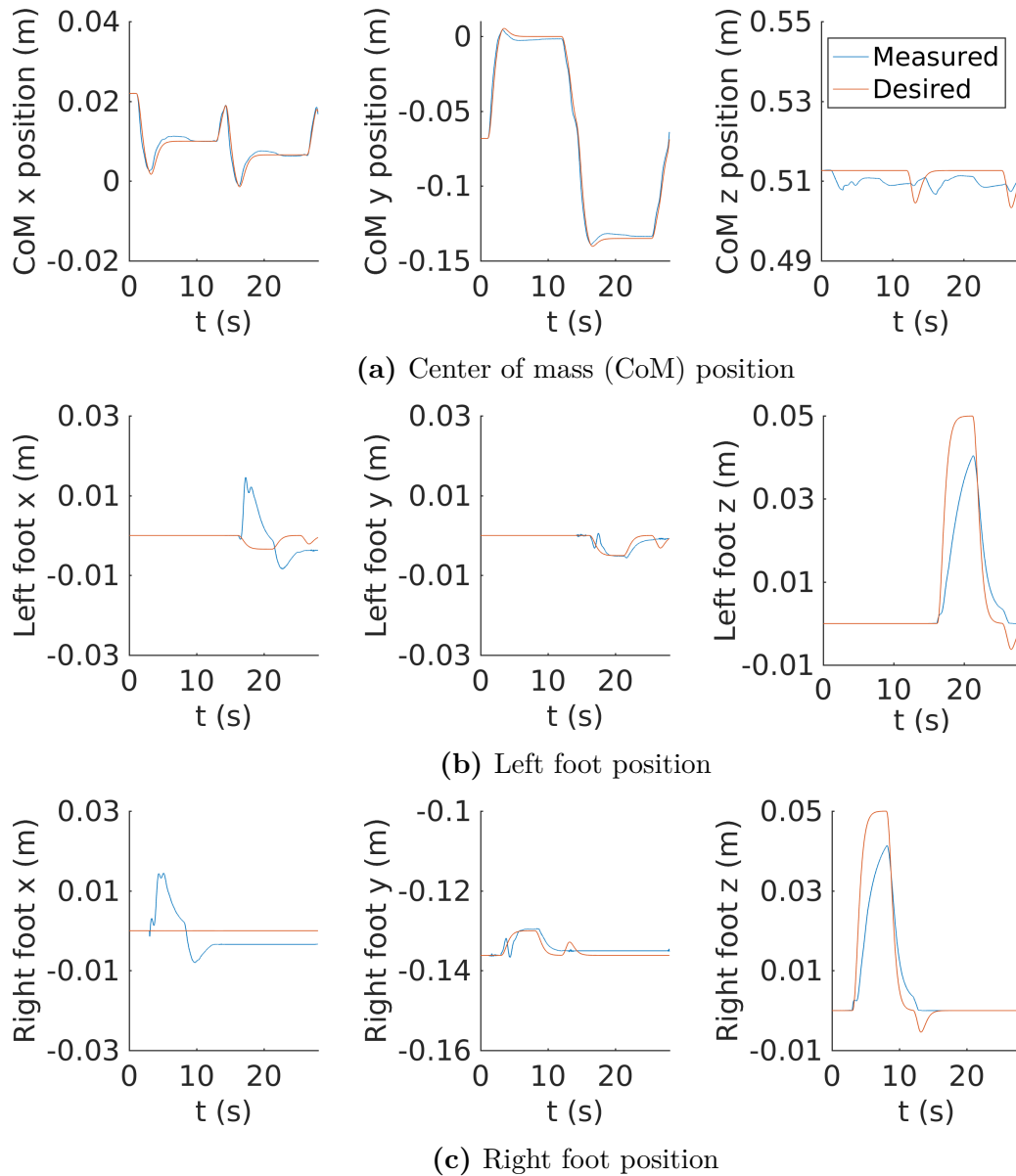


Figure 2.6 Evolution of position tasks for a sample of 1 stride, achieved in simulation with the **soft tasks controller #1**. Position values are given with respect to a world frame of which the x , y and z axes correspond respectively to the sagittal, frontal and vertical axes. Achieved trajectories are shown in blue, while the desired ones are shown in red.

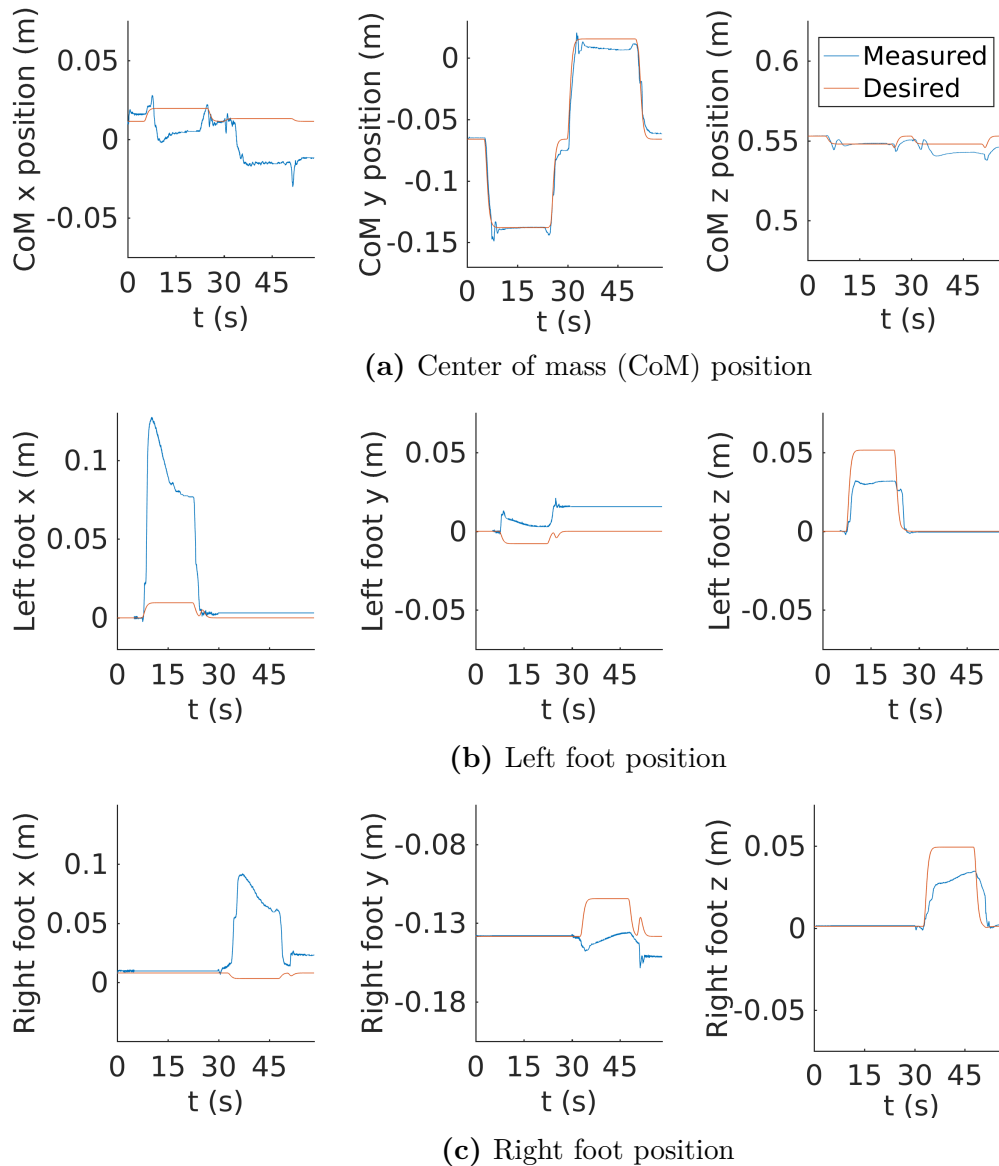


Figure 2.7 Evolution of position tasks for 1 stride performed in real-world experiments with the **soft tasks controller #1**. Position values are given with respect to a world frame of which the x , y and z axes correspond respectively to the sagittal, frontal and vertical axes. Achieved trajectories are shown in blue, while the desired ones are shown in red.

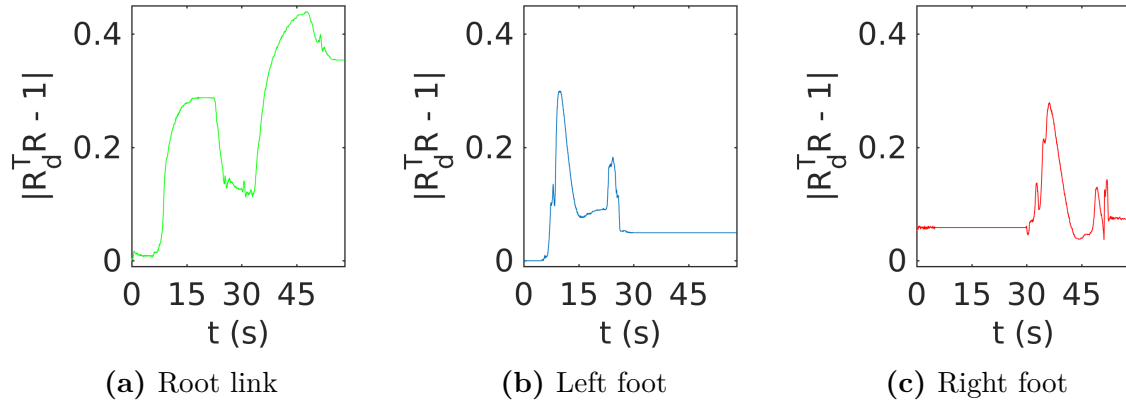


Figure 2.8 Evolution of orientation task errors for 1 stride performed in real-world experiments with the **soft tasks controller #1**.

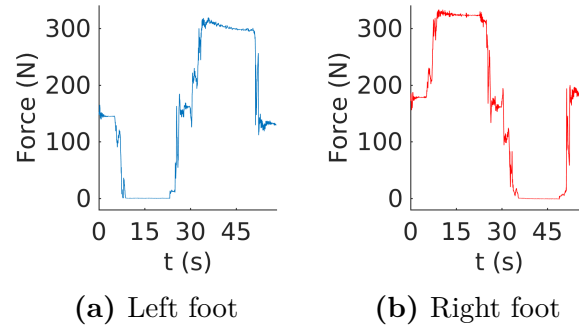


Figure 2.9 Evolution of estimated vertical contact forces on robot feet, for 1 stride with the **soft tasks controller #1**.

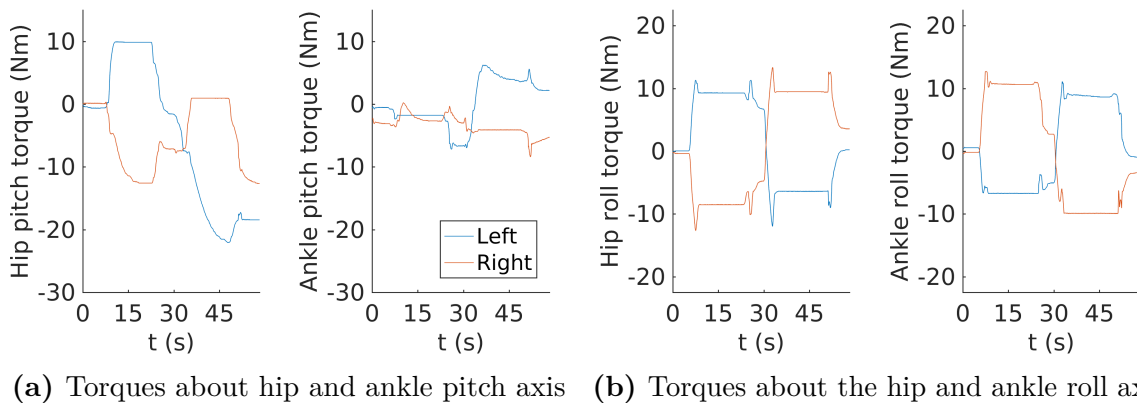


Figure 2.10 Evolution of estimated joint torques for 1 stride with the **soft tasks controller #1**, on the left and right hip and ankle joints. Torques on the pitch and roll axes of these joints are critical for keeping balance in the performed experiment.

2.4.2 Walking in place with the soft tasks controller #2

Experiments with the soft tasks controller #2 have been conducted in simulation. Differences in the implementations of the finite state machines between controllers #1 and #2 account for slight differences in the experiments that have been performed. With the soft tasks controller #2, the foot is lifted 2.5 cm above the ground, and it is brought back down to the ground once it has reached its desired position (or a maximum time of 6 seconds had been spent on the task).

In total, the following number of parameters need to be adjusted for the controller:

- 4 parameters for the stabilization of contact tasks
- 6 task weights
- 8 parameters of the finite state machine
- 18 proportional gains of Cartesian tasks
- 13 proportional gains of the postural task

This list amounts to a total of 49 parameters which may be tuned for the controller. Derivative gains are all automatically set as twice the squareroot of the corresponding proportional gain, which amounts to a significant reduction in the number of parameters to adjust. Eventually, if tests on the robot show that we cannot count on the symmetry of the robot and the desired movement, 10 additional parameters may be needed to account for non symmetric proportional gains for the Cartesian and postural tasks.

Parameters of the controller which allow to successfully achieve the desired motion have been obtained through manual tuning. They are set as described in appendix B, which provides task weights, proportional and derivative gains defined for Cartesian and postural tasks of the controller, as well as parameters used with the finite state machine.

Control torques obtained from the quadratic programming solver are directly applied to the simulated robot. They are shown in figure 2.11 for the joints which are most critical for balancing: the hips and ankles. The graphs show that torques are contained within a feasible range, and they are relatively smooth, apart from the peaks which can be observed at foot liftoff and touchdown.

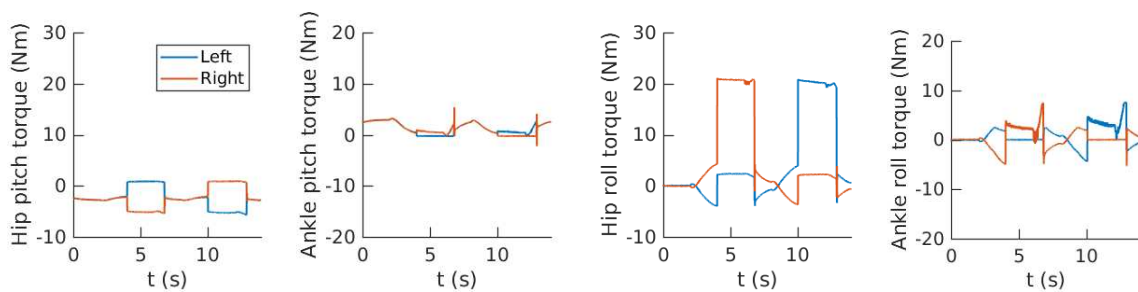
As shown in figure 2.12, contact forces obtained from the QP solver show to be relatively continuous and the measured contact forces show to follow closely the desired

values, with an error contained below 20 N at most, except at contact switching (which shows that the impact of the foot with the ground is not anticipated by the controller).

The robot behavior achieved in simulation is illustrated in figure 2.13: the robot begins in double support, then transitions to single support on the right foot, before lifting the left foot and lowering it back to the ground and repeating the same process on the other side. The robot can repeat the process of lifting one foot after the other practically indefinitely, without loss of balance: it has been validated in simulation for a minimum of 50 continuous cycles. However, the results presented here are limited to showing the first cycle.

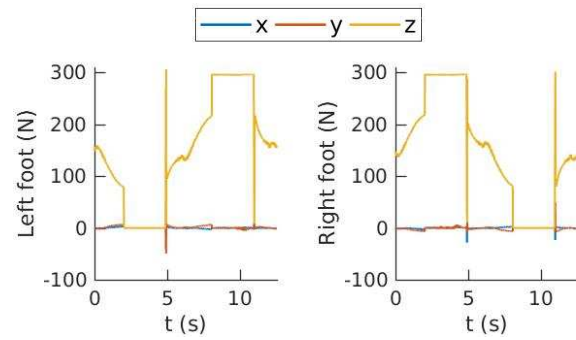
The trajectories obtained in simulation for the CoM and feet are shown in figure 2.14. The error on the CoM has been kept below 0.01 m at all times. As for the feet tasks, the largest measured error is of 0.04 m on the vertical axis, at the moment when the desired foot position is brought back on the ground. However, it is interesting to note that the desired vertical foot position does not reach the setpoint of 0.025 m. This is most likely due to the smoothing of the trajectory, and may be looked into, in a future iteration.

Tuning parameters would eventually allow to transfer results to the real robot.

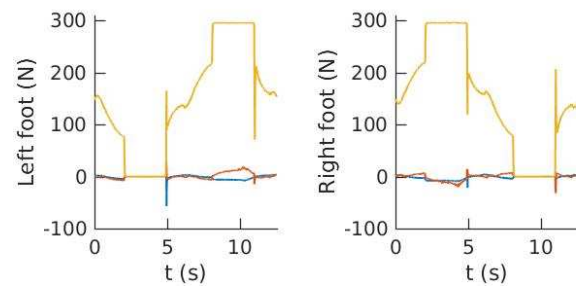


(a) Torques about hip and ankle pitch axis (b) Torques about hip and ankle roll axis

Figure 2.11 Evolution of joint torques measured in simulation for 1 stride achieved with the **soft tasks controller #2**, on the left and right hip and ankle joints.



(a) Contact forces obtained from the QP solver



(b) Estimated contact forces

Figure 2.12 Evolution of contact forces on robot feet, either obtained from the QP solver or as estimated from sensor measurements, for 1 stride with the **soft tasks controller #2**.

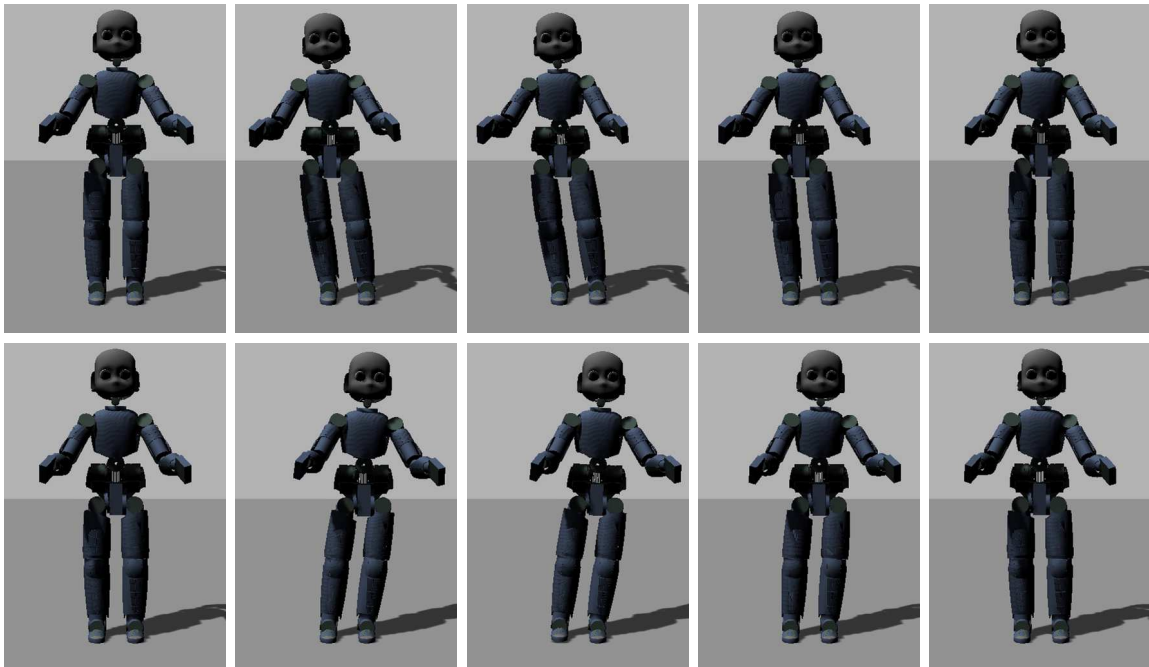


Figure 2.13 Walking in place motion achieved with the **soft tasks controller #2** in simulation experiments: lifting the left foot, then the right foot.

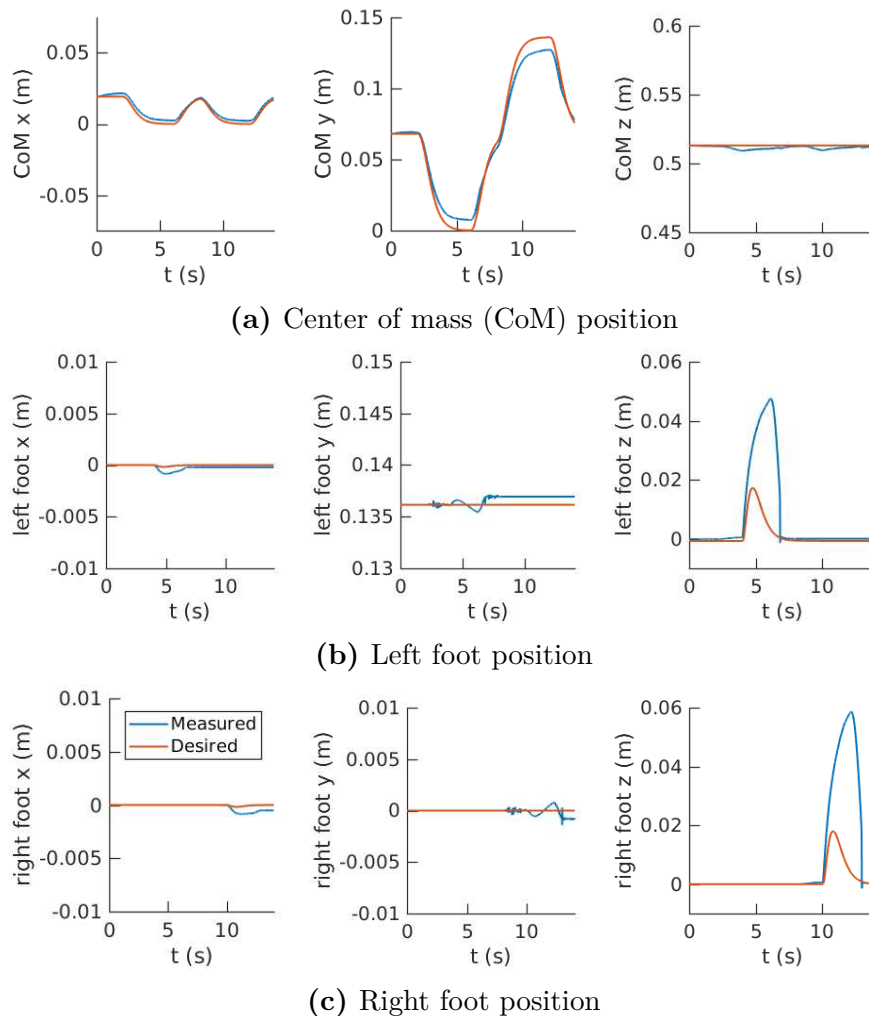


Figure 2.14 Evolution of position tasks for 1 stride of the simulated robot, achieved with the **soft tasks controller #2**. Position values are given with respect to a world frame of which the x , y and z axes correspond respectively to the sagittal, frontal and vertical axes. Achieved trajectories are shown in blue, while the desired ones are shown in red.

2.4.3 Discussion

Results show that the soft tasks controllers #1 and #2 can achieve balancing and contact switching with a torque-controlled, free-floating robot such as the iCub.

With the soft tasks controller #1, simulation results show that desired trajectories are closely followed. However, further work and experiments would be needed in order to improve the results obtained with the robot. It is highly likely that further parameter tuning would help achieve a behavior of the robot closer to the one obtained

in simulation. Furthermore, the robot used for experiments was new at the time the experiments have been conducted, and it probably needed further calibration and validation before achieving optimal results. For instance, we noticed that the performance of the robot may have been limited by issues related to the calibration of low-level torque-controllers, as well as the estimation of the CoM position, base pose, contact forces and joint torques.

Experiments performed with the soft tasks controller #1 show that the controller has a few drawbacks. For instance, it can get stuck in a state, when threshold conditions of the finite state machine are not getting met. We also found that the constraint on the rate of change of joint torques can be counterproductive for keeping balance, when dealing with switches in contact conditions. Since the rate of change of the contact forces themselves is not constrained in a similar way, then the joint torques may not always counterbalance the contact forces fast enough, compromising equilibrium and smoothness of robot trajectories. Furthermore, the controller assumes that input Cartesian postural task trajectories are feasible, and coherent with other tasks of the controller, which possibly limits the capacities of the controller. Measures to ensure coherence between tasks may eventually be beneficial. Finally, the soft tasks controller #1 requires a lot of parameters to be tuned. As a result, adjusting and testing the controller is significantly tedious.

The development of the soft tasks controller #2 allowed for improvements related to the issues mentioned in the previous paragraph, and to simplify the implementation of the controller, without compromising the results. It again allows to achieve balancing and contact switching of a floating-base robot, and the stepping behavior achieved with the simulated robot is similar as that obtained with the soft tasks controller #1.

However, when compared, results show that the movement of the robot achieved with the soft tasks controller #2 is faster: it takes about 14 s to achieve one stride, rather than 60 s with the soft tasks controller #1. Trajectories achieved with the soft tasks controller #2 on the simulated robot are also generally smoother, with smooth transitions of contact conditions. Finally, although the number of parameters has been drastically lowered in this second controller, it still requires a certain number of parameters to be tuned again for transferring results to the real robot.

2.5 Conclusion

In summary, this chapter has presented methods for developing whole-body torque-controllers, suitable for achieving a walking motion. Several iterations of a controller have been presented, showing how it can be simplified and improved, in order to achieve equivalent results, more easily.

Additional improvements may still be applied on the soft tasks controller #2 for a subsequent iteration. For instance, task weights have been kept constant in the scope of this work, but adapting weights over time, depending on the sequence of actions of the robot, could eventually be explored, in order to improve the global behavior of the robot [Liu et al., 2015; Modugno et al., 2016b].

The improvements from the soft tasks controller #1 (presented in section 2.3.2) to the soft tasks controller #2 (presented in section 2.3.3) allow for an easier implementation, while achieving similar results in simulation. However, it turns out that they do not quite solve the problem of transferring results from simulation to the real robot.

Several directions can be taken, in order to tackle this issue. First of all, it is highly possible that the challenges encountered when performing tests on the real robot are related to issues with force/torque sensors and estimation of the state of the robot. Further investigation on this subject could potentially make a significant impact on the results. However, this is not the focus of the present thesis.

Another possibility is to investigate ways to improve the robustness of the controller. One thing which has been noticed to be problematic when performing tests on the robot, is when a joint limit is reached. In the specific case of the iCub, safety measures implemented on the firmware of the robot take over, such that the joint becomes position-controlled and the torque-controller has no influence on this joint anymore. This is an important feature for the integrity of the robot, but it does not solve the fact that the torque-controller itself should avoid joint limits. In consequence, a method ensuring joint limit avoidance of torque-controllers may be developed. Chapter 3 shall focus on this subject, with the development of a joint limit avoiding method for torque-control, which is robust to external perturbations.

From another perspective, one may want to tackle directly the problem of the reality gap, which limits the transfer of results between simulation and real-world experiments. Notably, tuning the parameters of the controller shows to have an important impact

on the results, but generally, it is done manually and separately for each platform on which experiments are performed (i.e. for each simulation model and for each robot), as shown in chapter 4. This procedure can quickly become tedious if many parameters are involved. One way then to approach this problem could be to develop methods that automatically adjust the parameters of the controller in a robust way, such that the results can be more easily transferred between platforms, as presented in chapter 5.

Finally, one may also be interested in verifying that the whole-body controller is robust to different desired movements, such that given a set of parameters, the controller can achieve various movements. For instance, the controller presented here is specialized for the application of stepping in place, but it is likely that the controller will need to be adjusted again, when attempting walking movements. Chapter 6 is therefore investigating in this direction, by proposing a single framework for handling generic whole-body trajectories.

Chapter 3

Joint limit avoidance for torque-control

The whole-body torque-controllers presented in chapter 2 have shown to be effective in controlling the whole-body motion of a robot. However, they do not address motion constraints, such as ensuring joint limit avoidance.

When using position-control, joint limit avoidance can be achieved with constraints on joint positions. However, this method may be unsatisfying in the case of torque-control, as there is no theoretical guarantee that joints will be kept away from limits. Indeed, the inherent compliance achieved with torque-control makes it possible for joint limits to be reached in case of external perturbations.

The present chapter therefore proposes a solution in the form of a nonlinear control algorithm to ensure joint limit avoidance of a torque-controlled manipulator, which can then be applied to a whole-body torque-controller.

Joint limit avoidance is achieved by ensuring that the evolution of the joints always remains within the associated physical bounds. The essence of the proposed control algorithm is to parametrize the feasible joint space in terms of exogenous states, and then the control of these states allows for the achievement of joint limit avoidance. Stability and convergence, when the desired joint trajectories are feasible, are shown by means of an analysis based on Lyapunov theory.

The proposed method therefore defines nonlinear position feedback terms which can be used as a substitution of classical position correction terms when a desired

joint trajectory must be followed. The proposed control laws are reminiscent of those obtained by applying *barrier function*-based control approaches [Ngo and Mahony, 2006; Prajna and Jadbabaie, 2004; Tee et al., 2009; Wieland and Allgöwer, 2007], but they are derived from a different perspective.

Thanks to its formulation, our method can be applied to either fixed-base manipulators, or floating-base robots. For this reason, sections 3.1 and 3.2 first introduce the modelling of fixed-base robots, as well as a classical torque-controller for such a system, whereas the modelling of floating-base robots and whole-body torque-control were already presented in chapter 2. Then, the following sections describe the parametrization we propose, and the derivation of control laws for joint limit avoidance. The proposed methods are validated experimentally, first being implemented for the control of two DOFs on the torque-controlled iCub, and then within a whole-body torque-controller.

3.1 Modelling of fixed-base systems

The robot is assumed to be a multibody system composed of $n + 1$ rigid bodies, called links, connected by n joints with 1 DOF each. If one of the links has a constant pose with respect to an inertial frame \mathcal{I} , then this fixed link is referred to as the base, and the multibody system is considered as *fixed-base*. The configuration space of a fixed-base mechanical system can then be characterized by its generalized coordinates, e.g. the joint configurations in the case of a manipulator.

The Lagrangian derivation of the equations of motion of a robotic manipulator with n degrees of freedom yields a model of the following form [Siciliano and Khatib, 2007]:

$$M(s)\ddot{s} + C(s, \dot{s})\dot{s} + G(s) = \tau \quad (3.1)$$

where $s \in \mathbb{R}^n$ is the vector of generalized coordinates of the mechanical system, $M(s) \in \mathbb{R}^{n \times n}$, $C(s, \dot{q}) \in \mathbb{R}^{n \times n}$ and $G(s) \in \mathbb{R}^n$ are the inertia matrix, Coriolis matrix and gravity torques, respectively, and τ is the vector of input torques. The following properties of model (3.1) are assumed [Siciliano and Khatib, 2007]:

Property 1 *The inertia matrix M is bounded and symmetric positive definite for any s .*

Property 2 *The matrix $\dot{M} - 2C$ is skew-symmetric.*

3.2 Classical torque-control techniques for fixed-base systems

Let $s_d(t) \in \mathbb{R}^n$ denote a twice differentiable time function representing the desired trajectory for the joint configurations s . Throughout the chapter, we assume that:

Assumption 1 *The reference trajectory $s_d(t)$ is such that its first and second order time derivatives are well-defined and bounded $\forall t \in \mathbb{R}^+$.*

The control objective is then defined as the asymptotic stabilization of the tracking error \tilde{s} to zero, with \tilde{s} defined as follows.

$$\tilde{s} = s - s_d \quad (3.2)$$

To achieve this objective, classical control laws can be applied. For instance, passivity-based controllers are known to work robustly against modelling and actuation errors [Siciliano et al., 2008, ch. 8.5.1 p. 328], and the associated law is written as below.

$$\tau = M(s)\ddot{s}_d + C(s, \dot{s})\dot{s}_d + G(s) - K_P\tilde{s} - K_D\dot{\tilde{s}} \quad (3.3)$$

with K_P and K_D two symmetric, positive definite matrices representing proportional and derivative control gains. Applying control law (3.3) to system (3.1) results in bounded trajectories of the closed-loop dynamics and convergence of the tracking error to zero, for any initial condition $(s, \dot{s})(0)$.

3.3 Joint space parametrization

We propose a joint limit avoidance method based on a parametrization of the joint space, as explained in the following paragraphs.

Let $s_{min}, s_{max} \in \mathbb{R}^n$ denote the vectors defining the minimum and maximum values of the joint coordinates s . We define the feasible space \mathbb{S} for the joint coordinates as:

$$\mathbb{S} := \{s \in \mathbb{R}^n : s_{min_i} < s_i < s_{max_i} \quad \forall i = 1, \dots, n\}. \quad (3.4)$$

The control objective is then the global asymptotic stabilization of the tracking error (3.2) to zero, while ensuring that

$$s(t) \in \mathbb{S} \quad \forall t \geq 0 \quad (3.5)$$

To ensure that the variable s always belongs to \mathbb{S} , one may parametrize the feasible configuration space. Let $\xi \in \mathbb{R}^n$ denote an exogenous variable. Then, we propose here to consider the following parametrization of the space \mathbb{S} :

$$s(\xi) := \delta \tanh(\xi) + s_0 \quad (3.6)$$

with

$$s_0 := \frac{s_{max} + s_{min}}{2} \quad (3.7)$$

$$\delta := \text{diag} \left(\frac{s_{max} - s_{min}}{2} \right) \quad (3.8)$$

where $\text{diag}(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ is the operator that, given a vector $x \in \mathbb{R}^n$, returns a diagonal matrix having on the diagonal the elements of the vector x , and $\tanh(\xi) : \mathbb{R}^n \rightarrow \mathbb{R}^n$. As a consequence of the hyperbolic function nature, one clearly has that

$$s(\xi) \in \mathbb{S} \quad \forall \xi \in \mathbb{R}^n \quad (3.9)$$

We now make the following assumption.

Assumption 2 *Each joint coordinate s_i possesses a free motion domain different from zero, i.e.*

$$s_{max_i} - s_{min_i} > 0 \quad \forall i = 1, \dots, n \quad (3.10)$$

and the reference trajectory $s_d(t)$ is feasible, i.e.

$$s_d(t) \in \mathbb{S} \quad t \geq 0 \quad (3.11)$$

As a consequence of the above assumption, one can evaluate the desired trajectory $\xi_d(t)$ for the variable ξ via equation (3.6), i.e.

$$\xi_d(t) := \tanh^{-1} \left(\frac{s_d(t) - s_0}{\delta} \right) \quad (3.12)$$

and define the tracking error as

$$\tilde{\xi} := \xi - \xi_d \quad (3.13)$$

The main idea presented in this chapter is to conceive feedback control laws for the asymptotic stabilization of $\tilde{\xi}$ to zero, which, relying on the nature of the parametrization (3.6), would imply that $s(t) \in \mathbb{S} \forall t \geq 0$.

Now, it is observed that the relationship (3.6) can be viewed as a change of variable $\xi \rightarrow s$. So, the equation of motion (3.1) can be written in terms of ξ . To this purpose, note that

$$\dot{s} = J(\xi)\dot{\xi} \quad (3.13a)$$

$$\ddot{s} = J(\xi)\ddot{\xi} + \dot{J}(\xi, \dot{\xi})\dot{\xi} \quad (3.13b)$$

with $J \in \mathbb{R}^n$ a diagonal matrix of which the i -th element is given by

$$J_i(\xi) = \delta_i(1 - \tanh^2(\xi_i)), \quad (3.14)$$

and

$$\delta_i = \frac{s_{max_i} - s_{min_i}}{2} \quad (3.15)$$

It is important to observe that if Assumption 2 holds, which implies that $\delta_i \neq 0 \forall i$, then

$$\det(J(\xi)) \neq 0 \quad \forall \xi \in \mathbb{R}^n. \quad (3.16)$$

3.4 Joint space control with joint limit avoidance

Taking advantage of the parametrization defined in section 3.3, the next subsections present and discuss control laws for stabilizing a desired joint trajectory $s_d(t) \in \mathbb{S} \forall t$ that ensure joint limit avoidance. They are first derived for fixed-base systems, and then generalized for floating-base systems.

3.4.1 Joint limit avoidance for fixed-base systems

As long as the joint configurations belong to \mathbb{S} , the equations of motion (3.1) can be written as

$$M_\xi(\xi)\ddot{\xi} + C_\xi(\xi, \dot{\xi})\dot{\xi} + G_\xi(\xi) = \tau_\xi \quad (3.17)$$

with

$$M_\xi(\xi) = J^T(\xi)MJ(\xi) \quad (3.17a)$$

$$C_\xi(\xi, \dot{\xi}) = J^T(\xi)(M\dot{J}(\xi, \dot{\xi}) + CJ(\xi)) \quad (3.17b)$$

$$G_\xi(\xi) = J^T(\xi)G \quad (3.17c)$$

$$\tau_\xi = J^T(\xi)\tau \quad (3.17d)$$

Observe that the matrix $J(\xi)$ is bounded for any ξ . Then, it is straightforward to verify the following two properties of model (3.17), which reflect properties 1 and 2 of model (3.1) as introduced in section 3.1.

Property 3 *The inertia matrix M_ξ is bounded and symmetric positive definite for any ξ .*

Property 4 *The matrix $\dot{M}_\xi - 2C_\xi$ is skew-symmetric.*

Let us then remark an important fact. Once the system dynamics (3.1) is transformed into the form (3.17), any controller ensuring that the variable ξ is bounded would also imply that the joint trajectories belong to the feasible joint space \mathbb{S} . For instance, the computed-torque-like control strategy of equation (3.3) can be applied assuming τ_ξ as control input. This would ensure that ξ is bounded and, in turn, that $s(t) \in \mathbb{S} \forall t$.

Extending the passivity-based control strategy (3.3) to system (3.17) requires some close attention. The major technical difficulties reside in the fact that the variable change $\xi \rightarrow s$ is not one-to-one for any $s \in \mathbb{R}^n$, in the sense that if s is outside the feasible joint space, then $\nexists \xi$ such that $s = s(\xi)$. This implies that the matrix M_ξ tends to zero when joint trajectories approach their limits. The extension, however, is presented in the next lemma.

Lemma 3 *Assume that Property 1 and Assumption 1 hold. Apply to system (3.1) the following control law:*

$$\tau_\xi = M_\xi \ddot{\xi}_d + C_\xi(\xi, \dot{\xi}) \dot{\xi}_d + G_\xi(\xi) - K_P \tilde{\xi} - K_D \dot{\tilde{\xi}}. \quad (3.18)$$

Then, the following results hold.

1. The equilibrium point $(\tilde{\xi}, \dot{\tilde{\xi}}) = (0, 0)$ of the closed-loop dynamics (3.17)-(3.18) is globally asymptotically stable;
2. If $s(0) \in \mathbb{S}$, then $s(t) \in \mathbb{S} \forall t \geq 0$.

Proof is given in appendix C. The control law (3.18) ensures that the joint evolutions $s(t)$ belong to the feasible space \mathbb{S} for any time t , provided that the initial condition $s(0)$ belongs to this space.

The proof of this law exploits the passivity of the system dynamics expressed by Properties 3 and 4, and it must deal with the additional technicality that the mass matrix M_ξ tends to zero when the joint evolutions get closer to the joint limits. Observe the similarity between the control laws (3.18) and (3.3). All these similarities constitute the interest of the proposed parametrization (3.6).

The control torques τ can be directly evaluated from (3.18) and (3.17), that is

$$\tau = MJ(\xi) \ddot{\xi}_d + (M\dot{J}(\xi, \dot{\xi}) + CJ(\xi)) \dot{\xi}_d + G - J^{-1}(\xi) K_P \tilde{\xi} - J^{-1}(\xi) K_D \dot{\tilde{\xi}}. \quad (3.19)$$

Therefore, note that the similarities between the control laws (3.19) and (3.3) increase when the reference trajectory is a set point, i.e. $\dot{\xi}_d = \ddot{\xi}_d = 0$, which implies that

$$\tau = G - J^{-1}(\xi) K_P \tilde{\xi} - J^{-1}(\xi) K_D J^{-1}(\xi) \dot{s} \quad (3.20)$$

$J(\xi)$ being positive definite, one can choose the following control gains without compromising stability and convergence.

$$K_P = J(\xi) K'_P \quad (3.21)$$

$$K_D = J(\xi) K'_D J(\xi) \quad (3.22)$$

where $K'_P > 0$ and $K'_D > 0$

Then, in the case of set points, the main difference between classical control algorithms and the proposed control solutions resides in the feedback position terms:

$$\tau = G(s) - K'_P \tilde{\xi} - K'_D \dot{s}, \quad (3.23)$$

although theoretical guarantee of the stability and convergence of the control law (3.23) is missing at this point.

Equation (3.23) suggests that given the classical control scheme (3.3), joint limit avoidance can be attempted by substituting the feedback correction term

$$- K_p \tilde{s} \quad (3.24)$$

with either

$$- J^{-1}(\xi) K_P \tilde{\xi} \quad (3.25)$$

or

$$- K_P \tilde{\xi}, \quad (3.26)$$

since the associated control laws can be shown to ensure joint limit avoidance. This is a general procedure that may be attempted any time joint limits must be taken into account, and the control laws contain feedback position terms.

Remark The implementation on a real platform of the control law (3.23) requires close attention since it involves singularities of the variable ξ . These singularities may cause high-value for the torque input. Then, we suggest to use properly defined saturation functions to avoid explosions of the variable ξ depending on the torque limits of the underlying platform. Simulations and experiments we carried out, however, tend to show that the feedback correction terms (3.26) do not cause sharp, disruptive variations of the control variable τ , and we thus suggest the use of (3.26) over the other presented control laws.

3.4.2 Joint limit avoidance within a whole-body torque-controller

The control laws proposed in section 3.4.1 can be integrated within a whole-body torque-controller. In this case, reference accelerations for postural tasks are not computed with a classical PD control law, but may instead be obtained through the proposed feedback control policy of equation 3.25:

$$\ddot{s}^* = \ddot{s}_d - J(\xi)^{-1}K_p\tilde{\xi} - J(\xi)^{-1}K_dJ(\xi)^{-1}\tilde{\dot{s}} \quad (3.27)$$

3.5 Implementation for a fixed-base manipulator

The proposed control law (3.18) can be directly applied to the control of a fixed-base manipulator. For this purpose, the leg of the iCub can be considered as such: fixing the root link (pelvis) to a pole allows it to serve as a fixed base.

The following subsection therefore exposes how the method proposed in subsection 3.4.1 has been validated, with experiments using the leg of the iCub.

3.5.1 Application of joint limit avoidance for a torque-controlled iCub leg

The proposed control law (3.18) was developed in Matlab/Simulink using WBToolbox [Romano et al., 2017], and can be used either for a simulated or a real robot. In this case, real-world experiments have been performed with the iCub, using its leg as a 2-DOF manipulator, in order to verify the convergence and stability properties of the approach. Experiments have also allowed to observe the compliance and robustness obtained with the proposed control law.

The 2-DOF manipulator is composed of rotational joints at the hip (joint 1) and knee (joint 2) pitch: it is illustrated in figure 3.1. Joints are bounded within limits set to $[-30, 85]$ degrees for the hip and $[-100, 0]$ degrees for the knee, such that

$$s_{min} = \begin{bmatrix} -30 \\ -100 \end{bmatrix}$$

$$s_{max} = \begin{bmatrix} 85 \\ 0 \end{bmatrix}$$

Joint torques obtained from the control laws for the hip and knee pitch are stabilized by a low-level joint torque-controller. All other joints of the robot are kept fixed with a position controller. As discussed in the remark at the end of section 3.4, to avoid singularity issues, saturation has been defined for the variable ξ at a value of 100.

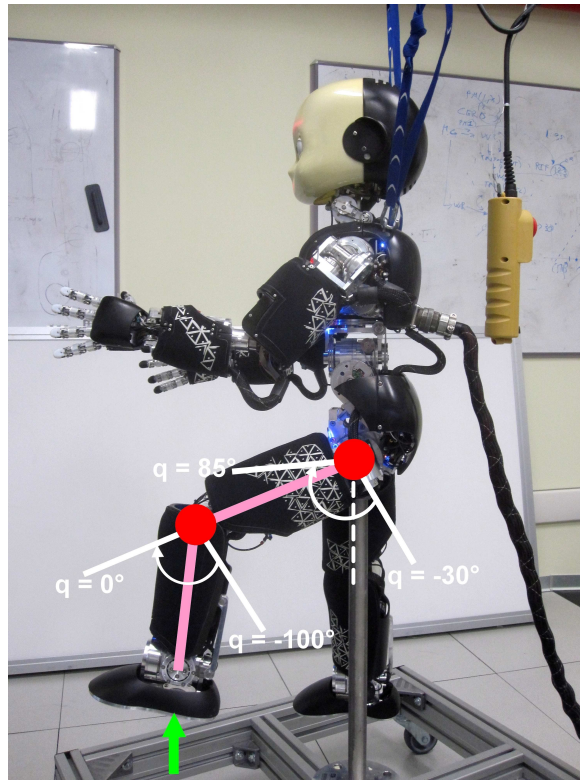


Figure 3.1 iCub leg setup used for the experiments. The red circles identify the hip and knee joints, while the white marks indicate joint limits. The green arrow shows the external force applied in Experiment 3.

Note that a small approximation in the control laws has been made, due to limitations of the WBToolbox software: it allows for the evaluation of bias forces $G(s)$ and $C(s, \dot{s})\dot{s}$ acting on each joint. However, it does not allow for the computation of the term $C(s, \dot{s})$. As a solution, $C(s, \dot{s}_d)$ and $C(s, J\dot{\xi}_d)$ are used in (3.3) and (3.19). The impact of this approximation is minor, since joint velocities used in the experiments are small, C is kept to a low value and the tracking errors \tilde{s} and $\tilde{\xi}$ are kept small.

We have performed three different experiments:

1. Validate that the proposed control law allows to prevent overpassing joint limits due to overshoot, by moving the leg to a constant desired position.
2. Verify the effectiveness of the method in tracking time-varying sinusoidal joint reference positions.
3. Assess the robustness and compliance achieved with the proposed control law, by subjecting the robot to unknown external perturbations.

The following subsections expose each of these experiments, before discussing the results.

Experiment 1 - constant reference position

The first experiment is set with initial position and velocity

$$s(0) = \begin{bmatrix} -14 \\ -60 \end{bmatrix}$$

$$\dot{s}(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

where positions are expressed in degrees. The task then consists in reaching a constant joint reference position

$$s_d = \begin{bmatrix} -18.5 \\ -10.0 \end{bmatrix}$$

The proportional and derivative gain matrices K_p and K_d are chosen as diagonal matrices with stiffness and damping values of

$$K_p = \begin{bmatrix} 20 & 0 \\ 0 & 10 \end{bmatrix}$$

$$K_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Damping gains are zero, since the motors of the robot already provide damping.

Figure 3.2 shows the evolution of the joint positions and control torques obtained during the experiment. Overshoot causes the knee joint to overpass its limit when

using the classical passivity-based law (3.3), while the knee joint remains within limits when using the proposed control law (3.19).

Note that the iCub platform is equipped with a low-level torque-control loop in charge of stabilizing any desired joint [Fumagalli et al., 2012, 2010]; it compensates for friction effects, but with some imperfections, and some viscous friction remains present. The fact that the tracking error does not converge to zero is thus mainly due to imperfect tracking of this low-level loop and to unmodeled friction effects.

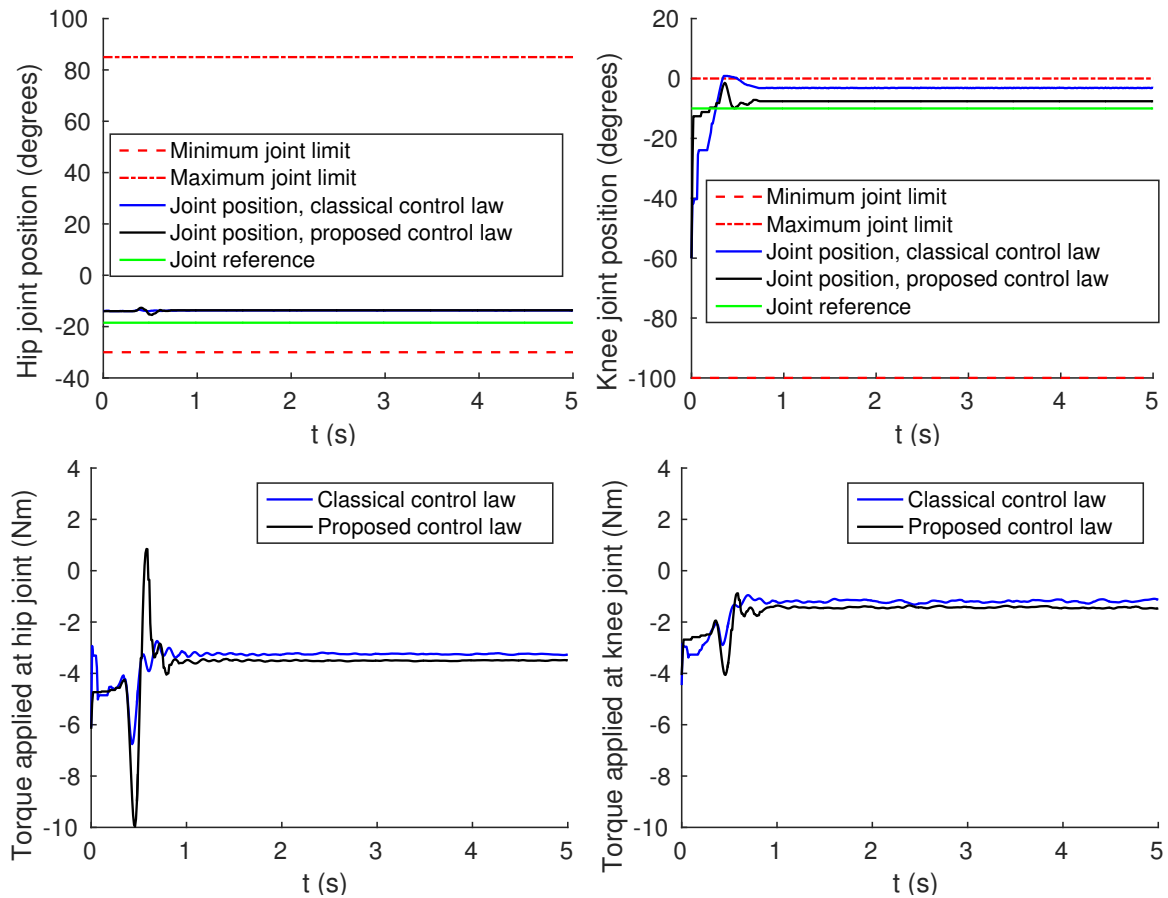


Figure 3.2 Hip and knee joint trajectories and torque, resulting from Experiment 1, presented in section 3.5.1.

Experiment 2 - sinusoidal reference position

The second experiment performed consists in tracking time-varying sinusoidal joint reference positions of the form

$$s_d(t) = \frac{\delta}{r} \sin(\omega t + \rho) + s_0 \quad (3.28)$$

Parameters of (3.28) are set to the following for the hip joint.

$$\begin{aligned} r &= 1.1 \\ \omega &= 0.25 \\ \rho &= 0 \end{aligned}$$

For the knee joint, the following parameters are instead used.

$$\begin{aligned} r &= 1.1 \\ \omega &= 0.65 \\ \rho &= -\pi/2 \end{aligned}$$

Once again, initial conditions are set as

$$\begin{aligned} s(0) &= \begin{bmatrix} -14 \\ -60 \end{bmatrix} \\ \dot{s}(0) &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

The proportional and derivative gain matrices K_p and K_d are chosen as diagonal matrices, as follows.

$$\begin{aligned} K_p &= \begin{bmatrix} 68 & 0 \\ 0 & 17 \end{bmatrix} \\ K_d &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{aligned}$$

The evolution of the joint positions and torques are shown in figure 3.3. Results are very similar between both control laws tested. However, it can be observed that

with classical control law (3.3), the knee joint limit is exceeded at times 7 s, 11.5 s and 19 s. On the other hand, with the proposed control law (3.19), the joint trajectories are kept within joint limits throughout the experiment.

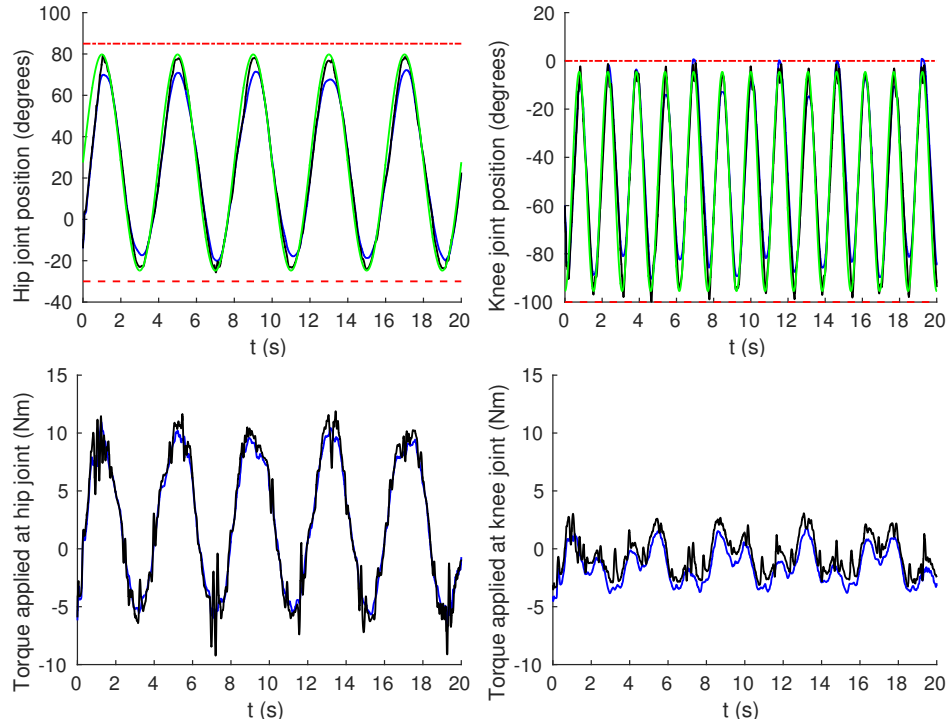


Figure 3.3 Hip and knee joint trajectories and torques, resulting from Experiment 2, presented in section 3.5.1. Refer to figure 3.2 for legend: green lines are used for the reference joint trajectories, blue lines denote results of classical control law and black lines results of proposed control law.

Experiment 3 - robustness to external perturbations

In the third and last experiment presented here, the initial position and velocity are set as

$$s(0) = \begin{bmatrix} -14 \\ -60 \end{bmatrix}$$

$$\dot{s}(0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The task then consists in tracking a constant joint reference position

$$s_d = \begin{bmatrix} 80 \\ -60 \end{bmatrix}$$

The proportional and derivative gain matrices K_p and K_d are chosen as the following diagonal matrices.

$$K_p = \begin{bmatrix} 68 & 0 \\ 0 & 17 \end{bmatrix}$$

$$K_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

External forces are then applied on the foot of the robot through physical interaction with a human. The experimenter is exerting pushing forces on the heel of the foot, with increasing strength. These applied forces are equivalent to applying an upward vertical force on the leg. As a result, the knee extends and the hip opens upward, moving both joints towards their limits. The positions reached by the robot are shown in figure 3.4, as well as the process of applying forces.

Figure 3.5 shows the evolution of the hip joint position and control torque, as well as the estimated vertical force applied on the foot during the experiment. Contact forces and joint torques are estimated using the force-torque sensor present in the leg of the robot, following a procedure based on a recursive Newton-Euler algorithm, as exposed in section 2.4.1.

Using the classical control law (3.3), when a force of 50 N is applied, it is sufficient to move the hip position over its limit. On the other hand, when using the proposed control law (3.19), the robot remains compliant to the applied external forces, but a larger force of 160 N needs to be applied in order to overpass the hip joint limit. Indeed, when a joint is near its limit, it could be noticed that its movement towards the limit is stiffer, but the robot remains compliant to the forces applied by the experimenter.

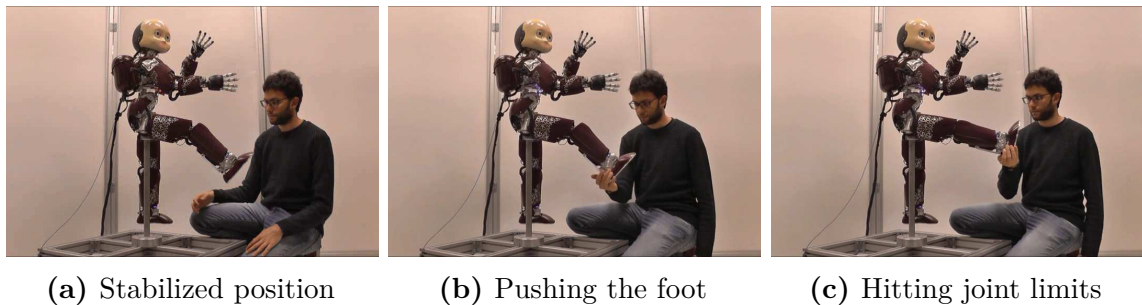


Figure 3.4 Pictures taken while performing Experiment 3.

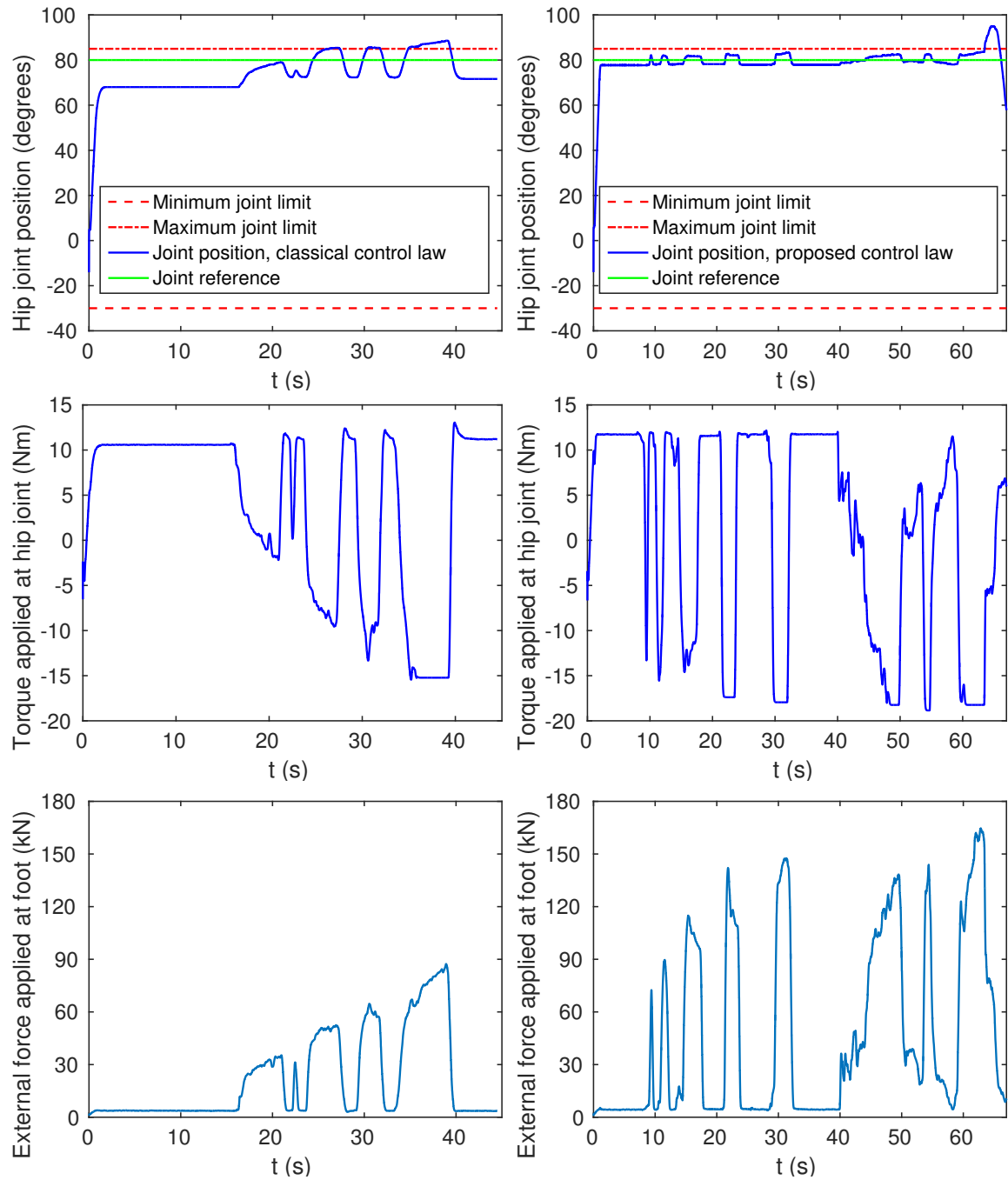


Figure 3.5 Hip joint trajectories and estimated joint torques, as well as estimated external forces applied on the foot, for the Experiment 3 presented in 3.5.1. On the left: results obtained with classical control law. On the right: results obtained with the proposed control law.

3.5.2 Discussion

The proposed approach has been validated for a torque-controlled fixed-base manipulator: it allows the asymptotic stabilization and convergence of a joint reference trajectory, while ensuring that the joint positions remain within their associated feasible range.

Stability and convergence of the tracking error have been shown by analysis based on Lyapunov theory, in appendix C. Then, the approach has been verified experimentally by torque-controlling 2 DOFs on the leg of the iCub. When compared with a classical passivity-based control law, the proposed approach shows higher robustness to external perturbations, without loss of compliance. In experiments, the controlled robot can resist, without overpassing joint limits, to the application of external forces 3 times larger than when controlled with a classical passivity-based control law.

The control law (3.19) shows to effectively generate torques that keep a joint away from its limits, partly due to the marked increase of the parameter ξ when nearing a joint limit. However, it is also partly due to the marked increase in the value of the term $J(\xi)^{-1}$ premultiplying the feedback gains in equation (3.19). The latter has the effect to increase the stiffness of the robot when nearing joint limits, further contributing to their avoidance.

In our experience, when torque-controlling the iCub, we have noticed that using high feedback gains produces a shaky behavior of the robot. The experiments performed to validate the proposed approach have shown to cause variations in gains sharp or disruptive enough as to partially create such a behavior only in a particular case: when a joint that is already kept close to a limit (e.g. through its desired position) is pushed further towards it. It may then be appropriate to ensure that desired joint positions are generally kept at a safe distance from joint limits, or to use a saturation function in order to limit this effect.

In essence, the approach consists in a change of variables, which makes it general enough to be applied to any torque-controlled robot subject to joint limits. It can therefore be advantageous to extend and implement this approach for whole-body torque-control.

3.6 Implementation of joint limit avoidance within a whole-body torque-controller

In order to verify the proposed approach for a whole-body torque-controller, we introduce the feedback control policy of equation (3.27) into an optimization-based controller. Figure 3.6 shows an overview of the approach we use for this purpose.

To validate the approach, the soft tasks controller #2 of chapter 2 is applied to the same stepping in place scenario, but in this case, external perturbations are applied to the robot in the shape of forces which may cause a joint limit to be violated. The following subsections shall provide a reminder of the implementation of the controller, before describing its application to testing the proposed method.

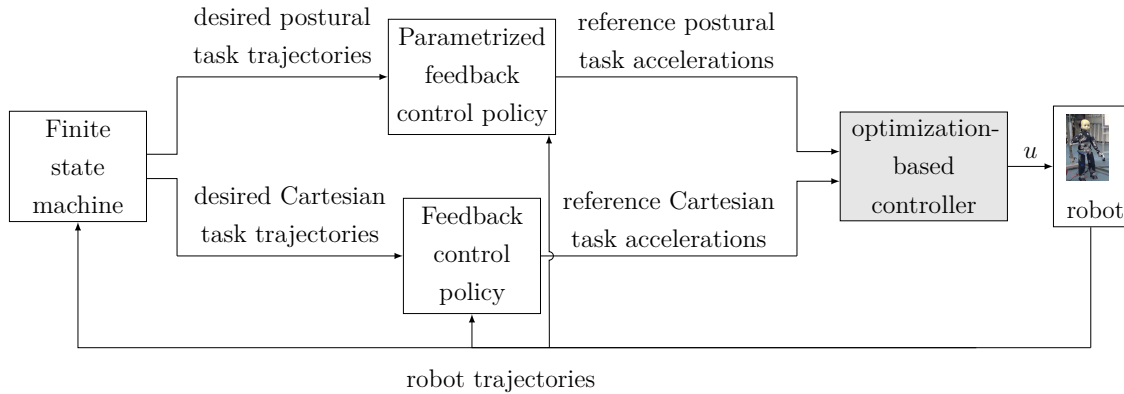


Figure 3.6 Overview of the proposed method for whole-body control with joint limit avoidance. A whole-body controller computes the control input achieving a combination of tasks as defined in the finite state machine.

3.6.1 Optimization-based controller

The controller used here is the same as the soft tasks controller #2 described in section 2.3.3. It defines an optimization problem, minimizing a weighted sum of squared errors on Cartesian task accelerations (CoM position, neck frame orientation, swing and stance feet pose), the squared error on postural task acceleration, and squared joint torques. The contact of the feet with the ground is stabilized with inequality constraints on the optimization problem. A control input, consisting of joint torques and contact forces, is obtained by solving this optimization problem.

For Cartesian tasks, the error on acceleration $\tilde{v}_{\mathcal{T}}(u)$ is obtained with

$$\tilde{v}_{\mathcal{T}}(u) = \dot{v}_{\mathcal{T}}(u) - \dot{v}_{\mathcal{T}}^* \quad (3.29)$$

in which the feedback term $\dot{v}_{\mathcal{T}}^*$ is computed with a proportional-derivative feedback control policy in $SE(3)$.

On the other hand, for the implementation of joint limit avoidance, the error on the postural task $\tilde{s}(u)$ is obtained with

$$\tilde{s}(u) = \ddot{s}(u) - \ddot{s}^* \quad (3.30)$$

in which the feedback term \ddot{s}^* is computed following the feedback control policy of equation 3.27.

The postural task is used to stabilize the motion by minimizing joint displacements from the initial pose of the robot, or user-defined desired joint positions. For Cartesian tasks instead, desired setpoints are obtained from a finite state machine. It is the same as the one introduced in section 2.3.3: it is composed of 5 states, defined in order to achieve a walking motion. The formulation of the state machine, desired trajectories defined for each task and state, as well as transitions between states are described in detail in appendix B.

3.6.2 Application of joint limit avoidance when walking in place

The controller described above is applied to the problem of walking in place, when subjected to external perturbations. The subsequent paragraphs discuss the conducted experiments and results achieved with the proposed method.

Experimental setup

The controller has been implemented in Matlab/Simulink using WBToolbox [Romano et al., 2017], and can be used either for a simulated or a real robot, although the experiment presented here has only been performed in simulation. Also, the open-source software package qpOASES [Ferreau et al., 2014] is used for solving the QP

control problem. This framework allows the implemented controllers to run in real-time, generating joint torque commands for the robot every hundredth of a second.

Experiments are performed on the iCub using 23 DOFs on legs, arms and torso, in order to validate empirically that the method described above allows for joint limit avoidance within a whole-body torque-controller. Table 3.1 lists the joint limits implemented on the iCub.

Parameters of the controller are defined in appendix B, with one exception: the reference position of the right arm is adjusted as follows. The elbow is extended along the body, to keep it only slightly bent at an angle of 22.5 deg (closer to its lower limit) as shown in the first image of figure 3.7.

Note that the initial and reference joint positions are all within their feasible range of motion. Note also that, as mentioned in remark 3, the value of ξ is saturated to $\tanh^{-1}(0.99)$, by allowing a maximal value of 0.99 to the expression $\frac{s-s_0}{\delta}$. This specific value has been attributed arbitrarily, and it could easily be adjusted to allow even more precise joint limit avoidance.

Then, external perturbations are applied with a certain number of pushing forces on the right hand of the robot, at different times during the experiment, as exposed in table 3.2. The interval of 5 seconds between the application of each push shows to be sufficient for the robot to recover from the previous perturbation. The application of these particular forces has for effect to push the right arm backwards, moving the elbow joint towards its lower limit.

Experimental results

Results achieved with the proposed feedback control law (3.27) are then compared with results achieved when using the classical PD feedback control law as recalled here: $\ddot{s}^* = \ddot{s}_d - K_p \tilde{s} - K_d \dot{\tilde{s}}$.

The robot behavior achieved in simulation, with and without joint limit avoidance, is illustrated in figure 3.7. In both cases, the robot succeeds in performing the desired stepping motion, but with slightly different reactions to external perturbations. Differences are evidenced in figure 3.8, where CoM trajectories are shown, as well as those of the joints that are more affected by the applied perturbations: elbow pitch, shoulder pitch and roll.

Table 3.1 Implemented joint limits of the iCub, in degrees

	Torso			Shoulder			Elbow	Hip			Knee	Ankle	
	pitch	roll	yaw	pitch	roll	yaw	pitch	pitch	roll	yaw	pitch	pitch	roll
q_{max}	50	30	70	10	160.8	80	106	85	80	70	0	30	20
q_{min}	-50	-30	-20	-95.5	0	-37	15	-30	-12	-70	-100	-30	-20

Table 3.2 Forces applied on the right hand of the robot

time (s)	force (N)	duration (s)
5	[15 0 0]	1
10	[20 0 0]	1
15	[-15 0 0]	1
20	[-20 0 0]	1

Forces are expressed with respect to world coordinates: the x -axis points forward.

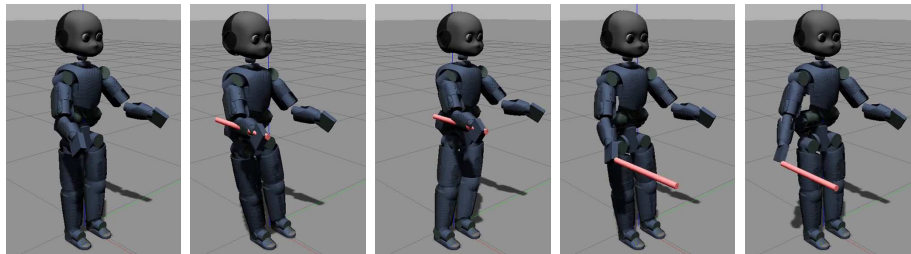
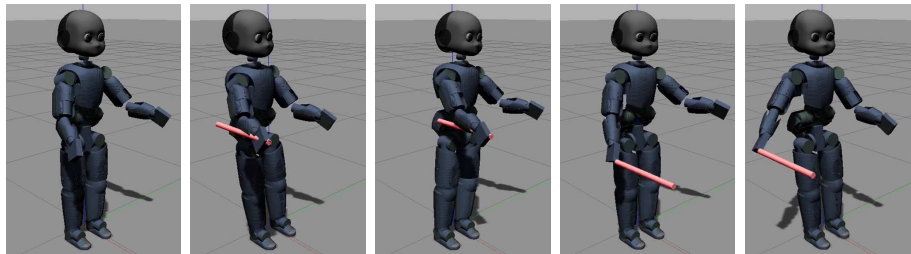
**(a)** With the classical feedback control policy**(b)** With the proposed feedback control policy

Figure 3.7 Behavior of the robot achieved when subjected to 4 increasing external perturbations (of 15, 20, -15 and -20 N, respectively) in simulation experiments: with the classical controller, compared with the proposed controller.

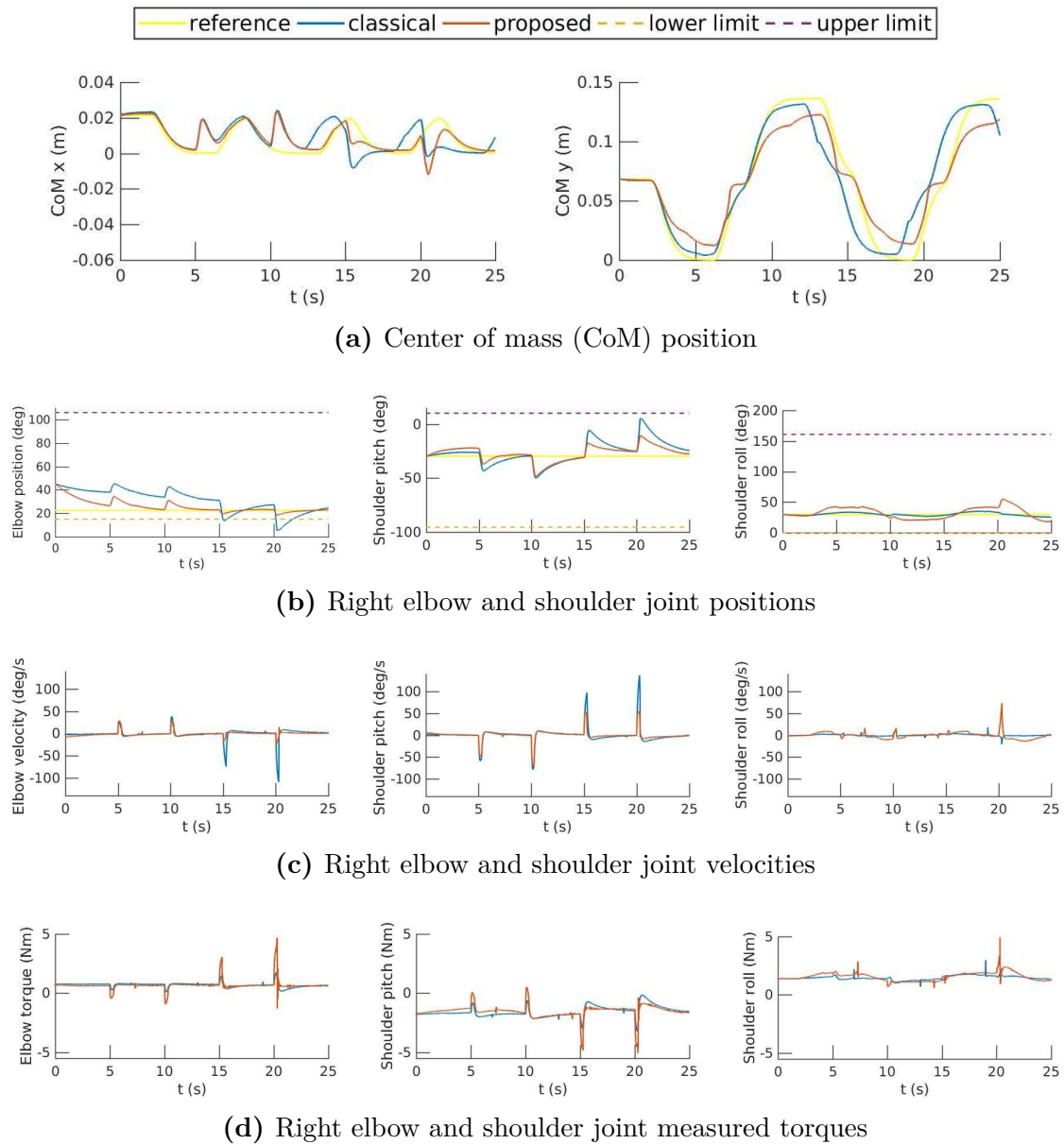


Figure 3.8 Evolution of CoM position as well as right elbow and shoulder joint positions, velocities and torques achieved during experiments with the simulated robot. CoM position values are given with respect to a world frame of which the x - and y - axes point respectively to the front and to the left of the robot. These graphs allow to compare results achieved with a classical controller and the proposed one. As depicted in the legend, reference positions, results achieved with the classical and proposed controller, lower and upper limits of the elbow joint are shown with different colored lines.

3.6.3 Discussion

The proposed approach has been validated with a whole-body torque-controller. Achieved results show stabilization and convergence of joint reference trajectories, while ensuring that the joint positions remain within their associated feasible range.

Using the classical feedback control law, we observe that when a force of -15 N is applied for 1 second on the hand, it is sufficient to move the elbow position over its limit. On the other hand, when using the proposed feedback control law (3.27), the robot remains compliant to the applied external forces, but a larger disturbance needs to be applied in order to force the elbow joint to its limit. However, note that even though the elbow is stiffer near its limit, the robot remains compliant to the applied forces. Hence, with the proposed approach, the robot can resist, without overpassing joint limits, to the application of external forces, contrary to a classical feedback control law, and this, without loss of compliance.

As for the CoM, results show that when forces are applied on the hand, the robot is destabilized more or less equally with both control laws compared. Thus, the proposed feedback control law does not appear to affect the ability of the whole-body controller to stabilize the robot.

Joint evolutions show that, when a disturbance that does not threaten to move a joint to its limit, both control laws generate similar joint trajectories. However, when the applied disturbance moves the elbow and shoulder pitch joints closer to limits, different behaviors are obtained. With the classical feedback control law, joint trajectories show large displacements of the elbow and shoulder pitch joints at a higher velocity, and the elbow joint limit is surpassed. On the other hand, with the proposed control law, displacements of the elbow and shoulder pitch joints are contained to prevent hitting a joint limit. Joints are moving at a fraction of the velocity displayed with the classical feedback control law. It can be observed that moving the shoulder roll is used as a way to absorb the larger disturbance.

In consequence, recovery from disturbances shows to be smoother with the proposed approach. Results indicate that the proposed approach allows to increase the robustness of a controller towards external disturbances such as unknown forces applied on the robot. Furthermore, while avoiding joint limits, the proposed control law also appears to reduce the velocity of the robot in reaction to external disturbances, thus making it safer for interaction with humans.

3.7 Conclusion

In summary, this chapter has presented new control laws for joint limit avoidance in torque-control. The proposed approach has been implemented for a fixed-base system, as well as within a whole-body controller. Results achieved with the fixed-base system, using the leg of the iCub, demonstrate the effectiveness of the approach for joint limit avoidance. Similarly, results achieved in simulation with the whole-body controller show that the approach successfully allows to avoid joint limits, thus increasing robustness in response to external perturbations.

The effectiveness of the method as it has been presented here relies on the shape of the hyperbolic tangent function. As such, additional parameters could be introduced into the parametrization, in order to adjust its specific shape. For instance, a scaling coefficient inside the hyperbolic tangent would allow to modulate how fast the function approaches infinity. By extension, it shall affect how close to joint limits do repulsive torques become effective. Furthermore, similar functions such as a sigmoid or arctangent could eventually be compared for their effectiveness and associated properties. A similar parametrization strategy could also be implemented, in order to contain joint motion within velocity limits.

Future works shall moreover include further testing of the joint limit avoiding feedback control policy. Due to time constraints, experiments on the real robot remain to be performed with the whole-body controller. The main limiting factor for the achievement of real-world experiments is the effort which needs to be spent adjusting the controller, in order to transfer results from simulation to the robot.

Developing methods to automatically adjust parameters of the controller would be helpful in this case, and possibly also for many more researchers, if the problem of parameter tuning is a general issue across projects involving QP-based controllers. Therefore, chapter 4 will assess the importance of this issue, from the experience of researchers working with similar controllers, as a first step before developing an automatic tuning method, which shall be presented in chapter 5.

Chapter 4

Survey on parameter tuning for QP-based controllers

In chapter 2, we have developed methods for whole-body torque-control. In this, the alliance of stack-of-tasks approaches and QP solvers have shown to be effective tools, simplifying the development of a controller.

Our general approach can be summarized as follows. The robot is modeled as a floating-base manipulator, with acceleration constraints at the contact of the feet with the ground. A stack-of-tasks is defined with the CoM position, feet pose, upper-body orientation and joint positions, allowing to track their acceleration. Reference accelerations, for their part, are computed using a proportional-derivative (PD) control strategy, given desired positions. The controller then relies on quadratic programming optimization to compute the joint torques and contact forces required for achieving the desired motions. The QP is formulated with strict and soft tasks, minimizing a weighted sum of task errors and a regularization term on the joint torques, given constraints on the strict priority tasks as well as feet/ground contact and support polygon. In addition to the desired trajectory of each task, variables of the controller include the priority or weights associated to each task, as well as PD gains used for computing reference accelerations.

Results achieved with our methods show that in simulation, the robot follows trajectories with enough precision to be able to walk in place practically indefinitely. However, on the real robot, although it succeeds in performing a walking in place motion, trajectories are followed with increasing errors and movements are slower.

The discrepancy between simulation and real world experiments has then been reported to be mostly related to the estimation of joint torques and floating base pose, which contribute to limiting the performance of the torque-controller.

Nonetheless, it can be observed that with a stack-of-tasks/QP-based controller, the number of parameters used to define desired trajectories, controller gains, as well as weights of tasks can rapidly increase with the complexity of the controller (e.g. number of tasks, number of states in the finite state machine, complexity of whole-body movements...). These parameters notably need to be specifically adjusted for a given robot platform, in order to achieve desired behaviors. Due to this fact, deployment on a real robot is not ensured to be straightforward. Indeed, when passing from experiments in simulation to the real-world robot, success or failure may highly depend on proper tuning of parameters, and not only on precise estimation of the state of the robot.

Considering that tuning is a significant part of a controller's success, given more precise tuning, a controller can be improved to achieve more impressive results.

Nonetheless, a number which so far has not generally appeared on paper is the time spent on tuning parameters of a controller. Indeed, it shows to be a significantly non-trivial task on its own, contributing to making the passage from simulation to experiments on the robot a challenge.

From anecdotal evidence, at the moment of writing this thesis, it does not seem so rare to find that, in the development of a new experiment with whole-body control, tuning of parameters is achieved over the course of several weeks, and is done by hand. This tuning may then additionally need to be corrected for any change in conditions or platform. As a result, parameter tuning can be considered a straining and thankless task, if one does not have the intuition for it. And when parameter tuning becomes tedious, effective tools would come in handy.

Therefore, it seems worth opening a discussion on the subject of parameter tuning, and investigating solutions to alleviate the difficulty of this task. At the moment of writing, only few papers have been published on the subject of parameter tuning for a humanoid robot [Modugno et al., 2017, 2016b; Pucci et al., 2016a]. Notably, a tool for automatic gain tuning of a whole-body torque-controller has been investigated in [Pucci et al., 2016a], but the solution calls for relatively large computations, which may not always be affordable for real-time applications.

Before delving deeper into this subject, we propose to interrogate the community working with similar controllers, to assess whether this is a problem which actually needs to be addressed.

In order to evaluate the interest of the community in the eventual development of solutions for the tuning of QP-based controllers, we prepared an anonymous online survey. The target population for this survey is therefore researchers who have experience in developing QP-based controllers. We can obtain a sample of this population through the users of related, well-established mailing lists in the European and worldwide scene [euRobotics, 2018; GdR-Robotique, 2018; robotics worldwide, 2018].

4.1 Description of the survey

The survey has been prepared in the form of a self-administered questionnaire using Google Forms, a free online software for building surveys. It was then distributed to respondents by email through the above-mentioned mailing lists. The email displayed in appendix D has been sent to each mailing list on June 13th, 2018.

The survey includes open-ended, close-ended and discrete questions, as well as rating scales. It is reproduced in its entirety in appendix D.

No particular screening procedures have been used, but we have informed respondents of the subject of the survey. In the email, we specifically invite those who have at any point worked with controllers based on quadratic programming (QP), to participate in the survey.

The survey questionnaire contains 5 pages.

1. The first page contains an introduction to the questionnaire, as well as the first question: “Are you using or have you been using QP controllers?”. If the given answer is “yes”, the questionnaire moves on to questions regarding QP controllers. Otherwise, questions on QP controllers are skipped and only demographic information (on page 5) is collected.
2. The second page defines the type of robot with which respondents have worked.
3. The third page aims to precise in which ways QP controllers have been used and defined by the respondents, and optionnally allows participants to indicate

- references in relation to the QP controller(s) they work(ed) with, or links to their source code.
4. The fourth page of the questionnaire contains the core of the questions related to QP controller tuning, meant to gather information from the experience of the respondents.
 - A first set of 3 questions assesses how important the respondents think it is to tune some specific parameters, if there are additional (unlisted) parameters which they think important to tune, and allows comments on the subject.
 - A second set of 3 questions assesses how much time the respondents spend tuning the same specific parameters as in the previous set of questions, if there is an additional (unlisted) parameter which they spend time and effort tuning, and allows comments on the subject.
 - Then, a series of questions assesses whether participants have developed or use tools or techniques for making tuning easier, if they find tuning tedious and if they think tools would be needed in that sense.
 - Finally, two open-ended questions allow respondents to share further information of feedback from their experience.
 5. The fifth and final page collects demographic information about the respondents.

4.2 Results of the survey

Data has been collected from 35 volunteer respondents. Note that although the present section reports the answers collected for the open-ended questions, the entirety of the free text answers are also provided in appendix D.

Information about the demographic profile of respondents is shown in figure 4.1. We may have sent reminders in order to increase the total number of respondents, but since the use of QP-based controllers is a rather precise field, we did not expect a large number of participants. The objective of the survey being to get an idea of the current situation, the number of responses was amply sufficient.

Out of the 35 respondents, 29 indicated that they are or have been using QP-based controllers and filled the related questions.

Most respondents (69%) have indicated to be working with a robot manipulator, while 37.9% have developed QP controllers with a humanoid robot. All the types of robots with which the respondents have worked are shown in figure 4.2.

Results shown in figure 4.3 indicate that in general, respondents do not limit their use of QP controllers to a single problem, nor do they always define them with the same kind of task prioritization scheme. References and links to source code shared by the respondents are compiled in table 4.3.

In average, answers show that tuning of gains, trajectories, task priorities and constraints are considered significantly important (5/7). On the other hand, respondents are in average neutral about the tuning of parameters that are intrinsic to a controller (e.g. threshold values) or external (e.g. contact properties), but the neutrality could also arise from the fact that the terms “intrinsic” and “external” are not extremely well defined. These findings are shown in figure 4.4. In addition, some participants pointed out that the following are also important to tune:

- regularization terms
- regularization parameters (e.g. tolerances, saturations...)
- sampling rate of the trajectory

As for additional comments, some participants had different insights on the subject of trajectories, commenting:

“Trajectories should not be tuned but should be the outcome!”

“I don’t consider “desired task trajectories” as part of QP tuning, but rather as part of motion planning/MPC.”

Responses on how participants evaluate the time spent tuning parameters are shown in figure 4.5. Controller gains and task priorities are tuned by the large majority of respondents (93% and 97%, respectively), and their tuning time can reach up to a month. Other parameters of the list (task trajectories, constraints, intrinsic and external parameters) are less likely to be tuned within the development of a new QP controller. However, tuning task trajectories turns out to be comparatively time consuming, with 21% of respondents indicating that they may spend over 6 months on the task.

Several respondents have provided additional comments regarding the question on time spent tuning parameters of QP controllers:

“What took the longest in terms of getting all of these things implemented was when you see some undesired behavior (like a foot bouncing off the ground on touch down) it can be difficult to figure out which part of the system to fix. It could be the foot trajectory, the trajectory following due to low gains, the trajectory following due to a bad damping estimate that gets fed in through inverse dynamics, maybe the objective weighting is too low, maybe we are switching from swing to stance too early/late. While this framework is incredibly flexible and useful it does inherently couple everything together so it can be difficult to tease things apart.”

“The crucial parts are to get the constraints right. And that changes with every task. However, if done smartly, they can be (partly) reused. Some parameters follow iterations (eg contact properties), and they can be short (weeks or months for modifying a manipulator’s end effector) or long (up to a year or longer for building a new robot). Many parameters are also dependent, so changing e.g. the compliance of the actuator triggers changes in other parameters. Therefore, most are continuously evolving and good sets vary from task to task.”

“It can be really fast on some mono robot manipulation scenario but can be really tedious for multi robot scenario.”

“Tuning time can vary extremely depending of the scenario.”

“I assumed a person is working on the robot every day, full time.”

As shown in figure 4.6, when asked whether they had developed or are using tools or techniques for easing the tuning process, most people (72%) replied that they do not. One respondent however commented

“We are just getting around to defining performance metrics such that you can quantify improvement while tuning... but currently it is all based on trying it on the robot and see how it looks.”

Among those who do use or have developed tools, the following were mentioned:

- “*see [Lober, 2017]*”
- “*We developed tools on top of RVIZ to graphically tune controller gain and tasks trajectory*”
- “*Basic zero-pole graphs or simulated time plots while tuning the low-level (PID) gains*”
- “*We are learning a mix of the controller -so best case, its parameters require no tuning- and the task Jacobian which takes care of some of the other parameters*”
- “*Mismatch Learning*”

The last comment on mismatch learning probably refers to something similar to [Koryakovskiy et al., 2018]. These comments show that out of the people who have developed tools, some are used for helping manual tuning, while others use a learning method to perform at least part of the tuning.

Answers show that while 21% of respondents evaluate parameter tuning moderately tedious, 62% identify it as positively tedious (and none indicated that it was not tedious), as shown in figure 4.7. The following comments have been collected, as to the reason for their evaluation on a scale of 1 (not tedious) to 7 (highly tedious):

- 1: No evaluation.
- 2: “*Once you work with it a while it certainly gets easier.*”
- 3: “*Parameters should have a physical meaning and should therefore be easy to tune. If that is not the case, your model is bullshit. Of course, it is not always easy to find the actual numbers for the physical parameters, but at least you can apply common sense to their ranges.*”
“*QP controller can be really tedious to tune if you use it with bad trajectory planning and try to play with tasks weights and gains to overcome this issue.*”
“*I usually find that tuning only the scale-of-order is enough to get a first set of working QP settings (this applies to both PD gain and weight tunings).*”
- 4: No comments.
- 5: “*Trial and error tuning is too long and often tuning one parameter will affect other tuned parameters.*”

- 6: *“A complex dynamical system may have too many parameters to tune, the possible combinations are endless and the combined effect of all parameters is not easy to predict.”*
- “Sometimes work for the configuration of the robot used during tuning, then fails for a different robot configuration”*
- 7: *“Application specific and mostly heuristic”*
- “Time spent tuning is unpredictable and always more than code development”*

Comments from those who find tuning to be positively tedious (above 4), bring attention to the fact that complex systems may require the tuning of many parameters which possibly do not act independently from each other, and which may produce different results with different robot configurations and different applications. As a result, a trial and error tuning procedure becomes time consuming and tedious, when one needs to (i) try many combinations of parameters, and (ii) repeat the tuning process for each change in working conditions of the controller.

Comments from the respondents who don't find tuning particularly tedious show that they have an approach to tuning in which the effect of each parameter is clearly defined, trajectory planning is well posed, or in which the accumulation of experience allows to build intuition for tuning. The first two approaches (clear definition of parameters, and ensuring suitable trajectories) are worth investigating, while the third one (building experience) unfortunately does not help tuning for those who do not yet have much experience or just lack the kind of intuition needed. Additionally, simplifying the system, minimizing the number of parameters, and developing a tuning technique which is not simply relying on manual trial and error, could possibly ease the problems reported above.

Similarly as in the previous question, when asked to evaluate the importance of making parameter tuning less tedious, 17% of respondents are neutral, and 65% rate it as positively important, as shown in figure 4.8. The following comments have been collected, as to the reason for their evaluation on a scale of 1 (not important) to 7 (crucial):

- 1: *“I think the main problem with parameter tuning is not that it might be tedious. The main problem is that one never knows when the current tuning is good enough.”*

- 2: No comment.
- 3: *“In our work the QP part was the most robust. Compared to the contact planning part it was really easy.”*
- 4: *“I see potential in there.”*
- 5: No comment.
- 6: *“It is tedious because it often takes a huge amount of time. If less tedious = less time, then I think it is really important.”*
“Easier implementation on industrial robots.”
“It’s never tuned correctly and/or for all applications.”
- 7: *“Save time and make QP’s useful for production systems.”*
“No one wants to tune parameters every time with a new platform.”

According to these comments, the interest of easing tuning lies in the possibility of saving time and making QP controllers more accessible to industrial contexts, as well as more easily successful across platforms and applications, and defining criteria on what is a satisfying tuning.

Furthermore, as shown in figure 4.9, when asked about their interest in an eventual tool for tuning QP controllers, 83% of respondents indicate to be more than moderately interested, by giving a score ≥ 5 on a scale of 1 (not interested) to 7 (very interested).

Respondents have expressed what they think of as an ideal tool with the following comments:

“It should take as input parameters that have a clear and predictable effect on the robot, then map these parameters to the ones that people usually tune, whose effect is neither clear or predictable.”

“Easy to use, good GUI.”

“Automatically move the robot and find the best PID values.”

“The ideal tool would not need to any major adaptation on the part of the controller itself.”

“It should rely on a dynamic simulator and on some machine learning (better if supervised learning, even if automatically supervised). The simulator replays the experiment systematically and tunes the gain. Some manual tuning will also be required on the real robot, but only at the end.”

“An easy to set-up tool that can be easily applied to different QP frameworks, and auto-tune the parameters quickly+safely.”

“The tool needs a way to guess what I’m looking for in terms of whole-body behavior. At first, I see two ways to do that:

- 1. The tool assumes the instantaneous QP cost function is the specification. It then goes on to play a motion following my desired tasks, and looks at the integral of this cost function over a given run. Then changes QP parameters, and tries again for a different run. This way, we know how to generate a dataset (parameters, costs); then we can follow a data-driven approach (a.k.a. "learning")! Expected pro: user can do something else while the computer is working. Expected con: computation time may be prohibitive; only applies to PD-gain tuning of a weighted QP.*
- 2. The tool does not assume a specification; rather, it generates a trajectory and asks me every time which one I find better. Expected pro: applies to both PD-gain and weight tunings; specification is implicit. Expected con: user spends brain time in the process, dataset will thus be smaller.*

These thoughts being sketched, the ideal tool in my opinion looks like this: the user specifies all of its cost terms (for a weighted QP: the expressions that are weighted; for a lexicographic QP: same across all layers), but not weights nor PD gains. The tool will generate several trajectories for, say, at most a week. As a final outcome, the user is presented with a cost-function design GUI:

- Input: cost function, i.e. weights on each cost terms and/or lexicographic separation between weighted layers.*
- Output: recommended set of PD gains, and statistics over each cost term: min/max/average/standard deviation of the cost value over a trajectory.”*

These comments outline the following desired qualities for a tool:

- ease of use
- rapidity
- safety
- automatic
- standalone (does not need to interfere with the controller)
- applicable to various frameworks

Some kind of learning scheme may be implemented for the task, and the tuning objectives may be specified by the user.

Finally, some further recommendations have been offered by a few participants, as follows:

“I think having more robustness w.r.t. a change of task is important. Usually, the hand-made tuning is really task-dependent.”

“A graphic “tutorial” on QP parameters could be useful for people getting started with QP controllers”

“PD-gain tuning is not so hard. The problem is, the user does not really know what she/he wants. Hence my hunch that the point is not helping users tune their QPs, but helping QPs tune their users!”

“Refer to works on machine learning for transferring from simulation to real robot.”

Some general comments to the survey have also been submitted, as follows:

“You should have included a “don’t know” answer to the question above (I am not sure about all).”

“I don’t tune the parameters, Phd students do, so do not consider my answers.”

This last comment may indeed indicate that more experienced researchers have a less focussed view on the subject of tuning. A deeper analysis could look for the influence of years of experience on tediousness and interest ratings. The same could be done to assess the influence of the type of robot or the specific use of QP controller on answers. However, considering that the goal of the survey is to evaluate the interest of the community in the eventual development of solutions for tuning QP controllers, the analysis done so far appears to be sufficient. The answers received show a marked interest in eventual tools for making the tuning of QP controllers easier.

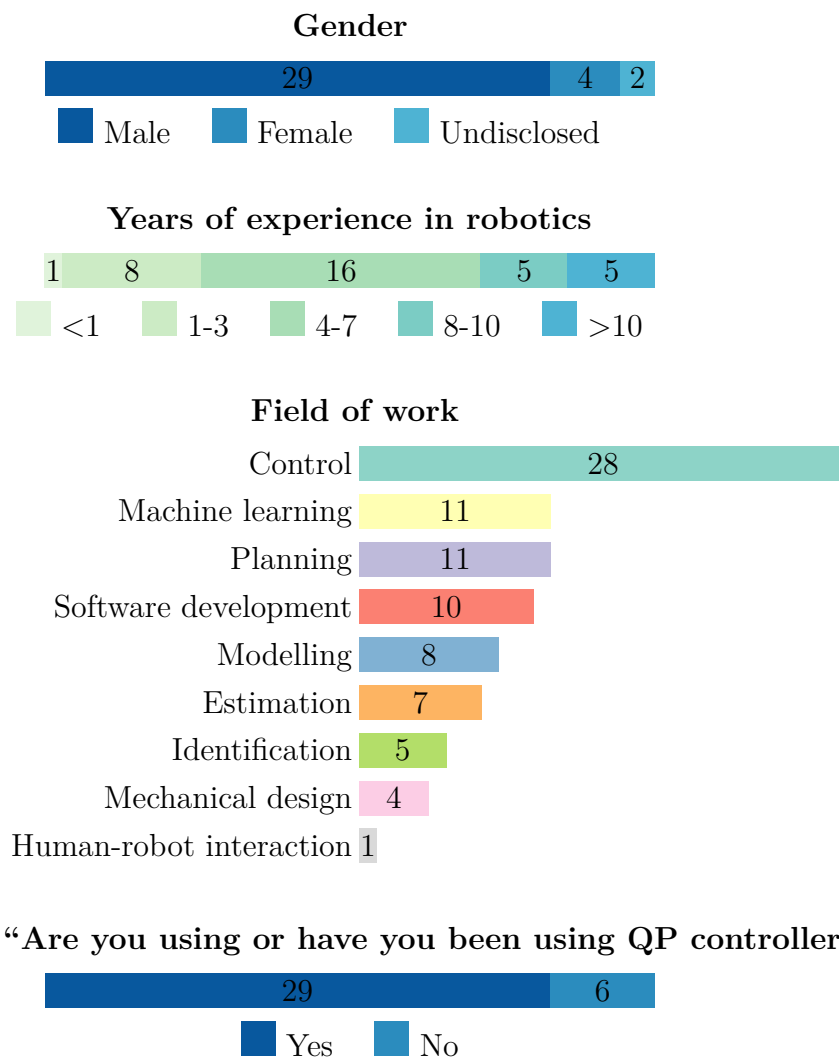


Figure 4.1 Demographic profile of the 35 respondents: gender, years of experience in robotics and related fields in which the respondents identified to be working

Answers to “Which type of robot are you developing your QP controller with?”

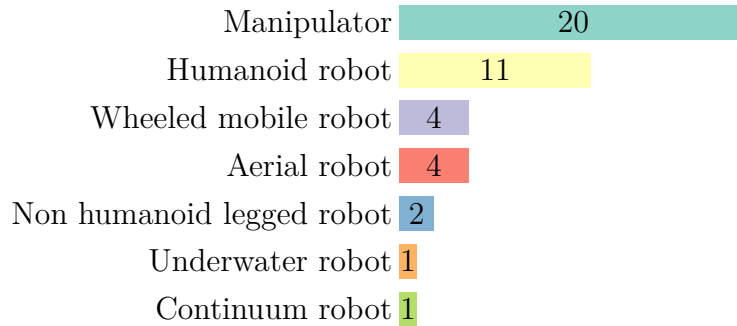
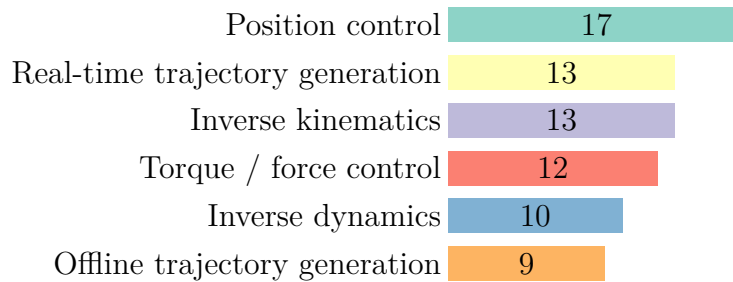


Figure 4.2 Types of robots which respondents are developing QP controllers with (respondents could select more than one type)

Answers to “In which way are you using QP controllers?”



Answers to “When multiple tasks are considered, various ways can be used to define a QP. What is your approach in this sense?”

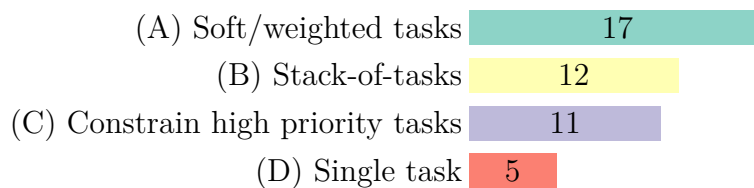


Figure 4.3 Use and formulation of QP controllers by the respondents (more than one type could be selected). More precisely, with (A) Soft/weighted tasks, we mean that the cost function is computed from a weighted sum of values associated to each task. With (B) Stack-of-tasks, we refer to a cascade of QPs solved with the lowest priority task acting in the nullspace of the highest priority tasks. With (C), we mean that high priority tasks are set as constraints of the QP while low priority tasks are considered into the cost function. (D) means that we only optimize over a single task.

Answers to “In your experience, how important is it to tune the following parameters associated to a QP controller?”

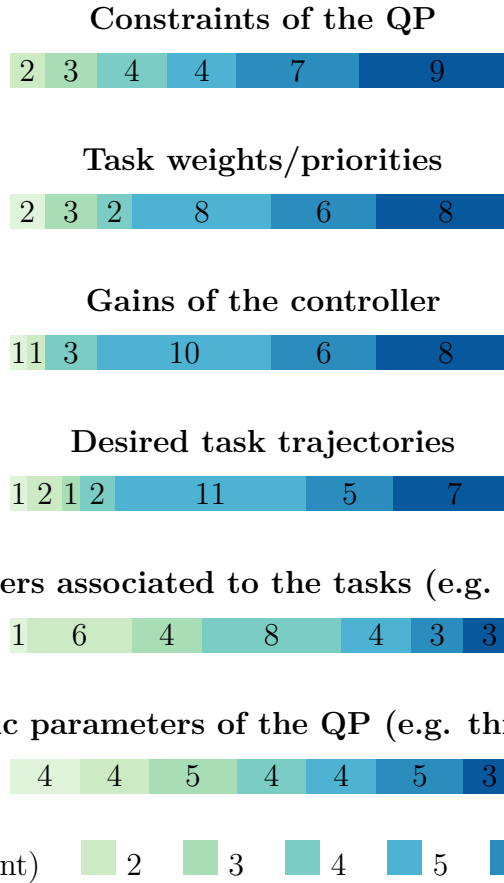


Figure 4.4 Respondents’ evaluation of the importance of tuning parameters, on a scale of 1 (no important impact on results) to 7 (crucial for successful results). Numbers in the graphs indicate the number of respondents who chose each category.

Answers to “From your experience, and please be honest: how much time and effort do you generally spend tuning the following parameters for a new QP controller?”

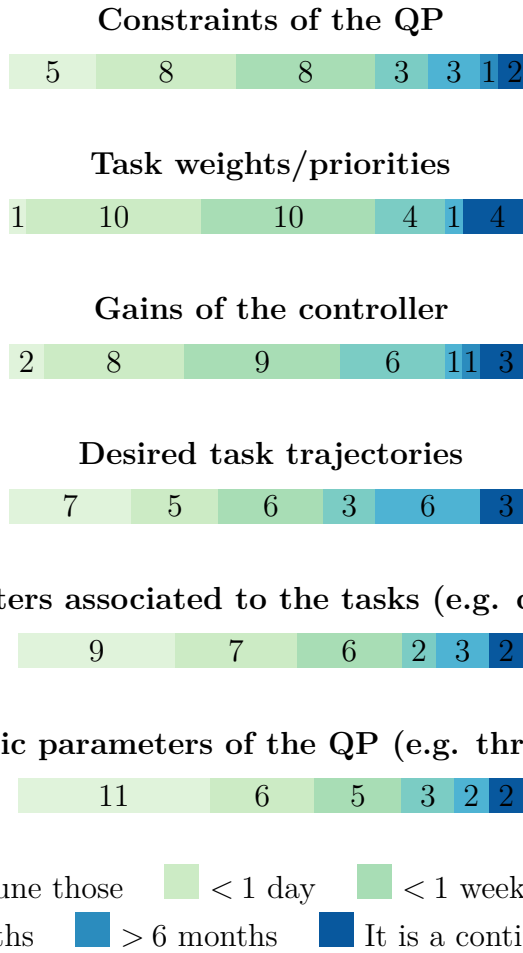


Figure 4.5 Time spent tuning parameters of QP controllers, according to respondents. Numbers in the graphs indicate the number of respondents who chose each category.

Answers to “Have you developed or are you using any tools or techniques for making the tuning process easier?”



Figure 4.6 Use of tools for tuning

Answers to “How tedious do you find parameter tuning to be?”

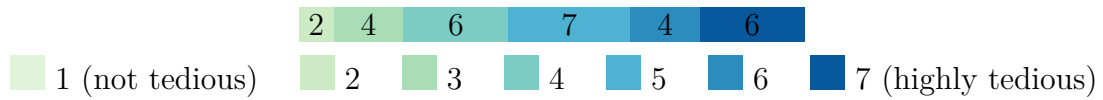


Figure 4.7 Respondents’ evaluation of how tedious tuning is, on a scale of 1 (It is not tedious, I find it rather enjoyable to tune parameters for QP controllers) to 7 (It is tedious enough to make me dislike tuning parameters)

Answers to “How important do you think it is to make parameter tuning less tedious?”

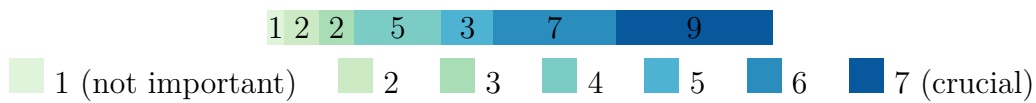


Figure 4.8 Importance of easing parameter tuning, according to respondents, on a scale of 1 (not important) to 7 (crucial).

Answers to “How would you rate your interest in an eventual tool for the tuning of your QP controller?”

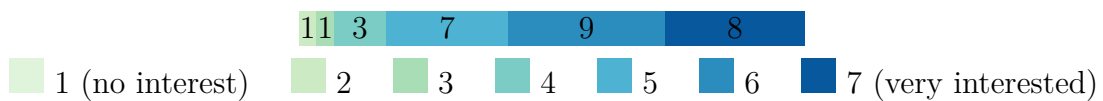


Figure 4.9 Respondents’ interest in eventual tools for tuning, on a scale of 1 (I am not interested in using tools for that) to 7 (I am very excited to try it out!)

Table 4.3 References and links to source code indicated by respondents

References to the software used

<https://projects.coin-or.org/qpOASES>

ORCA <https://orca-controller.readthedocs.io/en/dev/>

OpenSoT <http://opensot.github.io/> (not publicly accessible anymore)

MC-RTC

The QP used on HRP4 at LIRMM montpellier

Controller developed by Joseph Salini (ISIR) for XDE

Links to reference documentation

<https://hal.archives-ouvertes.fr/hal-01735462v1>

<http://www.roboticsproceedings.org/rss14/p54.pdf>

<https://ieeexplore.ieee.org/document/6482266/>

<https://hal.archives-ouvertes.fr/hal-01276931/document>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7433460>

Links to source code

<https://github.com/robotology/whole-body-controllers>

https://github.com/kuka-isir/cart_opt_ctrl

<https://github.com/jrl-umi3218/Tasks>

https://github.com/semroco/giskard_core

<https://github.com/semroco/giskardpy>

[https://github.com/robotology/walking-controllers/...](https://github.com/robotology/walking-controllers/)

<https://github.com/jrl-umi3218/Tasks/>

4.3 Conclusion

In this chapter, the results of a survey on the subject of parameter tuning for QP-based controllers have been presented. The answers received show that in general, significant time is spent tuning parameters for controllers, and this activity is generally considered tedious. Respondents also show a marked interest in eventual tools for making the tuning of QP controllers easier.

In response to this interest, it appears highly appropriate to investigate novel approaches for automatically tuning parameters such as task priorities or gains of a controller.

Chapter 5

Learning task priorities of whole-body controllers

The information gathered from the survey presented in chapter 4 exposes an interest of the community, in making the deployment of whole-body controllers easier. This is therefore the concern of the current chapter, aiming to define methods general enough to be applied to a wide range of optimization-based controllers.

In particular, data collected in chapter 4 shows that task priorities are generally considered to have a significant impact on the results achieved with a QP-based controller, but their tuning is rather tedious. There is therefore a marked interest in developing methods which would make the adjustment of task priorities less tedious, in order to save time and to make the implementation of controllers easier on different platforms or with different applications. An automatic method for finding the best prioritization is eventually necessary for robots to reach true autonomy.

Indeed, task priorities are usually designed *a priori* by experts, then manually tuned to adjust task ordering, timing, transitions, *etc.* It is a tedious operation which, depending on the complexity of the system and the task, can take a notable portion of a researcher's time, and can be particularly hard in the case of whole-body control of floating-base platforms. In particular, proper task priorities may not always be evident at first sight, especially when several tasks are used, or when working conditions may vary. In the case of whole-body control of floating-base platforms, the coordination of multiple tasks can be particularly hard, especially when it involves keeping balance, while navigating the environment or fulfilling additional activities such as upper limbs

and torso movements for manipulation. Another challenge with tuning parameters by hand, is that quantifiable metrics on performance, allowing to measure improvements while doing the tuning, still need to be defined. As a result, one may have difficulty discerning how changing a parameter affects the results, or even discerning when the current tuning is good enough.

Works related to this issue have been introduced in section 1.2.1, where in particular, the reality gap problem has been evidenced, and the need to achieve automatic parameter tuning. To achieve adaptability of learned solutions to new scenarios, the approaches presented in section 1.2.1 can benefit from *domain randomization* (DR) [Tobin et al., 2017], which consists in randomizing some aspects of the simulation to enrich the range of possible environments experienced by the learner. For example, in [Antonova et al., 2017], robust policies for pivoting a tool held in the robot’s gripper are learned in simulation, given random friction and control delays, such that the learned policies proved to be effective on the real robot as well.

The work presented in this chapter proposes to apply the idea of DR to whole-body controllers. In this context, we want to ensure that balance is maintained while performing a task, even if large differences exist between the learning domain and the testing domain, for example as illustrated in figure 5.1. To achieve this result, the idea is to combine a DR approach with fitness functions promoting the robustness of the learned controller. The developed method would then allow to learn, in simulation, robust task priorities which achieve desired goals, while allowing to facilitate the transfer of results from simulation to reality.

Since the proposed method relies on constrained stochastic optimization for learning task priorities, section 5.1 first introduces this subject, before section 5.2 describes the framework we developed. Its implementation for learning task priorities of a whole-body torque-controller is then presented in section 5.3, and its application to a stepping scenario is presented in section 5.4.

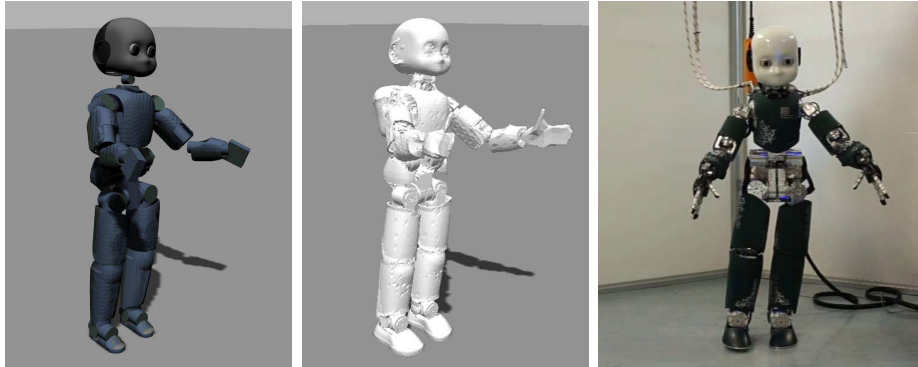


Figure 5.1 Different iCub models performing the same whole-body motion with several tasks. In this context, task priorities adjusted to achieve desired goals on a first iCub model may not allow to achieve the same results when used with different models, or when passing from simulated to real robot. With the purpose to eventually ease the passage from simulated to real-world robots, we propose to optimize the task priorities for robustness, in order to allow their transfer from one robot model to a different one, without the need of re-tuning.

5.1 Constrained stochastic optimization

One of the most common problems in applied mathematics and related fields is how to find an approximate of an optimal solution for a function defined on a subset of finite dimensional space. For instance, such an optimization problem is fundamental to many machine learning approaches. **Stochastic optimization** methods offer an approach to solve this kind of problem, and have become widely used in the last decades. More precisely, stochastic optimization refers to a collection of optimization methods that employ randomness for minimizing or maximizing an objective function. The randomness in these methods may be introduced in the formulation of the optimization problem, using random objective functions or random constraints to the problem.

Alternatively, and as will be the main focus here, randomness may be introduced into the search process of solving a problem. This allows to accelerate progress, can enable to escape a local optimum during the search to a global optimum, and can make the method less sensitive to modeling errors. For these reasons, this randomization principle represents a simple and effective way to achieve good performance of a stochastic optimization method, over many different problems. Methods which use this principle include for example simulated annealing, random search, as well as evolutionary algorithms such as genetic algorithms and evolution strategies. We will concentrate on the latter.

Evolution Strategies (ES) are a class of black box optimization algorithms, consisting in heuristic search procedures. The idea is the following. At every iteration of the algorithm (called a “generation”), a population of individuals containing candidate solutions (called “objective vectors” below) is perturbed (through mutation and recombination of their parameters), and the objective function value of each individual (called “fitness”) is evaluated. The individuals with the highest scores are selected and recombined with new candidate solutions to form the population of the next generation. Iterations are repeated until the optimization is considered done. Differences between algorithms of this class generally consist in different representations of the population, or different ways to perform mutation and recombination.

The most widely known evolution strategy is the **covariance matrix adaptation evolution strategy** (CMA-ES) [Hansen and Ostermeier, 2001]. In this approach, new candidate solutions are sampled from a multivariate normal distribution. The effect of recombination is to select a new mean value for the distribution, while that of a mutation is to add a random vector. Furthermore, a covariance matrix is used to represent dependencies between variables of the distribution. The concept behind CMA-ES is then to update the covariance matrix of the distribution.

Stochastic optimization is a whole field in itself, and could justify much more than the space which it is allotted here. The present section provides a functional explanation of the algorithm used in our work, but does not offer a deep explanation of how it works, as this is not the focus of this chapter. For further understanding directly related to this subject, the reader is encouraged to consult reference works [Arnold and Hansen, 2012; Hansen and Ostermeier, 2001; Igel et al., 2006].

In the work presented in this chapter, the learning problem is cast as a black-box constrained stochastic optimization. Given a fitness function $\phi(\mathbf{w}) : \mathbb{R}^{n_P} \rightarrow \mathbb{R}$ (with n_P the number of parameters to optimize), sets of n_{IC} inequality constraints and n_{EC} equality constraints h, g , the idea is to find an optimal solution $\mathbf{w}^\circ \in \mathbf{W} \subseteq \mathbb{R}^{n_P}$ to the problem:

$$\mathbf{w}^\circ = \arg \max_{\mathbf{w}} \phi(\mathbf{w}) \quad (5.1a)$$

subject to

$$g_i(\mathbf{w}) \leq 0 \quad i = 1, \dots, n_{IC} \quad (5.1b)$$

$$h_i(\mathbf{w}) = 0 \quad i = 1, \dots, n_{EC} \quad (5.1c)$$

As mentioned above, CMA-ES is a standard tool for solving black-box optimization problems. Each generation of the algorithm creates *offspring* objective vectors from the recombination of the objective vectors of the parent population, and mutation. This mutation comes from a random vector distributed according to a multivariate zero-mean Gaussian distribution \mathcal{N} , of which the covariance matrix \mathbf{C} is decomposed into Cholesky factors \mathbf{A} such that $\mathbf{C} = \mathbf{A}\mathbf{A}^\top$. An important feature of evolution strategies is that the covariance matrix is subject to adaptation: it is altered in order to sample more often steps in the search space that promise higher progress [Igel et al., 2006].

In particular, the CMA-ES algorithm has the advantage of having few parameters to tune. Even though the classical CMA-ES algorithm was not originally designed to solve constrained optimization problems, several variants which do handle constraints can be found in the literature. For instance, variants of CMA-ES have been benchmarked in [Modugno et al., 2016a], where (1+1)-CMA-ES with Covariance Constrained Adaptation (CCA) [Arnold and Hansen, 2012] is shown as the most appropriate variant for applications in robotics with several parameters and constraints. Therefore, we adopt this variant here.

As described in [Arnold and Hansen, 2012], this particular algorithm considers the constraints g_i and h_i , for which the normal vectors of constraint boundaries can be approximated in the vicinity of the current candidate solution. Then, the variances of the distribution \mathcal{N} in the directions of the normal vectors are reduced, as illustrated in figure 5.2. This is achieved by maintaining an exponentially fading record $\mathbf{v}_i \in \mathbb{R}^m$, $m = n_{IC} + n_{EC}$ of steps that have violated a constraint and updating Cholesky factors in function of it.

Furthermore, the algorithm requires that information about the successful iterations be stored in a *search path*, or *evolution path* $\mathbf{p}_c \in \mathbb{R}^{n_P}$.

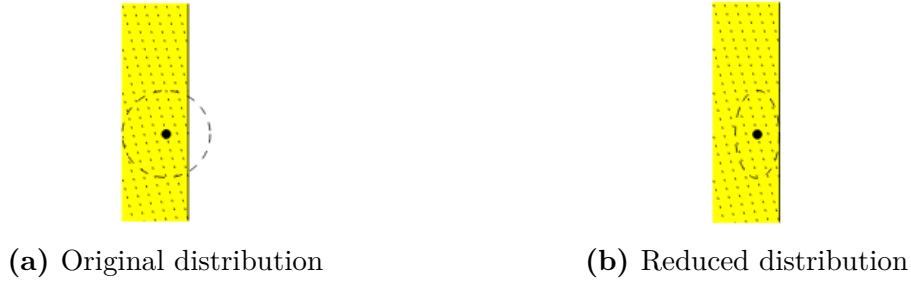


Figure 5.2 Effect of reducing the variance of the offspring distribution in the direction of the normal vector of a linear constraint boundary. In these images, taken from [Arnold and Hansen, 2012], the current candidate solution is marked with a black dot, the offspring distribution is indicated by a dashed circle, and the constraint boundary is indicated with a solid vertical line. While (a) shows the original distribution trespassing the constraint boundary, (b) shows the distribution reduced in the normal direction of the constraint plane, which allows it to remain within the feasible domain.

The state of the (1+1)-CMA-ES algorithm with CCA can be described by:

- the current candidate solution \mathbf{w} and its objective function value $\phi(\mathbf{w})$
- the five most recent ancestors of the current candidate solution \mathbf{w}_{-i} and their objective function value $\phi(\mathbf{w}_{-i})$, where $i = 1, \dots, 5$
- the global step size σ
- the success probability estimate P_{succ}
- the Cholesky factorization \mathbf{A}
- the search path \mathbf{p}_c
- constraint vector \mathbf{v}_j

After initialization, each iteration of the algorithm updates the above quantities with the following steps 1a to 1e:

0. **Initialization.** In this case, since we are applying the algorithm from [Arnold and Hansen, 2012], the parameter values are as suggested in this work. One may have a look at [Arnold and Hansen, 2012; Igel et al., 2006] for a detailed

explanation on the origin of the following values:

$$\beta = \frac{0.1}{n_P + 2} \quad (5.2a)$$

$$c = \frac{2}{n_P + 2} \quad (5.2b)$$

$$c_c = \frac{1}{n_P + 2} \quad (5.2c)$$

$$c_{cov}^- = \min \left(\frac{0.4}{n_P^{1.6} + 1}, \frac{1}{2|\mathbf{z}|^2 - 1} \right) \quad (5.2d)$$

$$c_{cov}^+ = \frac{2}{n_P^2 + 6} \quad (5.2e)$$

$$c_P = \frac{1}{12} \quad (5.2f)$$

$$d = 1 + \frac{n_P}{2} \quad (5.2g)$$

$$P_{succ}^{target} = \frac{2}{11} \quad (5.2h)$$

The following variables are then initialized:

$$\mathbf{A} = \mathbf{1}_{n_P \times n_P} \quad (5.3a)$$

$$\mathbf{p}_c = \mathbf{0}_{n_P \times 1} \quad (5.3b)$$

$$P_{succ} = P_{succ}^{target} \quad (5.3c)$$

$$\sigma = 0.1 \quad (5.3d)$$

$$\mathbf{v}_j = \mathbf{0}_{m \times 1} \quad (5.3e)$$

$$(5.3f)$$

The maximum number of iterations max_{iter} also needs to be chosen.

Finally, the initial candidate w must be chosen as a feasible solution (i.e. a solution which does not violate the constraints of the problem).

1. **Iterations.** Repeat the following steps, until terminating conditions have been met or the maximum number of iterations max_{iter} has been reached.
 - (a) Generate offspring candidate solution \mathbf{w}_1 according to $\mathbf{w}_1 = \mathbf{w} + \sigma \mathbf{A} \mathbf{z}$, where \mathbf{z} is a n_P -dimensional normally distributed random vector $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ with unit covariance matrix and zero mean.
 - (b) Check for constraint violations, and update v_j and A accordingly.

- i. Check for inequality constraint violations: determine whether $g_i \leq 0$ for $i = 1, \dots, n_{IC}$. If the j th constraint is violated, update constraint vector \mathbf{v}_j and constraint violation information $\mathbb{1}_{v_j}$, with $j = i$, according to

$$\mathbf{v}_j \leftarrow \begin{cases} (1 - c_c)\mathbf{v}_j + c_c\mathbf{A}\mathbf{z} & \text{if } j\text{th constraint violated} \\ \mathbf{v}_j & \text{otherwise} \end{cases} \quad (5.4)$$

$$\mathbb{1}_{v_j} \leftarrow \begin{cases} 1 & \text{if } j\text{th constraint violated} \\ 0 & \text{otherwise} \end{cases} \quad (5.5)$$

where the parameter $c_c \in (0, 1)$ determines how quickly the information present in the constraint vectors fades.

- ii. Check for equality constraint violations: determine whether $h_i = 0$ for $i = 1, \dots, n_{EC}$. If the j th constraint is violated, update \mathbf{v}_j , $\mathbb{1}_{v_j}$ according to equations (5.4), (5.5) with $j = n_{IC} + i$.
- iii. If $m_v = \sum_{j=1}^m \mathbb{1}_{v_j} > 0$ constraints were violated, update the Cholesky factor of the covariance matrix according to

$$\mathbf{A} \leftarrow \mathbf{A} - \frac{\beta}{m_v} \sum_{j=1}^m \mathbb{1}_{v_j} \frac{\mathbf{v}_j \mathbf{w}_j^\top}{\mathbf{w}_j^\top \mathbf{w}_j} \quad (5.6)$$

where the parameter β controls the size of the updates, and $\mathbf{w}_j = \mathbf{A}^{-1}\mathbf{v}_j$.

- iv. If $m_v > 0$ constraints were violated, jump to 1e.
- (c) Evaluate fitness $\phi(\mathbf{w}_1)$, update the success $\mathbb{1}_{succ}$, success probability estimate P_{succ} and global step size σ according to:

$$\mathbb{1}_{succ} \leftarrow \begin{cases} 1 & \text{if } \phi(\mathbf{w}_1) \geq \phi(\mathbf{w}) \\ 0 & \text{otherwise} \end{cases} \quad (5.7)$$

$$P_{succ} \leftarrow (1 - c_P)P_{succ} + c_P\mathbb{1}_{succ} \quad (5.8)$$

$$\sigma \leftarrow \exp\left(\frac{1}{d} \frac{P_{succ} - P_{succ}^{target}}{1 - P_{succ}^{target}}\right) \quad (5.9)$$

where $c_P (0 < c_P \leq 1)$ is the learning rate, P_{succ}^{target} is a target success rate, and d is a damping parameter that controls the rate of the step size adaptation.

- (d) Update Cholesky factor of the covariance matrix
- i. If $\phi(\mathbf{w}_1) \geq \phi(\mathbf{w})$, update current candidate \mathbf{w} , search path \mathbf{p}_c and Cholesky factor matrix \mathbf{A} :

$$\mathbf{w} \leftarrow \mathbf{w}_1 \quad (5.10)$$

$$\mathbf{p}_c \leftarrow (1-c)\mathbf{p}_c + \sqrt{c(2-c)}\mathbf{A}\mathbf{z} \quad (5.11)$$

$$\mathbf{A} \leftarrow \sqrt{1-c_{cov}^+}\mathbf{A} + \frac{\sqrt{1-c_{cov}^+}}{|\mathbf{w}|^2} \left(\sqrt{1 + \frac{c_{cov}^+|\mathbf{w}|^2}{1-c_{cov}^+}} - 1 \right) \mathbf{p}_c \mathbf{w}^\top \quad (5.12)$$

where the constants c and c_{cov} ($0 \leq c_{cov} < c \leq 1$) are the learning rates for the search path and the Cholesky factor matrix, respectively.

- ii. If $\phi(\mathbf{w}_1) < \phi(\mathbf{w})$, and if $\phi(\mathbf{w}_1) < \phi(\mathbf{w}_{-5})$, update Cholesky factor matrix \mathbf{A} according to

$$\mathbf{A} \leftarrow \sqrt{1-c_{cov}^-}\mathbf{A} + \frac{\sqrt{1-c_{cov}^-}}{|\mathbf{z}|^2} \left(\sqrt{1 + \frac{c_{cov}^-|\mathbf{z}|^2}{1-c_{cov}^-}} - 1 \right) \mathbf{A}\mathbf{z}\mathbf{z}^\top \quad (5.13)$$

- (e) This iteration is completed. Go back to 1a.

At the end of the iterations, \mathbf{w} contains the value of the optimized parameters, which are then used as the solution of the algorithm.

5.2 Framework for learning task priorities

The method proposed for learning robust task priorities is outlined in figure 5.3. It relies on two main parts:

- (i) an optimization-based whole-body torque-controller, using soft (weighted) task priorities to track desired task trajectories and send joint torque commands to the robot.
- (ii) a black-box constrained stochastic optimization procedure, that poses no restriction on the structure of the learning problem, as described in section 5.1. It is used to optimize task priorities: at the end of a rollout, the fitness of the obtained robot behavior is evaluated, and the optimization algorithm updates the task weights.

From there, the introduction of relevant randomized conditions under which the robot performs experiments allows to take advantage of domain randomization. Furthermore, the careful design of fitness functions allows the algorithm to optimize task priorities towards the desired goals.

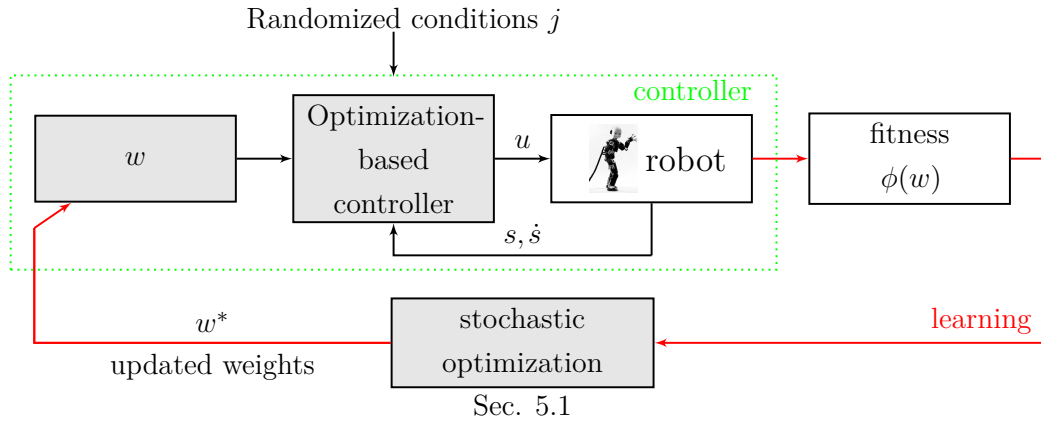


Figure 5.3 Overview of the proposed method. Given task priorities w , the QP-based controller computes a control input u under a set of randomized conditions j (e.g. desired trajectories, disturbances, noise). An outer learning loop allows the optimization of the task priorities through the weights w associated to each task.

The following paragraphs add a few words about the controller, fitness and randomized conditions used with the proposed framework.

Control problem formulation

For the proposed approach, the whole-body control problem is defined from a stack-of-tasks formed by a given number of tasks. Given soft tasks as prioritization scheme, each task is attributed a priority within a set of task weights w . The control problem is subjected to equality and inequality constraints, and can be formulated as an optimization problem of the following shape, where the cost function represents the weighted sum of squared task errors $\tilde{X}_i(u)$, given task weights w_i for each task i .

$$u^* = \arg \min_u \sum_{i=1}^{n_{tasks}} w_i |\tilde{X}_i(u)|^2 \quad (5.14a)$$

$$\text{subject to } D_{eq}u + d_{eq} = 0 \quad (5.14b)$$

$$Du + d \leq 0 \quad (5.14c)$$

The problem (5.14) can be reformulated as QP:

$$u^* = \arg \min_u \frac{1}{2} u^\top H u + u^\top g \quad (5.15a)$$

$$\text{subject to } \underline{b} \leq A u \leq \bar{b} \quad (5.15b)$$

In this formulation, the Hessian matrix H is symmetric and positive definite and g is the gradient vector. A is the constraint matrix, with \underline{b} and \bar{b} the associated lower and upper constraint vectors.

Fitness

As introduced in section 5.1, the fitness is the value computed from the objective function $\phi(w)$, as a way to evaluate the results achieved with the robot after a rollout, given a set of task priorities w . The learning algorithm then seeks to maximize the fitness.

As a result, the formulation of the objective function has an important impact on the optimization process, and should be defined with care. The objective function may be chosen to reflect properties of the desired results on the robot. For example, it may attempt to favor performance of the controller on Cartesian or postural tasks by computing the negative sum of squared errors on desired tasks. A higher balance of the robot could for example be favored by computing the excursion of the zero moment point (ZMP) with respect to the center of the support polygon. The objective function can eventually be formulated from a combination of different desired properties.

Randomized conditions

To achieve robustness through domain randomization and enable the controller to eventually cope with real-world data, the idea is to subject the robot to randomized conditions during each training rollout. These conditions can include for example randomized desired trajectories, disturbances, or noise on sensor signal. A given set of these conditions constitute the set of conditions j (as shown in figure 5.3) under which the controller has to perform.

5.3 Implementation for whole-body torque-control

The proposed method can be implemented for learning task priorities of a whole-body torque-controller. For this purpose, the controller must first be defined, as well as the objective function used to compute the fitness, and the randomized conditions under which the robot shall perform experiments.

5.3.1 Control problem formulation

The controller used for this purpose is the same as the soft tasks controller #2 described in section 2.3.3. It defines an optimization problem, minimizing a weighted sum of squared errors on Cartesian task accelerations (CoM position, neck frame orientation, swing and stance feet pose), the squared error on postural task acceleration, and squared joint torques. The contact of the feet with the ground is stabilized with inequality constraints on the optimization problem. A control input, consisting of joint torques and contact forces, is obtained by solving this optimization problem through QP optimization.

Each task of the controller is attributed a priority within the following set of task weights.

$$\mathbf{w} = \{w_{CoM}, w_{stance}, w_{swing}, w_{neck}, w_s, w_\tau\} \quad (5.16)$$

where the terms $w_{CoM}, w_{stance}, w_{swing}, w_{neck} \in \mathbb{R}$ refer to weights associated to the CoM, stance foot, swing foot and neck Cartesian tasks, and $w_s, w_\tau \in \mathbb{R}$ to the weights associated to the postural task and joint torque regularization, respectively.

The controller is applied to the particular action of performing steps, i.e. performing the following sequential movements for each step: move the CoM above the stance foot, move the swing foot up by 0.025 m, move the swing foot back to its initial pose, and move the CoM back to its initial position.

The passage from one movement to the next takes place when Cartesian task errors are smaller than a threshold and contact forces are consistent with the desired motion, or a maximum state duration has been reached. Over the course of a step, the desired stance foot pose, neck orientation and posture remain at their initial value.

The finite state machine defined in order to achieve this desired movement is the same as described in section 2.3.3: it is divided into 5 states, and takes as input some

user-defined positions for the Cartesian tasks in order to output desired setpoints for Cartesian tasks, in function of the state of the robot. The formulation of the state machine, desired trajectories defined for each task and state, as well as transitions between states are described in detail in appendix B.

5.3.2 Constraints on stochastic optimization

Inequality constraints to the learning problem are defined on joint position limits and torque limits: they act as an extra safety built on top of the QP controller. Lower and upper bounds on joint positions are obtained from the URDF file of each iCub model, while lower and upper torque limits of -60N and +60N are applied to all joints.

5.3.3 Fitness

In order to optimize task weights, the objective function used to compute the fitness must be defined. In our implementation, we compare three different objective functions. The first one, ϕ_p , favors performance on the Cartesian tasks and less deployed effort. The second one, ϕ_r , focuses on robustness, by favoring solutions with smaller excursion of the ZMP position P_{ZMP} with respect to the center of the support polygon O_{SP} . The third objective function, ϕ_{pr} , is a combination of the first two.

$$\phi_p = -\frac{1}{X_{Tmax}} \sum_{t=0}^{t_{end}} \sum_T |\tilde{X}_T|^2 - \frac{0.0001}{\tau_{max}} \sum_{t=0}^{t_{end}} |\tau|^2 \quad (5.17a)$$

$$\phi_r = -\frac{1}{P_{ZMPmax}} \sum_{t=0}^{t_{end}} |P_{ZMP} - O_{SP}|^2 \quad (5.17b)$$

$$\phi_{pr} = \frac{1}{2}(\phi_p + \phi_r) \quad (5.17c)$$

X_{Tmax} , τ_{max} and P_{ZMPmax} are normalization factors with the following values:

$$X_{Tmax} = 4n_s \quad (5.18a)$$

$$\tau_{max} = 2000n_s \quad (5.18b)$$

$$P_{ZMPmax} = 0.005n_s \quad (5.18c)$$

In the above, n_s refers to the number of time samples between $t = 0$ and t_{end} , the duration of an experiment.

We define the duration of an experiment as a fixed amount of time, subject to **early termination** in cases where the robot has fallen or the QP controller could not be successfully solved (which may happen when the weights being tested are far from being optimal, and lead to a configuration in which the QP solver simply can not find a solution). In these cases, a penalty of -1.5 is added to the fitness in equation (5.17).

5.3.4 Randomized conditions

To achieve robustness through domain randomization and enable the controller to eventually cope with real-world data, the robot is subjected to randomized conditions as presented in table 5.1. A given set of these conditions constitutes the set of conditions j (as shown in figure 5.3) under which the controller has to perform.

In particular, tests performed in simulation have shown that applying a force of 10 N on the chest for 1 second is sufficient to destabilize the robot when using unoptimal weights. Thus, using such disturbances during learning can encourage the generation of robust task priorities. As part of the randomized conditions, a random number of wrenches of varying amplitude and direction are applied to the chest of the robot at random times during simulation.

Moreover, with torque-control, it can happen that the gains used for the computation of feedback terms on desired trajectories may not be rigorously tuned, estimation errors of the floating base pose may occur, and force-torque sensors may yield noisy measurements. Therefore, under these circumstances, the robot may not perform the desired motion with precision. However, it is desirable that the controller maintains balance of the robot. For this reason, it has been deemed useful to randomize trajectories of the CoM and feet, as a way to generate a motion of the robot which is different from optimal trajectories. Furthermore, Gaussian noise on force-torque sensor measurements and joint velocity signals are integrated into the randomized conditions.

Table 5.1 Randomized set of conditions j

Randomized Condition (RC) in j	Random value
1. Gaussian noise on F/T sensor signals	On / Off
2. Appointed swing foot	Left / Right
3. Direction in which swing foot is moved	Front / Back
4. X_{CoM_d} moved forward by δ (m)	$\{\delta \mid \delta \in \mathbb{R}^+, \delta \leq 0.02\}$
5. n_F external wrenches applied on chest	$\{n_F \mid n_F \in \mathbb{Z}, n_F \leq 7\}$
<i>and for each wrench i:</i>	
at time t_{F_i} (s, with 1e-2 precision)	$\{t_{F_i} \mid t_{F_i} \in \mathbb{R}^+, t_{F_i} \leq 10\}$
duration d_{F_i} (s, with 1e-2 precision)	$\{d_{F_i} \mid d_{F_i} \in \mathbb{R}^+, d_{F_i} \leq 1\}$
direction $(\gamma_{F_i}, \theta_{F_i}, \varphi_{F_i})$ (rad)	$\{\gamma_{F_i} \mid \gamma_{F_i} \in \mathbb{R}, \gamma_{F_i} \leq 2\pi\}$ $\{\theta_{F_i} \mid \theta_{F_i} \in \mathbb{R}, \theta_{F_i} \leq 2\pi\}$ $\{\varphi_{F_i} \mid \varphi_{F_i} \in \mathbb{R}, \varphi_{F_i} \leq 2\pi\}$
force magnitude F_{F_i} (N)	$\{F_{F_i} \mid F_{F_i} \in \mathbb{R}, F_{F_i} \leq 10\}$
torque magnitude τ_{F_i} (Nm)	$\{\tau_{F_i} \mid \tau_{F_i} \in \mathbb{R}, \tau_{F_i} \leq 10\}$
6. Gaussian noise on joint velocity signals	On
7. Gaussian noise on F/T sensor signals	On

5.4 Application to learning task priorities for walking in place

In order to assess the effectiveness of the proposed method, it is applied to the problem of learning task priorities of the whole-body torque-controller presented in section 5.3, which allows the robot to perform a stepping motion. This application in particular can be difficult to achieve when the robot model is inaccurate. Additionally, from the point of view of learning task priorities, it may be considered challenging, due to the changing contacts and conditions arising from the stepping motion.

Therefore, using the (1+1)-CMA-ES algorithm with CCA described in section 5.1, task priorities are optimized in simulation, for the goals of making the iCub perform steps, and showing that the method presented in this chapter allows to overcome issues related to the transferability problem.

Experiments are then performed with iCub simulations (using 23 DOFs on legs, arms and torso), in order to validate empirically that the method described above is capable of generating task priorities which (i) produce robust whole-body motions, even when contacts due to physical interaction with the environment evolve in time, and (ii) can cope with imperfections in the robot model, disturbances and noise.

In order to verify that the method also allows to overcome the transferability problem, experiments are conducted on two different robot models, which shall be termed **tethered** and **backpacked**. These are two distinct models of the iCub with different inertial properties: the first one functions with a tethered power supply, and the second one with a battery pack on the back, as shown in figure 5.1.

Experiments are conducted in three parts:

1. Training is performed with the tethered iCub model, in order to obtain optimized task weights. During training, the robot is subjected to a set of randomized conditions as defined in subsection 5.3.4. The stochastic optimization is also subjected to constraints as defined in subsection 5.3.2.
2. Testing on the tethered iCub model allows to assess the success achieved with the optimized task weights.
3. Testing is performed with the backpacked iCub model, under a different set of randomized conditions, allowing to confirm if the optimized task priorities also help the transfer of results between different platforms.

Over these experiments, three different objective functions are tested, as defined in subsection 5.3.3.

Note that over all experiments, the proportional-derivative gains used to compute reference accelerations associated to Cartesian and postural tasks are kept constant.

The following subsections describe the experiments in more detail, along with the achieved results.

5.4.1 Training with the tethered iCub model

As mentioned above, training is performed with the tethered iCub model. The task used for the optimization is to perform 1 step, as described in section 5.3.1. The simulation is limited to 10 seconds, allowing the robot to perform one step and shift

its weight to the side in order to start a second one, making sure that the robot has remained stable after foot touchdown.

Since task priorities are relative to each other, w_{CoM} is attributed a fixed value of 1. The remaining task priorities are attributed bounds as shown in the bottom of table 5.2. Furthermore, the weight values

$$w_0 = \{1, \quad 1, \quad 1, \quad 0.1, \quad 1e-3, \quad 1e-4\}$$

obtained by hand have been verified to allow the tethered iCub model to successfully perform the desired stepping motion, and are therefore used as a starting point.

Four learning experiments have been performed:

- one for each of the three fitness functions from equation (5.17) with DR, where, to encourage the generation of robust task priorities through DR, the robot is subjected to the randomized conditions 1 to 5 (see table 5.1), for each learning iteration performed when optimizing task weights.
- one for ϕ_{pr} , without the use of DR. In this case, randomized conditions are disabled. This additional experiment allows to assess the contribution of DR.

Optimized task priorities are obtained by performing 200 iterations of (1+1)-CMA-ES, as introduced in section 5.1, applied to the control framework, with an initial exploration rate of 0.1. Furthermore, each learning experiment has been repeated 10 times, allowing to assess the repeatability of the procedure.

Results have shown that 200 iterations are sufficient to achieve strong convergence. Weights obtained with each of the fitness functions in equation (5.17) are shown in table 5.2.

5.4.2 Testing with the tethered iCub model

In order to validate the robustness achieved with the optimized weights, each set of them has been tested on the same iCub model as the one used for training, but not subjected to randomized conditions. The testing scenario in this case is to achieve 6 consecutive steps.

Typical ZMP, CoM and feet trajectories achieved using each fitness function are shown in figures 5.4 and 5.5. Typical estimated base velocities, ground contact wrenches

and joint torques are also shown in figures 5.6, 5.7 and 5.8. These values are estimated using the force-torque sensors present in the legs of the robot, following a procedure based on a recursive Newton-Euler algorithm, as introduced in section 2.4.1.

Also, considering a successful experiment as being one where the robot succeeds in performing 6 consecutive steps, the success rates achieved with the optimized weights from each fitness function are shown in table 5.2: weights optimized with ϕ_p , ϕ_r and ϕ_{pr} and DR achieved success rates of 50%, 70% and 100%, respectively. On the other hand, weights optimized with ϕ_{pr} and no DR showed a lower success rate of 80%.

5.4.3 Testing with the backpacked iCub model

In order to create conditions similar to performing experiments on the real robot, we have tested the optimized weights on a different platform, using the backpacked iCub model. It is subjected to randomized conditions 5 to 7 (see table 5.1). For the rest, this robot uses the same parameters as the tethered iCub in the experiments of section 5.4.2, and the testing scenario is also to achieve 6 consecutive steps.

Successful CoM and feet trajectories obtained with initial weights w_0 , and weights optimized with ϕ_r , ϕ_{pr} using DR, and ϕ_{pr} not using DR, are shown in figure 5.5. As could be expected, due to the addition of noise, ZMP trajectories turned out to be highly noisy and are therefore not shown. The success rates achieved with the optimized weights from each fitness function are shown in table 5.2.

These results show that initial weights w_0 do not allow the robot to perform more than a couple of steps, and weights optimized with ϕ_p , ϕ_r and ϕ_{pr} and DR achieve success rates of 0%, 50% and 100%, respectively. On the other hand, weights optimized with ϕ_{pr} without DR yield a success rate of 20%.

Therefore, weights obtained with ϕ_{pr} allow for a higher robustness of the controller, when tested on a different platform than the one used for learning. Additionally, the rate of success achieved with DR shows to be significantly higher than without DR, demonstrating that DR did have a measurable impact on the achieved robustness.

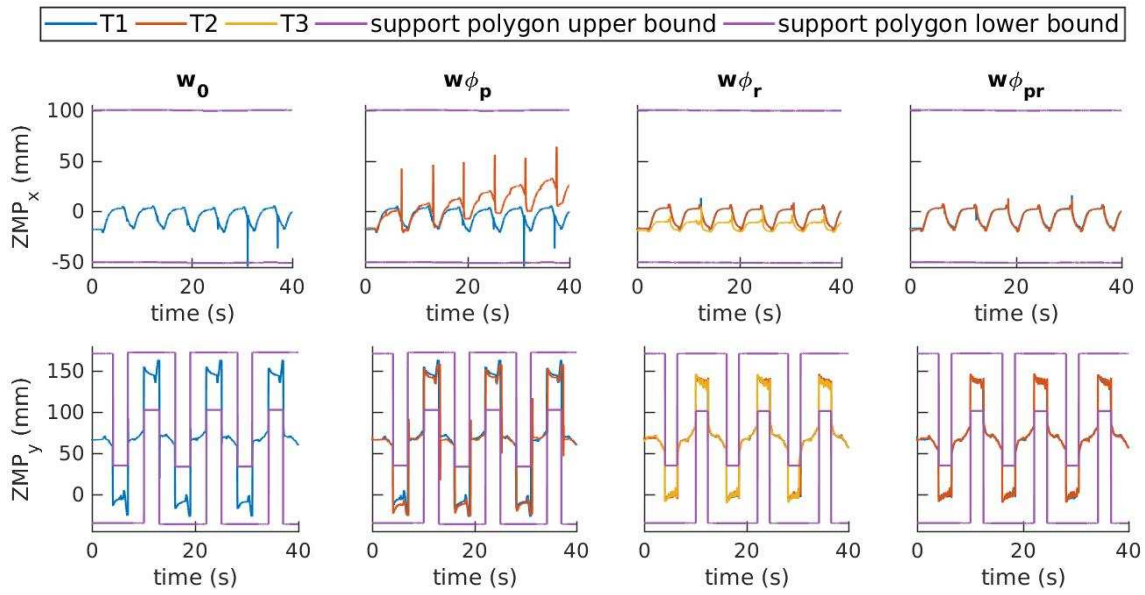


Figure 5.4 Typical ZMP trajectories obtained for 6 steps performed with the tethered iCub model. From left to right: given initial weights w_0 , and weights optimized with ϕ_p , ϕ_r , ϕ_{pr} . Each color (blue, red or yellow) denotes the use of a different set of optimized weights, while purple lines mark the bounds of the support polygon. The x , y and z axes correspond respectively to the sagittal, frontal and vertical axes.

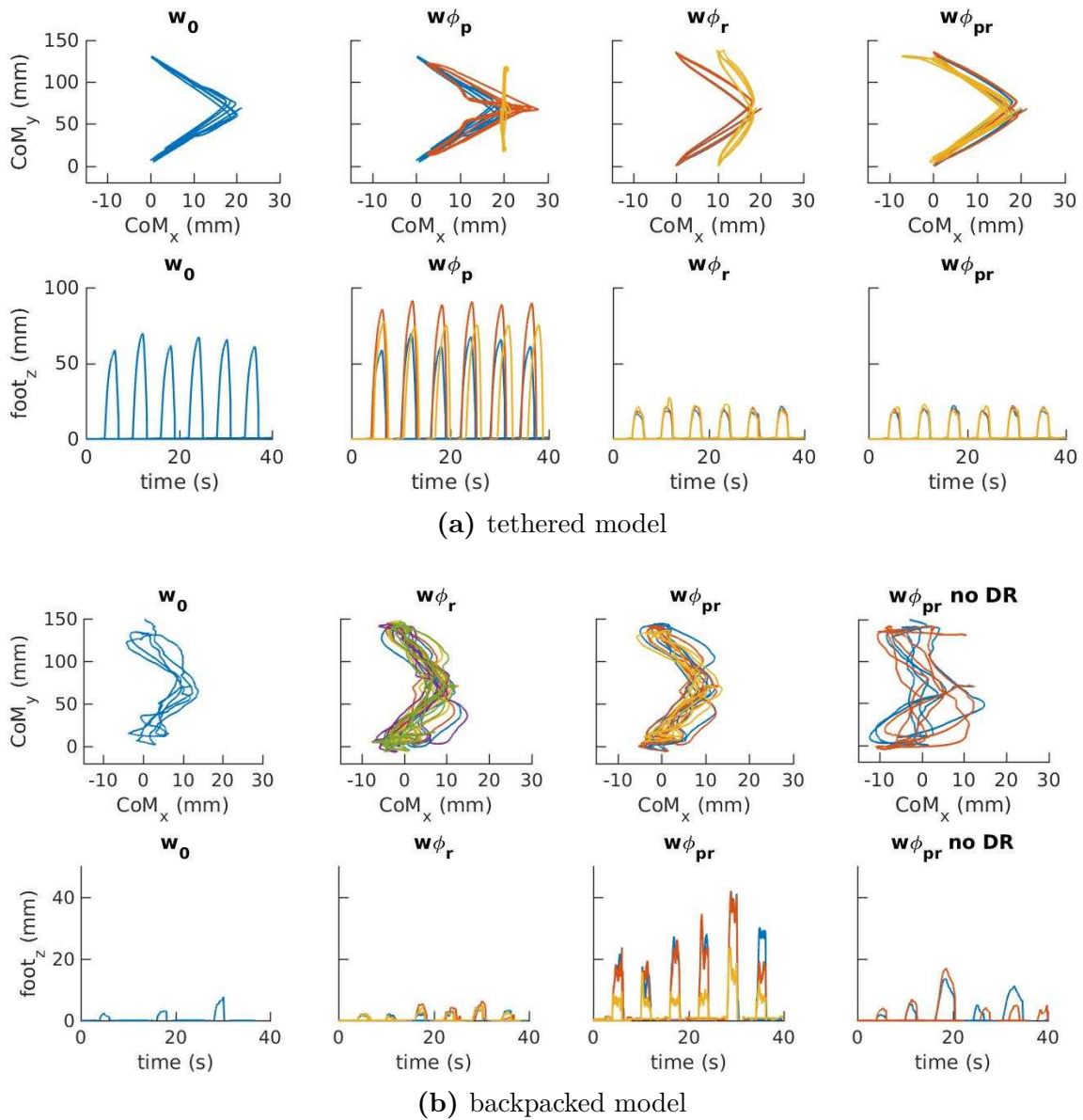


Figure 5.5 Typical CoM and feet trajectories obtained for 6 strides (a) performed with the tethered iCub model, given initial weights w_0 , and weights optimized using DR with ϕ_p , ϕ_r , ϕ_{pr} , and (b) performed with the backpacked iCub model, given initial weights w_0 or weights optimized using DR with ϕ_r , ϕ_{pr} , and weights optimized with ϕ_{pr} but without the use of DR (results obtained with ϕ_p are not shown in this case, since they were not successful in achieving 6 strides). Each color denotes the use of a different set of optimized weights. The x , y and z axes correspond respectively to the sagittal, frontal and vertical axes.

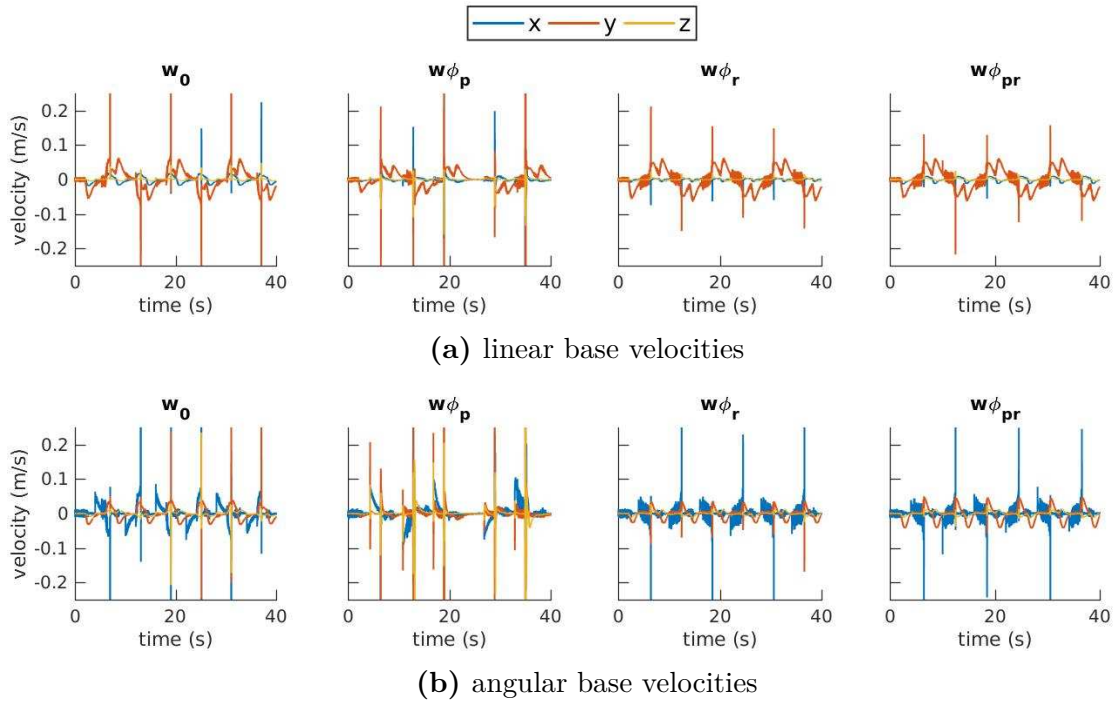


Figure 5.6 Typical base velocities obtained for 6 strides performed with the tethered iCub model, given initial weights w_0 , and weights optimized using DR with ϕ_p , ϕ_r , ϕ_{pr} . The peaks shown in these graphs are the highest for linear velocities obtained with initial weights, for which they reach up to 1.7m/s. The x , y and z axes correspond respectively to the sagittal, frontal and vertical axes.

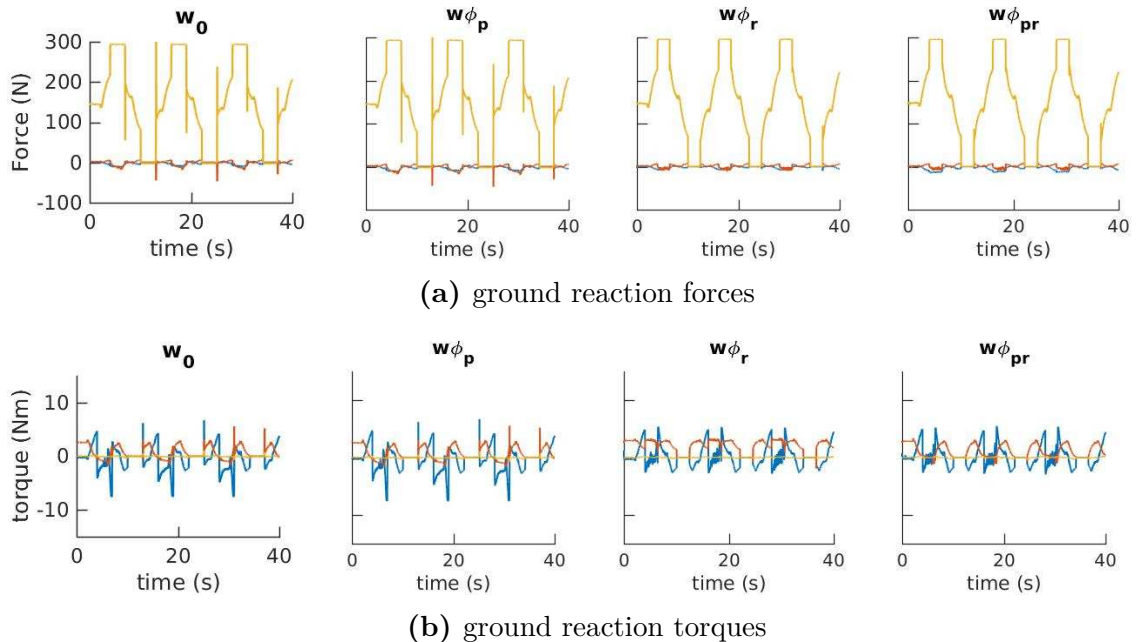


Figure 5.7 Typical ground reaction wrenches estimated on the right foot, obtained for 6 strides performed with the tethered iCub model, given initial weights w_0 , and weights optimized using DR with ϕ_p , ϕ_r , ϕ_{pr} . Note that the left foot shows the same pattern.

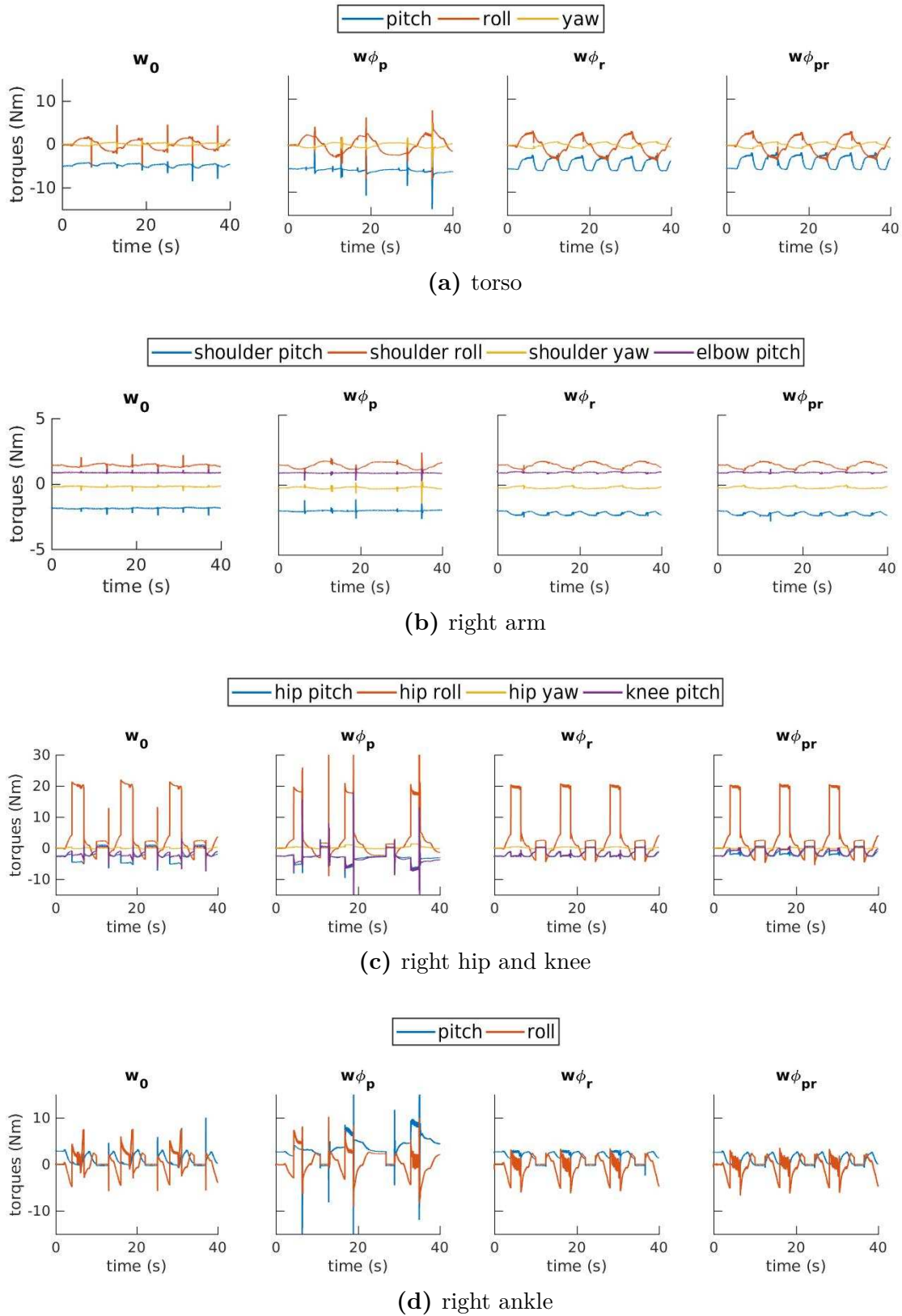


Figure 5.8 Typical estimated joint torques at the torso, right arm and right leg, obtained for 6 strides performed with the tethered iCub model, given initial weights w_0 , and weights optimized using DR with ϕ_p , ϕ_r , ϕ_{pr} . Note that the left arm and leg show the same pattern as the right ones.

Table 5.2 Summary of performed experiments and achieved results

Scenario	Training									Testing 1	Testing 2
Robot	tethered									tethered	backpacked
Task	1 step									6 steps	6 steps
RC	1, 2, 3, 4, 5									-	5, 6, 7
	Fitness	#	DR	w_{CoM}	w_{stance}	w_{swing}	w_{neck}	w_s	w_τ	success	success
	hand tuning	1	No	1	1	1	1	1e-3	1e-4	1/1	0/1
	ϕ_p	10	Yes	1	0.2 ± 0.3	1.1 ± 1.2	$(1 \pm 3)e-3$	0.5 ± 0.3	$(2 \pm 5)e-5$	5/10	0/10
	ϕ_r	10	Yes	1	1.6 ± 1.2	1.8 ± 1.0	0.1 ± 0.1	$(4 \pm 7)e-3$	1e-10	7/10	5/10
	ϕ_{pr}	10	Yes	1	0.9 ± 1.3	2.4 ± 1.1	0.6 ± 1.2	1e-6	1e-10	10/10	10/10
	ϕ_{pr}	10	No	1	1.0 ± 0.3	0.1 ± 0.3	0.2 ± 0.1	1e-6	$(4 \pm 5)e-6$	8/10	2/10
	weights lower bounds:			1	1e-4	1e-4	1e-10	1e-6	1e-10		
	weights upper bounds:			1	10	10	10	10	0.1		

This table presents the experiments that have been performed, in summary form. Training and testing have been performed with different robot models and stepping tasks, as well as under different randomized conditions (RC) as defined in table 5.1. The column “#” gives the number of training experiments performed given each fitness function, while only 1 set of manually tuned gains is used. In the column “DR”, “Yes” means that the randomized conditions defined at the top of the table have been used for domain randomization while training; “No” means that they have been disabled. The “success” columns report success rates achieved when testing the sets of trained weights.

5.4.4 Discussion

In summary, the proposed method generates task priorities for successful whole-body control of different robot models, in 200 learning iterations. It has been demonstrated by performing training on a first model of the iCub robot, then testing on a different model equipped with a battery pack on its back and subjected to diverse working conditions.

Results achieved with ϕ_p , a fitness function favoring task performance, are likely limited by overfitting on the model used for learning and did not allow much robustness with respect to disturbances. On the other hand, optimizing for robustness with ϕ_r allowed higher chances of success when changing conditions, or robot model, by encouraging smaller ground reaction forces and the generation of angular momentum through the torso and arms, as shown in figures 5.7 and 5.8. However, ϕ_r might have neglected the fulfillment of tasks, which are used for keeping balance. Instead, using ϕ_{pr} , a fitness function combining robustness and performance, allowed to obtain sensible optimized task priorities, achieving superior results compared to the two previous fitness functions, and significantly better than the previously hand tuned solution w_0 .

With ϕ_{pr} , the swing foot placement, crucial for stability at touchdown, is given high importance, while the neck orientation task a lesser one. This allows for compliance to external forces, which thus facilitates recovery from external perturbations and contact switching. As for the postural task, its allotted low priority allows it to be used as regularization (just as joint torques), instead of competing with Cartesian tasks. Such a solution is interesting, as it may not have been *a priori* self-evident to an expert defining task priorities. However, this seems sensible: the postural task can be considered to compete with the CoM and feet tasks, while the orientation task, if too strong, may not facilitate the recovery from external perturbations and contact switching.

Furthermore, the ranges over which sets of optimized weights are obtained show that the problem has multiple local minima. Therefore, although task priorities require proper tuning, the controller is not highly sensitive to a single precise adjustment of task weights. Nevertheless, considering that the search space can span values across $1e-10$ to 10 , it could actually be appropriate to adapt the search algorithm, such that it first searches on a log scale to identify an appropriate range of values, before performing a finer search within that range.

Eventually, in order to further improve performance when using a different robot model or moving on to experiments on the real robot, one may be interested in performing another round of optimization on the task priorities, for example exploiting data-efficient learning methods, as done in [Spitz et al., 2017]. Furthermore, the controller used in this work, as presented in section 5.3.1, can be considered as low-level. As such, the addition of a higher level controller or motion planner, such as done in [Dafarra et al., 2018] with model predictive control, would allow higher stability in order to achieve a complete walking motion.

Nonetheless, the robustness achieved with the proposed method is promising and has the potential to allow higher success when passing from simulation to real-world experiments. The exploration of different DR conditions may eventually be considered, such as randomizing values associated to the dynamics of the system, control delay, or measurement noise amplitude and delay, which could potentially be more consistent with the differences constituting the reality gap.

Regarding the specific methods used here, note that previous related work [Modugno et al., 2016b] has provided a framework that could directly be adapted to verify that learning task priorities, with the alliance of domain randomization and appropriate fitness functions, can help with transferring results. Different learning approaches could then be used with the same idea, for achieving related results. For example, policy learning could also be used in the same context, to find the parameters that allow to achieve certain behaviors of the robot, under randomized conditions or parameters of the cost function. Considering that policy learning has been shown to learn new, similar behaviors in few experiments starting from previously learned behaviors [Bischoff et al., 2014], it could be used for a second round of optimization when attempting to transfer results on the real robot, or in order to expand the walking in place motion to walking forward.

5.5 Conclusion

In this chapter, a novel approach for automatically tuning task priorities of a controller has been proposed, based on stochastic optimization and domain randomization. Results achieved with a carefully designed fitness function have allowed to transfer results between robot models subjected to different working conditions, without the need to re-tune task priorities.

This achievement, not having to re-tune task priorities between platforms and for changing conditions, amounts to beneficial time and effort saving during the deployment of a controller.

Regarding task priorities, they can be crucial in the case where external perturbations happen, in prioritizing balance over following precise trajectories or minimizing the effort spent by the robot. However, isolated tasks of a whole-body controller may, in some contexts, be seen as conflicting with each other.

For example, in the whole-body controller used in this chapter for validating our approach, tasks are defined such that the robot moves, but also such that its effort is minimized. These two actions conflict with each other in some sense, and a tradeoff is achieved through the use of task priorities. Therefore, this provides a partial explanation as to why task priorities have an important impact on the achieved behavior of the robot.

Our controller also uses a postural task on all joints, whereas the position of the center of mass and feet, combined with the minimization of joint torques, may be sufficient to define the movement of the lower-body. From this perspective, in addition to tuning parameters, maybe it is worth to *reformulate* the whole-body control problem to avoid this kind of conflict. We may also ponder whether the formulation of whole-body controllers, such as the one used in this chapter, is the best one. Could it be possible that tuning be so tedious due to the way tasks and parameters are posed, and could a different formulation help in this sense?

A new version of the controller, following the information gained here, shall eventually be implemented. For example, since the postural task is prioritized as regularization, one can conclude that it did not have a significant effect on the generation of motion. However, using the postural task as regularization does not allow a precise control of the arms, which may be beneficial in case of disturbances directed to the arms. A possible solution would be to partition the postural task between upper-body and lower-body.

Nonetheless, one may argue that the black-box constrained stochastic optimization does not volunteer deep insight. At this point, we can offer plausible speculations as to why the algorithm obtained the weight values it obtained, but a deeper analysis of how task priorities affect the robustness of the system may help gain further scientific insight.

Coming back to the issue of tuning whole-body torque-controller parameters, proportional-derivative gains used for the computation of feedback terms also require notable tuning. Also, just as for task priorities, their proper adjustment is crucial for the realization of successful experiments on the real robot. Since the method presented above does not address their tuning, manual adjustment of controller gains still needs to be performed for experiments on the real robot. This fact limits further experiments aiming to assess the benefits of the presented method on bridging the reality gap. As a solution, the extension of the proposed method to additional parameters such as feedback gains should be examined.

On a different note, while the optimized task priorities have shown to allow the transfer of results between platforms for the same whole-body motion, it would also be highly interesting to verify that, given well adjusted parameters, a single controller allows to produce a range of different whole-body motions. This question shall be the subject of the next chapter, where we define a teleoperation framework that allows to produce generic whole-body motions through human imitation.

Chapter 6

Teleoperation of generic whole-body motions

In general, an elaborate control architecture, such as the one of figure 6.1, is used to generate reference trajectories for the whole-body control of a humanoid robot. Two architectural layers are employed (trajectory optimization and simplified model control), in order to generate feasible trajectories for the robot, from a given desired motion. Notably, the process may require offline computations in view of achieving real-time whole-body control. For instance, during trajectory optimization, desired feet and ZMP trajectories may be obtained with an offline planner.

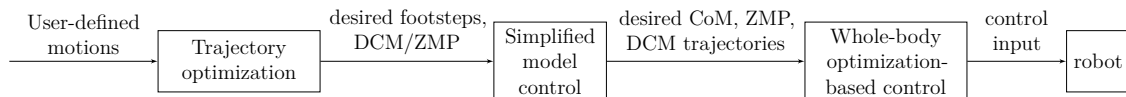


Figure 6.1 Typical whole-body control architecture for humanoid robots, adapted from [Romualdi et al., 2018]

This approach is extremely useful for achieving impressive results, but it may lead controllers to be specialized for specific movements of the robot. Our main interest in this chapter is to investigate ways that a controller can be developed for robustness to more generic reference trajectories.

In order to achieve a higher flexibility in the generation of whole-body movements, we propose to eliminate the need for trajectory optimization and simplified model control, by instead obtaining generic and feasible trajectories for a humanoid robot through human imitation by teleoperation, as shown in figure 6.2.

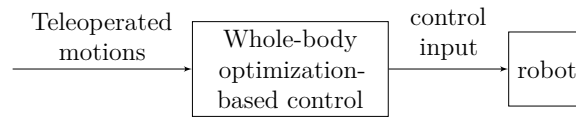


Figure 6.2 The control architecture used for whole-body control in this chapter. Trajectories for the robot are obtained from teleoperation of a human operator. They are then directly used by the whole-body optimization-based controller to compute the control input (contact forces and joint torques) for the robot to achieve these trajectories.

On the subject of teleoperation, the classical motion retargeting approaches are generally not applicable to the robot body parts that are the most involved in a dynamic movement, e.g. legs and feet, particularly when footsteps and contact transitions are involved. For this reason, challenging lower-body motions such as walking are generally dissociated from the teleoperation of upper-body manipulation tasks. Associating these two is therefore a highly interesting motivation for developing a controller that is robust to a variety of whole-body motions.

In this chapter, we address the retargeting in real-time of generic whole-body human motions, acquired by a wearable motion capture suit, onto a humanoid robot. For this purpose, we propose a generic teleoperation framework that can handle a variety of whole-body motions demonstrated by a human operator, and then generate appropriate motions for the humanoid robot.

The whole-body optimization-based controller for the robot is then based on the quadratic programming formalism, allowing to take into consideration a variety of retargeting tasks, without violating robot constraints. We have developed a stack-of-tasks for this purpose, building on observations gained from chapter 5. This stack-of-tasks is then applied to a whole-body velocity-controller designed to keep balance, given input generic reference trajectories.

The reason why a velocity-controller is used here instead of a torque-controller, as in the previous works, is mainly due to the adoption of OpenSoT [Hoffman et al., 2018, 2017; Rocchi et al., 2015] as library to develop the controller. OpenSoT in this case offers easy compatibility with the teleoperation module first developed in [Penco et al., 2018], which shows to be highly efficient for real-time whole-body teleoperation of a humanoid robot. As additional perks, OpenSoT offers faster compilation and development of a controller when compared with a controller developed in Matlab/Simulink, as well as real-time safety. At the moment the presented work has been developed, only the velocity-control library of OpenSoT was apt for whole-body movements with time-

evolving contact conditions, but it has the advantage of not requiring extensive tuning of PD gains, making it easier to transfer results from simulation to real-world.

Our approach differs from existing work by the flexibility it allows. It takes into consideration various complex whole-body movements involving simultaneous lower-body and upper-body movements, such as (but not limited to) squatting or walking while waving the arms. More than only involving EE position retargeting, our method considers also specific tasks for the head, torso, waist and each arm of the robot, providing a complete and human-like teleoperation performance, as illustrated for example in figure 6.3. Nonetheless, the methods used in [Ishiguro et al., 2017, 2018; Penco et al., 2018] are not incompatible with the framework proposed in this paper. For instance, a stabilizer or dynamic filter can easily and advantageously be integrated into our framework.

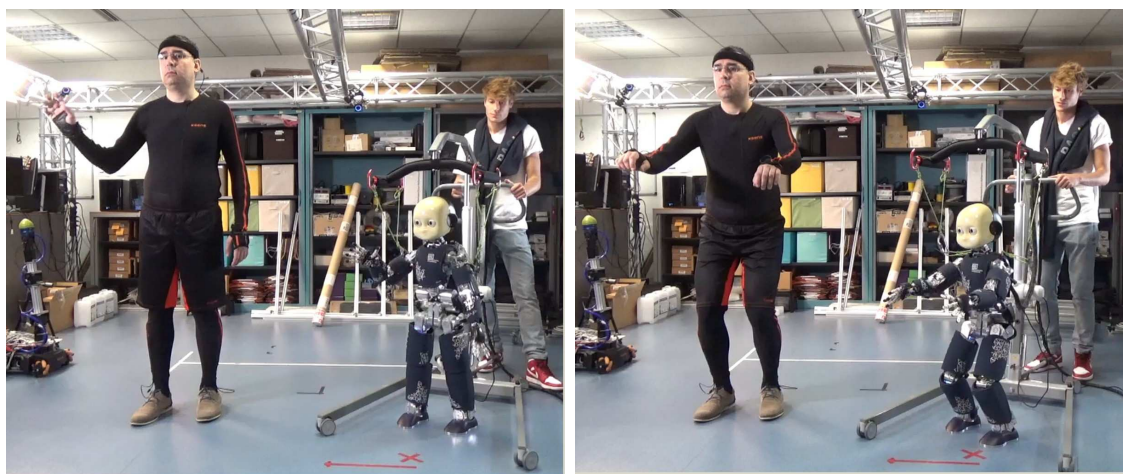


Figure 6.3 Retargeting of human whole-body movements on the robot: moving the arms while walking (on the left) and squatting (on the right).

Section 6.1 introduces the framework developed for this work, including the retargeting method for teleoperation and the optimization-based whole-body controller used for this approach. It has been implemented for the iCub and applied to the problem of retargeting upper-body movements onto the robot, while walking. The capacity of the framework in handling generic motions is evaluated in section 6.2, where several experiments are performed with the iCub, involving whole-body movements such as squatting and walking, simultaneously with upper-body movements.

6.1 Whole-body velocity-control framework for teleoperation of humanoid robots

In this section, a framework for whole-body velocity-control is defined, in order to achieve whole-body actions, such as walking while allowing for generic upper-body movements. The structure is similar to the controllers used in the previous chapters, but includes a retargeting module that produces reference trajectories for the upper-body (torso, arms and head).

An overview of the proposed framework is illustrated in figure 6.4, where trajectories of a human are retargeted, in real-time, into corresponding movements for the robot. A finite state machine is used in parallel, for defining Cartesian trajectories, such as CoM and feet trajectories when walking. The outputs of the retargeting module and finite state machine are used by an optimization-based whole-body controller in order to compute a control input which allows the robot to achieve the desired motion. The controller is formulated as a QP problem, and uses the stack-of-tasks approach.

The following subsections present the control framework and its components in detail. The retargeting module and finite state machine are first defined in subsections 6.1.1 and 6.1.2. Subsection 6.1.3 then defines the stack-of-tasks to be used by the QP controller, described in subsection 6.1.4 as an optimization problem.

As a matter of fact, an important advantage of the proposed framework lies in the flexibility it allows in its detailed implementation, which means that it can easily be adapted to different robots and applications. Because the aim is to expose the framework, some components may use basic or state of the art methods that do not require being verified for validity themselves, but the adaptability of the framework makes them easy to upgrade.

6.1.1 Retargeting module

The first step for a motion retargeting technique is to capture the evolution of the human operator's movements. For this purpose, state of the art motion capture cameras and wearable sensors allow high-fidelity, high-frequency tracking of human motion. Once human posture data is acquired from the motion capture system, it can be mapped to feasible corresponding values for the robot.

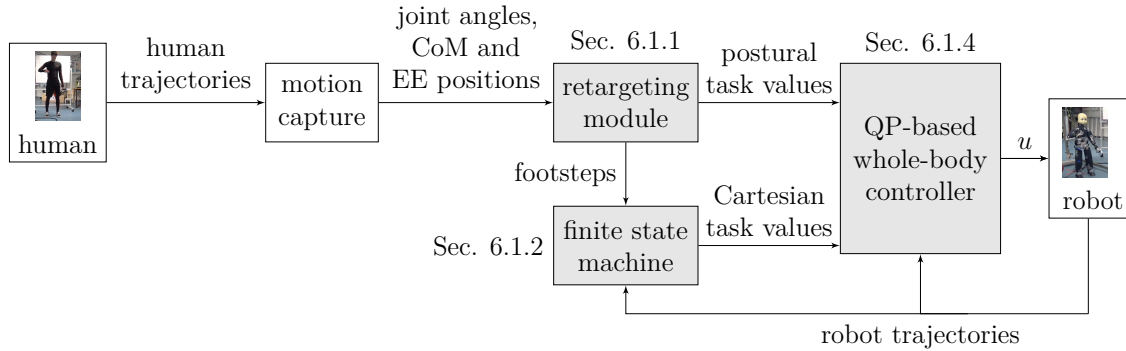


Figure 6.4 Overview of the proposed method. Human motion is measured and retargeted in real-time to the robot, while a finite state machine provides feasible Cartesian task values (e.g. CoM and feet trajectories when walking is detected). A whole-body controller computes the control input achieving desired trajectories on the robot.

Body dimensions, mass distribution and motion constraints may greatly differ between a robot and a human operator. As a result, a robot must generally walk in a different way than a human, which can affect the global position of its head and hands (for example, the robot may produce a more exaggerated pendulum movement). Retargeting Cartesian tasks is therefore not always straightforward, and for this reason we use postural tasks for the upper-body instead.

In the proposed approach, upper-body joint positions are measured and grouped into subcategories: head, torso, left arm and right arm. As for measurements relating to the lower-body, the ground projection of the center of mass, the height of the waist and the position of the feet are measured, but it would be straightforward to include also leg joint position measurements.

The retargeting method used here has a few advantages. For one, it does not assume the initial body orientations of the human operator and robot to match precisely. Also, the only human/robot ratio required is for the waist height.

Mapping and retargeting joint positions

The motion capture human model considered here comes from the motion capture software used in this work (XSens MVN [Xsens, 2017]): it is composed of spherical joints, most of which can easily be assigned to the corresponding robot joints on the arms and legs.

However, unlike on a human body, the robot torso is a rigid link controlled with 3 DOFs, which makes mapping joints related to the torso less intuitive. Therefore, an approximate mapping is performed, considering the motion capture joints most involved in the torso motion to be the ones placed on the vertebrae going from the second lowest lumbar vertebra up to the thoracic vertebra at the level of the breastbone. The rotation of each DOF of the robot torso is approximated by computing the sum of the corresponding rotations on these joints. Sanity tests performed in simulation allowed to confirm the validity of the mapping reported in figure 6.5.

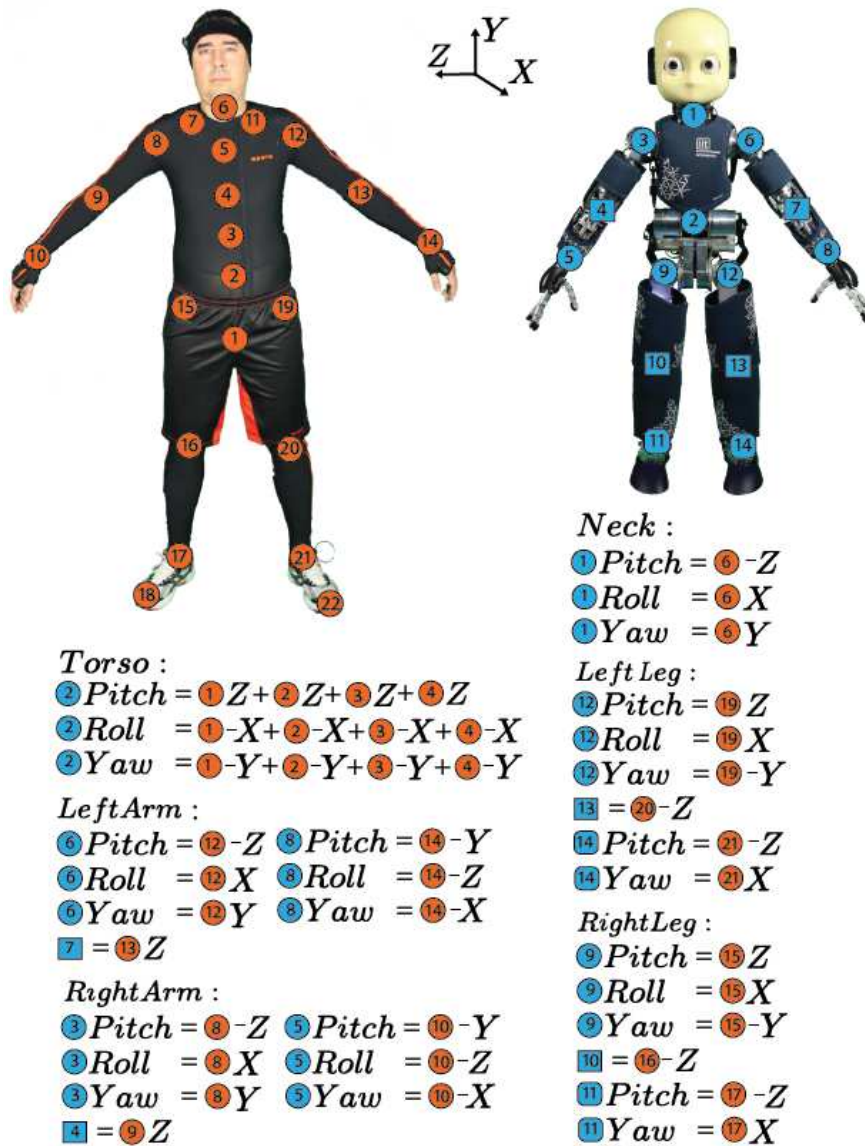


Figure 6.5 Mapping between motion capture and iCub joints. Our motion capture model, here, is the one of the XSens MVN system.

Then, this mapping allows to retarget the variation of joint positions with respect to the starting posture, according to

$$\Delta s_{i_H} = s_{i_H} - s_{0_H} \quad (6.1)$$

$$\Delta s_{i_R} = s_{0_R} + \Delta s_{i_H} \quad (6.2)$$

where s is the vector of current joint positions, Δs is the vector of joint variations with respect to the initial posture, the indices 0 and i refer to measurements at initial time and at time i , and the subindices H and R indicate measurements on human and robot, respectively.

Retargeting lower-body movements

Due to important differences between human and robot inertial properties and physical constraints, stable balancing and walking may not be achieved on the robot simply by retargeting joint movements. Therefore, in order to ensure stable walking of the robot, as well as whole-body movements, we define CoM and feet motions as follows.

Since mass distribution can be significantly different between human and robot, the CoM of the robot may be in a different location on the body as compared to a human¹. For this reason, retargeting vertical CoM movement might cause the motion of the robot to differ significantly from that of the human (e.g. bending the torso over, rather than bending the knees).

As a solution, we choose to control the height of the robot CoM through the variation of waist height Δz_{waist} , obtained for the teleoperator as follows.

$$\Delta z_{waist_{i_H}} = z_{waist_{i_H}} - z_{waist_{0_H}} \quad (6.3)$$

In this case, mapping can be achieved using the ratio of robot/human waist height measured at a resting position (i.e. standing with straight arms along the body).

$$\Delta z_{waist_{i_R}} = \frac{z_{waist_{0,R}}}{z_{waist_{0,H}}} \Delta z_{waist_{i_H}} \quad (6.4)$$

¹For instance, with the iCub, the CoM is situated higher on the chest. The robot CoM may be in a different location in other humanoid robots.

The height of the robot waist at each time instant can then be computed as:

$$z_{waist_{i_R}} = z_{waist_{0_R}} + \Delta z_{waist_{i_R}} \quad (6.5)$$

Then, we propose to use a finite state machine (such as described in subsection 6.1.2), in order to generate CoM ground projection and feet trajectories that are appropriate for different types of contacts, or actions to be performed.

Measurements of the teleoperator's movements can then be used for detecting the type of action to perform. For example in our implementation, measuring a vertical and horizontal displacement of one of the teleoperator's feet over a certain threshold is used to detect walking. From there, the number of footsteps can be counted with every time a foot is brought back to the ground.

Our approach to walking is then to retarget footstep length, and have the robot perform an equal number of steps as the teleoperator. In view of achieving a performance that is closer to the human's, we choose not to use the teleoperation for activating an offline walking pattern generator (as done for example in [Elobaid et al., 2018; Kim et al., 2013]). Due to this, the robot will keep walking until the right number of footsteps has been achieved.

For footstep retargeting, let us assume that the feet move along the x -axis (forward or backward) when walking; however, the extension to a 2D foot position retargeting is straightforward. We first measure the displacement of the human foot over a footstep. Then, we retarget the human footstep onto the robot, considering the range of feasible values for the human and the robot (comprised within minimum and maximum values $x_{foot_{min}}$ and $x_{foot_{max}}$), by computing the offset

$$o_{footstep} = \frac{(x_{foot_H} - x_{foot_{min_H}})}{(x_{foot_{max_H}} - x_{foot_{min_H}})} \quad (6.6)$$

from which we get the corresponding robot footstep length

$$x_{foot_R} = o_{footstep}(x_{foot_{max_R}} - x_{foot_{min_R}}) + x_{foot_{min_R}} \quad (6.7)$$

The same approach can be applied to retarget the step height.

6.1.2 Finite state machine

The finite state machine is used to define Cartesian reference trajectories, depending on motion capture data and desired behaviors. It allows to switch between teleoperated and non teleoperated motion, as well as between different types of motions with contact switching.

To demonstrate the potential of the framework, we set up a basic example of hierarchical state machine, as illustrated in figure 6.6.

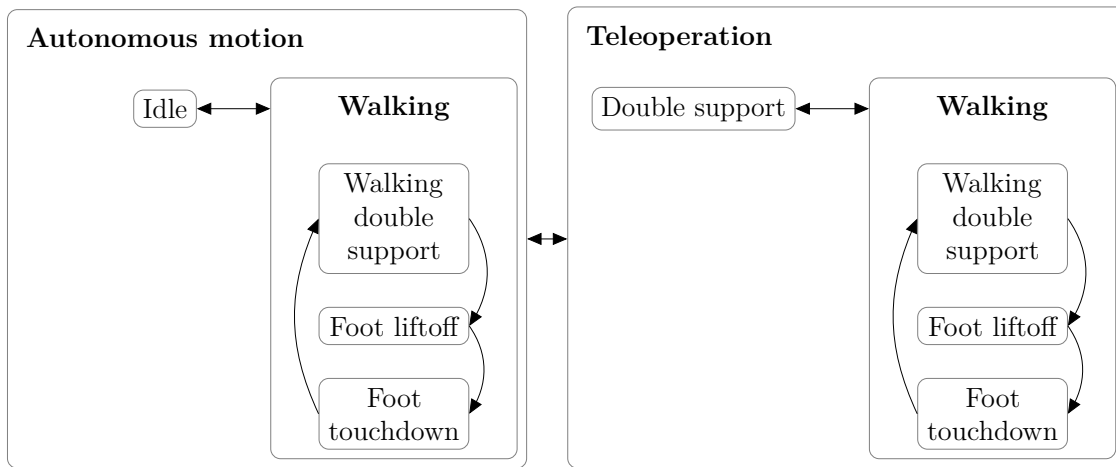


Figure 6.6 States of a hierarchical finite state machine for teleoperation and walking

States of figure 6.6 can be described as follows.

1. **Autonomous motion** – used when teleoperation is inactive.
 - (a) **Idle** is the state in which the controller is initialized. At this point, the initial pose of the robot, as well as the initial pose associated to each task, are stored. The robot remains in the same pose, until teleoperation is started.
 - (b) **Walking** is used for generating a walking motion. It is composed of the same states as state 2b (teleoperated walking), except that reference poses for the postural tasks remain constant.
2. **Teleoperation** is used when teleoperation is active. At this point, the retargeting module streams in real time retargeted waist height and feet positions to the state machine, as well as retargeted upper-body joint positions, to the controller.

- (a) **Double support** is the state which is first used, when teleoperation starts. It is used for movements that keep both feet on the ground. The ground projection of the CoM is set to lie in the middle between the feet, which remain in place on the ground. Reference poses for the postural and waist height tasks are continuously updated from the streamed retargeted joint positions and waist height.
- (b) *Walking* – used when walking is detected.
- i. **Walking double support** is used to move the ground projection of the CoM to the middle between the feet, while the feet remain in place. Reference poses for the postural tasks are continuously updated from the streamed retargeted joint positions. As long as the maximum number of footsteps to be performed (as defined in section 6.1.1) has not been reached, the size of the next footstep is adjusted according to teleoperation signals and the state machine moves on to state 2(b)ii when the error on CoM position is smaller than a threshold e_{CoM} . When the maximum number of footsteps has been reached, the state machine moves on to state 2a.
 - ii. **Foot liftoff** is used to move the CoM above the stance foot, and to lift the swing foot off the ground, up and in the walking direction, once the CoM error is smaller than a threshold e_{CoM} . Reference poses for the postural tasks are continuously updated from the streamed retargeted joint positions. The state machine moves on to state 2(b)iii once the swing foot has reached the desired footstep height, and half of the footstep size.
 - iii. **Foot touchdown** is used to bring the swing foot back down to the ground, at the position defined from the desired footstep size, and to move the CoM back between the feet, once the error on the foot pose is smaller than a threshold e_{foot} . Reference poses for the postural tasks are continuously updated from the streamed retargeted joint positions. Once the error on the CoM is smaller than a threshold e_{CoM} , the vertical contact force at the swing foot is above a threshold $f_{touchdown}$ and the swing foot has reached the desired pose, swing and stance feet are switched and the state machine moves on to state 2(b)i, in order to initiate the next footstep.

Note that references for the waist height and orientation tasks remain at their measured initial values, during the walking states.

Additional states can easily be added to achieve a larger diversity of behaviors, e.g. teleoperated single support or autonomous motions. Given that the main idea here is to show the potential use of the framework and its flexibility, we have allowed the state machine to define simple reference trajectories, but this is in effect limiting the potential of the motions that could actually be accomplished.

In fact, handling stability constraints and generating lower-body trajectories based on the divergent component of motion, as benchmarked in [Romualdi et al., 2018] and proposed in [Ishiguro et al., 2017, 2018], would be beneficial. Intrinsically stable model predictive control approaches [Dafarra et al., 2018; Scianca et al., 2016] can also be used for compliance to external interactions, with the proposed framework.

The next subsection explains the motion controller that tracks desired trajectories defined by the state machine.

6.1.3 Stack-of-tasks for teleoperated whole-body velocity-control of a humanoid robot

The stack-of-tasks considered for a whole-body velocity-controller is similar to that used for a torque-controller, but with some adaptations. For the purpose of the proposed framework, it also includes modifications for retargeting movements from a human operator, as described in section 6.1.1.

Since with velocity-control, contact stabilization may not be achieved with friction cone constraints, it is instead attempted by using a two-layer hierarchy of tasks, in which the tasks on the lower priority layer are achieved in the null space of the higher priority layer. The stabilization of the stance and swing foot pose are placed on the higher priority layer, while other tasks are on the lower priority layer. In this case, soft tasks will be used as prioritization scheme between tasks on the same hierarchy layer.

A stack-of-tasks is proposed to be defined from the following tasks, where the indentation denotes hierarchy.

- Stabilize the stance foot pose $T_{stance} \in SE(3)$
- Stabilize the swing foot pose $T_{swing} \in SE(3)$

- Stabilize the projection of the CoM on the ground $p_{CoM} \in \mathbb{R}^2$
- Stabilize the waist height $z_{waist} \in \mathbb{R}$
- Stabilize the waist frame orientation $R_{waist} \in SO(3)$
- Stabilize the neck frame orientation $R_{neck} \in SO(3)$
- Track upper-body joint positions $s_{up} \in \mathbb{R}^{n_{up}}$
- Minimize robot velocity ν

Regarding the upper-body postural task, n_{up} is the number of controlled DOFs on the upper-body postural task, which can be broken down into subtasks on the head (i.e. neck joints), torso and arms. In the particular implementation presented here, it includes 3 DOFs on the neck joints, 3 DOFs on the torso joints and 4 DOFs on the joints of each arm, for a total of 14 DOFs.

Note that both feet are equally considered with the highest priority, as opposed to the stance foot only. Indeed, encouraging the swing foot to be well posed at the moment of touchdown showed to have a positive impact on stability of the robot, during preliminary tests performed on the robot.

The task on waist orientation is added for the case of whole-body teleoperation in double support, where the controller can use these additional controlled DOFs in order to stabilize the whole-body motion.

Tasks are then attributed priorities with respect to each other, with the following set of task weights.

$$\mathbf{w} = \{w_{CoM}, w_{neck}, w_{s_{up}}, w_{\nu}\} \quad (6.8)$$

where the weight $w_{CoM} \in \mathbb{R}$ is associated to the stabilization of p_{CoM} , z_{waist} and R_{waist} . Then, $w_{neck}, w_s \in \mathbb{R}$ are associated to tasks on R_{neck} and s_{up} , respectively. Finally, $w_{\nu} \in \mathbb{R}$ is a parameter which controls the importance of the regularization on the velocity of the robot.

Given the Cartesian and postural tasks defined here for the controller, each of them may be stabilized as described in the following paragraphs.

Stabilization of Cartesian tasks

With velocity-control, the robot velocity ν is considered as the control input u . In consequence, the velocity $v_{\mathcal{T}}$ of the frame \mathcal{T} can be computed as a function of the

control input, from the associated Jacobian $J_{\mathcal{T}}$ and the robot velocity, as follows.

$$v_{\mathcal{T}}(u) = J_{\mathcal{T}}\nu \quad (6.9)$$

Stabilization of a Cartesian task, i.e. the pose $T_{\mathcal{T}}$ of a frame \mathcal{T} can then be attempted by minimizing the error on its velocity $\tilde{v}_{\mathcal{T}}(u)$, given a feedback term $v_{\mathcal{T}}^*$, as follows.

$$\tilde{v}_{\mathcal{T}}(u) = v_{\mathcal{T}}(u) - v_{\mathcal{T}}^* \quad (6.10)$$

In this case, the feedback term $v_{\mathcal{T}}^*$ may be computed with

$$v_{\mathcal{T}}^* = v_{\mathcal{T}}^d + K \left(p_{\mathcal{T}}^d - p_{\mathcal{T}}(q) \right) \quad (6.11)$$

where $p_{\mathcal{T}}$ is a vector representing the pose of the frame \mathcal{T} using Euler parameters for its rotation, the superscript d indicates desired values, and K is a feedback gain matrix with different gain values for linear and angular terms.

Stabilization of postural tasks

In order to stabilize a postural task, one can obtain a formulation of the joint velocities \dot{s} as a function of the control input, as

$$\dot{s}(u) = \zeta\nu \quad (6.12)$$

where $\zeta = (0_{n \times 6}, 1_n)^\top$ is a selector matrix.

Stabilization of the postural task may then be attempted by minimizing the error on the joint velocities $\tilde{\dot{s}}(u)$, given a feedback term \dot{s}^* .

$$\tilde{\dot{s}}(u) = \dot{s}(u) - \dot{s}^* \quad (6.13)$$

In this case, the feedback term \dot{s}^* can be computed with

$$\dot{s}^* = \dot{s}^d + K_s \left(s^d - s \right) \quad (6.14)$$

where the superscript d indicates desired values, and K_s is a feedback gain.

6.1.4 QP controller

Once feasible postural and Cartesian values are obtained for the robot, they can be set as reference set points for a whole-body controller.

In the case of the present whole-body velocity-controller, one is interested in managing the velocity of the robot ν as control input u to the robot. For this purpose, we define a QP-based velocity-controller based on the stack-of-tasks presented in subsection 6.1.3.

Additional safety constraints are added to the control problem. First, constraints on the joint limits of the robot are defined in the shape

$$\mu(s_{min} - s) \leq \zeta u \leq \mu(s_{max} - s) \quad (6.15)$$

where s_{min} , s_{max} are the minimum and maximum joint limits and μ is a scaling factor defined in OpenSoT. Then, constraints on the joint velocities are defined in the shape

$$-\dot{s}_{max} \leq \zeta u \leq \dot{s}_{max} \quad (6.16)$$

where \dot{s}_{max} is the maximum allowed joint velocity.

The following paragraphs describe the proposed optimization problem that computes a control input in order to achieve objectives.

The problem is formulated with a mixture of hard and soft task priorities, using the two-layer hierarchy of soft tasks introduced in subsection 6.1.3. In this case, the cost functions C_{high} , C_{low} of the higher and lower priority layers are defined as the weighted sum of task errors, as introduced in equations (6.10) and (6.13).

The control architecture can then be formulated as the following optimization problem:

$$u^* = \arg \min_u \frac{1}{2} C_{low} \quad (6.17a)$$

$$\text{s. t.} \quad (6.17b)$$

$$\mu(s_{min} - s) \leq \zeta u \leq \mu(s_{max} - s) \quad (6.17c)$$

$$-\dot{s}_{max} \leq \zeta u \leq \dot{s}_{max} \quad (6.17d)$$

$$u = \arg \min_u \frac{1}{2} C_{high} \quad (6.17e)$$

where the lower priority tasks accounted for in (6.17a) act in the null space of the higher priority tasks accounted for in (6.17e).

In our specific implementation, the cost function of each layer is adapted between the **walking** states and the others, for which a greater mobility can be achieved.

When in states other than **walking**, C_{high} includes tasks on stance foot, swing foot and head posture (neck joint velocities), while C_{low} includes tasks on CoM, waist height and orientation, as well as posture of torso and arms, and regularization of robot velocity:

$$C_{high} = |\tilde{v}_{stance}(u)|^2 + |\tilde{v}_{swing}(u)|^2 + w_s |\tilde{s}_{head}(u)|^2 \quad (6.18)$$

$$C_{low} = w_{CoM} (|\tilde{v}_{CoM}(u)|^2 + |\tilde{v}_{waist}(u)|^2) + w_{neck} |\tilde{v}_{neck}(u)|^2 + w_s |\tilde{s}_{up}(u)|^2 + w_\nu |\nu|^2 \quad (6.19)$$

where $\tilde{v}_{stance}(u)$, $\tilde{v}_{swing}(u)$, \tilde{v}_{CoM} , \tilde{v}_{neck} are the error on stance foot, swing foot, CoM, and neck Cartesian tasks, respectively. The waist height and orientation are included in \tilde{v}_{waist} , but \tilde{v}_{zwaist} (used below) only includes waist height, Also, \tilde{s}_{up} is the error on the upper-body postural task, with \tilde{s}_{head} for the neck joints specifically.

Instead, when in a **walking** state, C_{high} only includes tasks on stance and swing feet, while C_{low} includes tasks on CoM, waist height, as well as posture of the upper-body (head, torso and arms), and regularization of robot velocity:

$$C_{high} = |\tilde{v}_{stance}(u)|^2 + |\tilde{v}_{swing}(u)|^2 \quad (6.20)$$

$$C_{low} = w_{CoM} (|\tilde{v}_{CoM}(u)|^2 + |\tilde{v}_{zwaist}(u)|^2) + w_{neck} |\tilde{v}_{neck}(u)|^2 + w_s |\tilde{s}_{up}(u)|^2 + w_\nu |\nu|^2 \quad (6.21)$$

Reorganizing terms, the optimization problems of equations 6.17a and 6.17e can easily be formulated as QP.

Note that the controller could as well be formulated for position or force/torque control, using different control objectives and constraints.

6.2 Applications of the whole-body teleoperation framework

Two different applications of the framework are presented in the following subsections. In the first one, the application is limited to walking, but provided increasingly complex foot movements, as a way to verify that the framework can achieve generic footsteps. In the second one, the framework is applied to whole-body teleoperation, for the specific problems of whole-body teleoperation while balancing on two feet, and upper-body teleoperation while walking.

The whole-body controller described in section 6.1.4 is implemented using the OpenSoT software library, as introduced in [Hoffman et al., 2018, 2017; Rocchi et al., 2015]. OpenSoT is specifically developed to solve optimization problems based on a stack-of-tasks. At the moment this project took place, it was an open-source library for implementing QP controllers based on a stack-of-tasks¹. It offers interfaces for defining QP problems related to whole-body control, including the definition of tasks, as well as specifying cost functions and constraints.

Presented experiments are performed with the iCub, using 26 DOFs for whole-body control: 3 DOFs for the torso, 3 for the neck, 4 for each arm and 6 for each leg. The force-torque sensors in the feet of the robot are used to detect forces exchanged between the feet and the ground. In the case of simulation experiments, they are conducted using the open-source robot simulator Gazebo [Koenig and Howard, 2004].

For all applications presented below, parameters of the controller are defined as follows. Task weights for the controller (6.17) are set as in table 6.1.

Table 6.1 Task weights w used with the controller of equation (6.17)

w_{CoM}	w_{waist}	w_{neck}	w_{sup}	w_{ν}
1	1	0.6	0.05	0.1

The weight values are taken from the average results obtained in chapter 5, where task priorities have been optimized using domain randomization, as well as an objective function encouraging both robustness and performance on Cartesian tasks. w_{CoM}

¹Even though OpenSoT appears to be no more open source at the time of writing this thesis, the source code used in this project is still available at github.com/EnricoMingo/OpenSoT-superbuild.

and w_{waist} are attributed a value of 1, w_{neck} a value of 0.6 (as the average neck task value shown in table 5.2), while feet are already provided the highest priorities by the formulation of the optimization problem. w_{ν} is attributed a weight of 0.1, as suggested by expert users of the OpenSoT framework. In the end, the postural task weight w_{sup} is the only weight that needs manual tuning, so the tuning process is straightforward.

As for the feedback gains used for stabilization of Cartesian and postural tasks in equations (6.11) and (6.14), the default values provided by OpenSoT are used, except for the following orientation gains: they are set to a value of 1 for swing and stance feet tasks, and 0.1 for the neck orientation task.

Finally, the parameters used for the finite state machine as introduced in subsection 6.1.2 are defined as in table 6.2.

Table 6.2 Parameters of the finite state machine described in 6.1.2

parameter	value
e_{CoM}	0.03 m
e_{foot}	0.02 m
$F_{touchdown}$	80 N

In our specific implementation of the *walking* states, desired CoM and feet trajectories are generated using the minimum jerk trajectory generator of [Pattacini et al., 2010]. As a consequence, the robot walks more slowly than the teleoperator, and the waist height needs to be lowered by 0.01 m at foot liftoff and touchdown of short steps (≤ 0.04 m) for better stability. In retrospect, we realize that this choice limits the achievement of dynamic motions and would recommend to use other approaches, as suggested for example in [Ishiguro et al., 2017, 2018; Romualdi et al., 2018].

6.2.1 Application to walking

This section exposes a preliminary application of the framework presented in section 6.1 to autonomous walking. Experiments with the controller are performed in simulation and with the iCub, in order to first verify the effectiveness of the defined controller, stack-of-tasks and state machine. The retargeting module is therefore disabled for these experiments. The preliminary results shown here have been collected from 3 different

experiments, in which the robot either steps in place, walks forward, or performs generic footsteps.

Experiment 1 - Stepping in place

A first experiment performed on the robot validates that the proposed controller allows to achieve a similar stepping behavior as the controllers presented in chapter 2. In this case, the desired movement of the foot is set to be lifted 5 cm up, while keeping its position constant along the x - and y -axes. The stepping motion has been successfully tested on the iCub for 10 consecutive steps. Figure 6.7 shows the stepping behavior achieved with the robot.

Experiment 2 - Walking

A second experiment performed on the robot shows that the proposed controller can be used to achieve walking. In this case, the desired movement of the foot is set to be lifted 2.5 cm up, and brought 10 cm forward. The stepping motion has been successfully tested on the iCub over 7 consecutive steps. Figure 6.8 shows the walking behavior achieved with the robot.

Experiment 3 - Generic footsteps in simulation

Simulation experiments have then been conducted for walking with a variety of footsteps, to assess the robustness of the controller with respect to desired foot trajectories. In simulation experiments, the controller shows to be robust to arbitrary footsteps, allowing foot displacements along x - and y -axes as well as foot rotations about the z -axis, while walking. Figure 6.9 displays some examples of the walking behaviors achieved by varying desired feet trajectories. Each of those have been tested for stability over at least 15 steps. This experiment has not yet been performed on the robot, due to time constraints and because it was deemed more interesting to move on to teleoperation at this point.

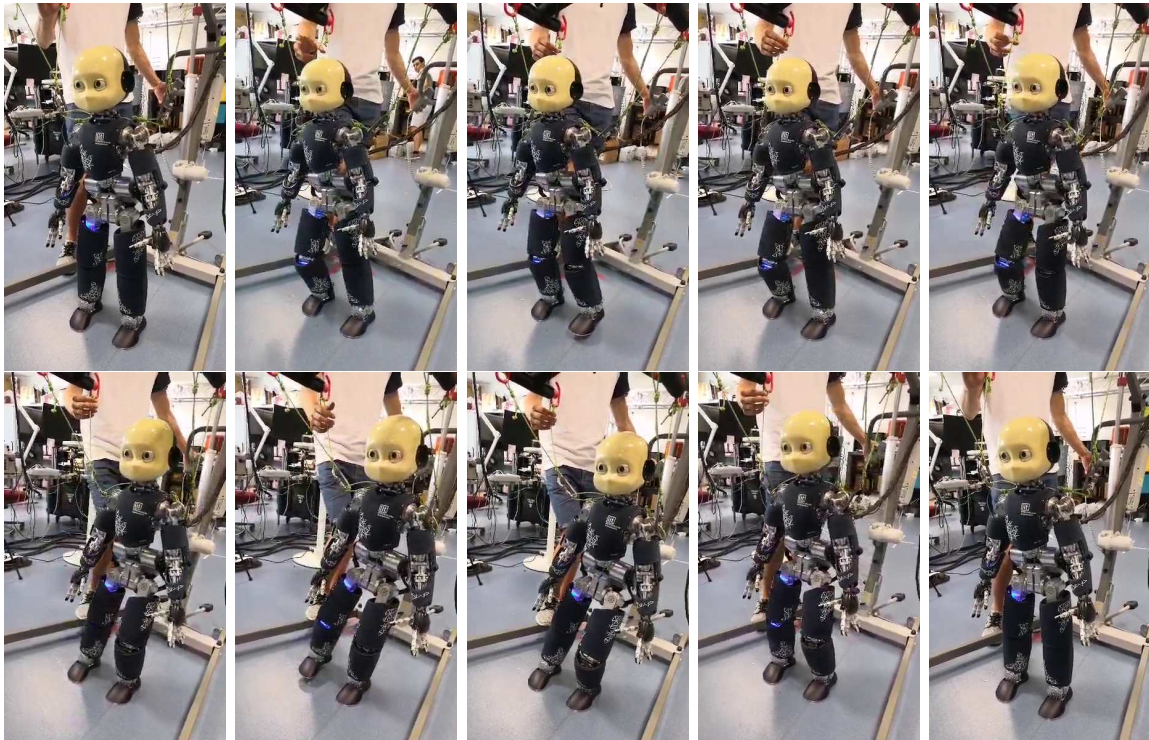


Figure 6.7 Walking experiment 1: snapshots of the robot stepping in place for the first step (above) and last step (below): transferring the weight to the stance leg, lifting the swing foot, bringing the swing foot back down and transferring the weight back to both legs.

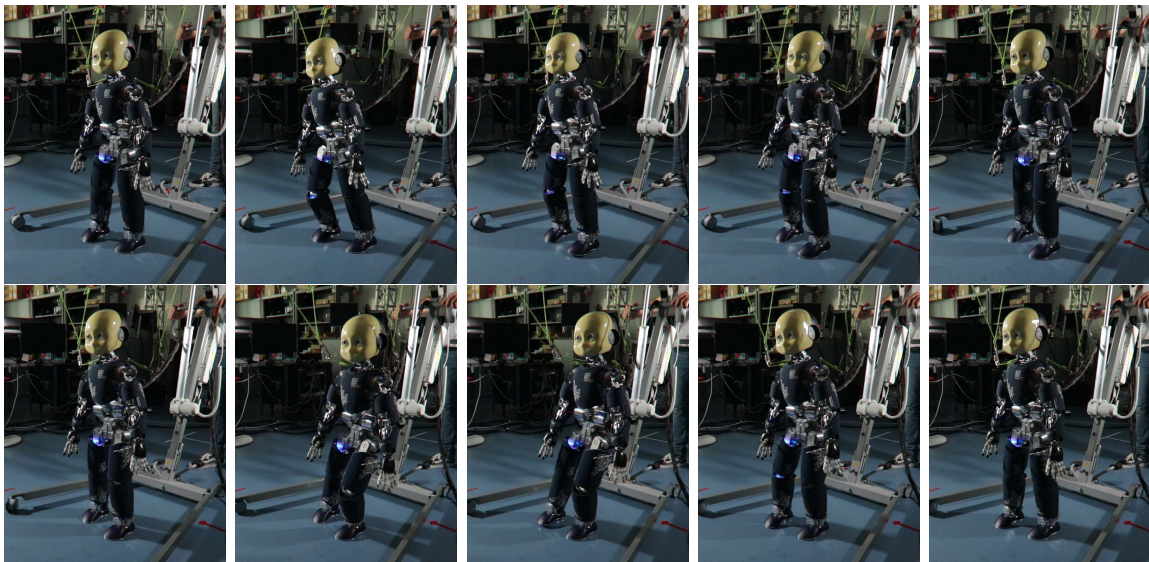
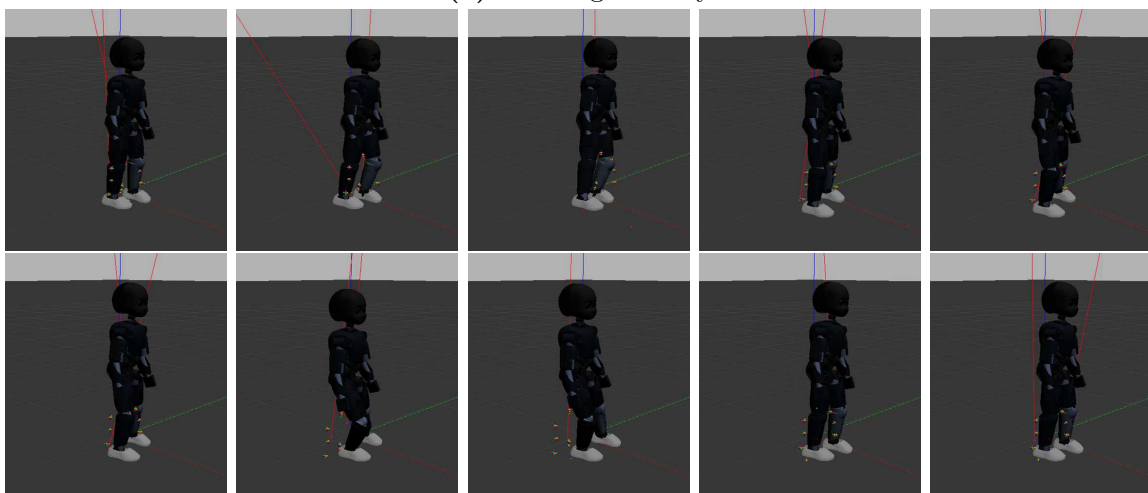


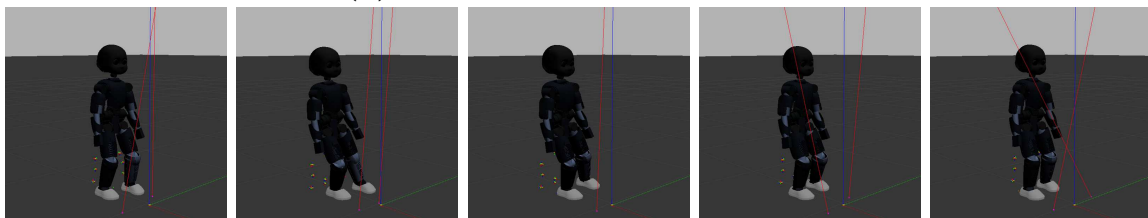
Figure 6.8 Walking experiment 2: snapshots of the robot walking forward for two consecutive steps: transferring the weight to the stance leg, lifting the swing foot forward, bringing the swing foot back forward and down and transferring the weight back to both legs.



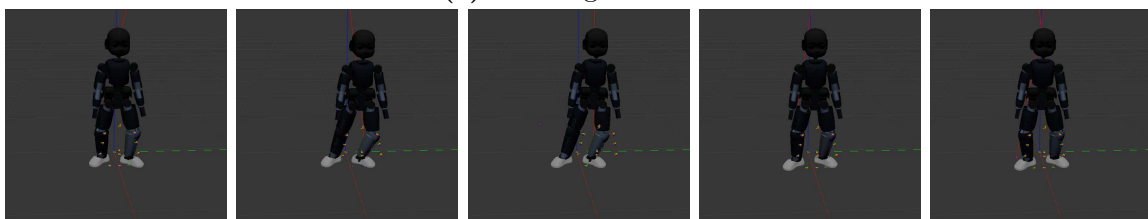
(a) Walking sideways



(b) Walking forward and to the right



(c) Walking backward



(d) Walking with wide open feet

Figure 6.9 Walking experiment 3: snapshots of various walking behaviors achieved with the simulated robot.

6.2.2 Application to whole-body teleoperation

This section exposes experiments performed with a human teleoperator¹ and the iCub robot, in order to validate that the proposed framework is capable of generating complex whole-body motions with teleoperation including simultaneous upper-body and lower-body motions.

Experiments involve the use of a human motion capture system, which allows to provide real-time estimation of the teleoperator's posture. The Xsens MVN system [Xsens, 2017] is used for this purpose. It is a wearable system consisting of 17 IMUs, that considers a model of the human with 66 DOFs corresponding to 22 spherical joints. The motion capture data of the Xsens can be visualized using the MVN Animate software that generates a 3D human reconstructed skeleton, allowing to visually compare the movements of the teleoperator and the robot.

Data captured with the Xsens is then used for motion retargeting to the robot, as described in section 6.1.1.

Results shown here have been collected from 3 different teleoperation experiments, in which the teleoperator begins in a resting pose (standing with the arms along the body), and performs a sequence of movements, as described in the following paragraphs.

Experiment 1

The first experiment consists in the following sequence of movements.

1. Wave the left hand
2. Perform rotation movements of the head
3. Lift both arms
4. Bring arms forward
5. Perform two squats
6. Bring arms down
7. Walk four steps forward
8. Wave the right arm

¹Note that in our experiments the operator is a sociologist, not an expert in robotics.

It has been performed on the simulated and the real robot. Snapshots of squatting, arm and walking motions, obtained with the virtual teleoperator and the simulated iCub, are superposed in figure 6.11; they show that the robot closely follows the movements of the teleoperator.

Figure 6.10 shows the upper-body movements obtained in the real iCub for experiment 1, starting from the beginning of the sequence of movements, until the arms are moved forward. The graphs show that the head, arms and torso roll follow closely the orientation of the teleoperator, within the physical limits of the robot.

Also, figure 6.12 shows that the retargeted footstep lengths have been adapted to those of the teleoperator.

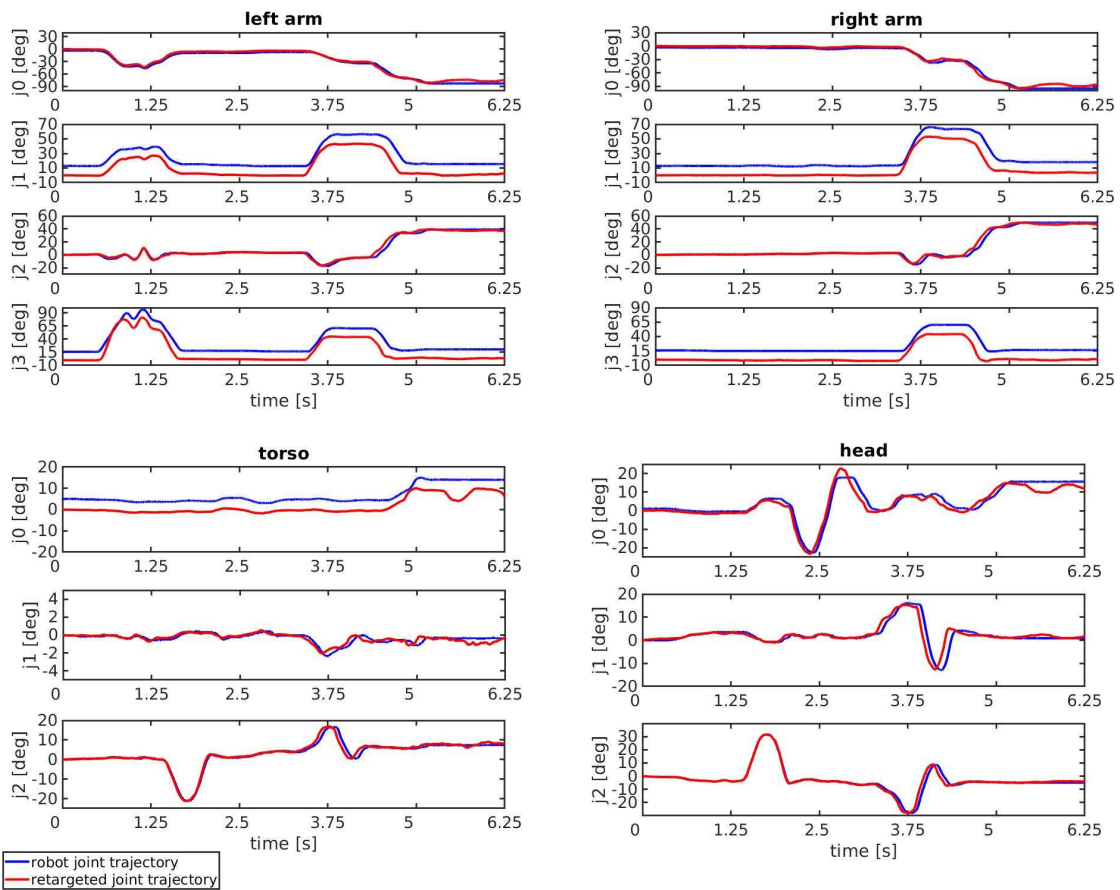


Figure 6.10 Teleoperation experiment 1: results of retargeting upper-body joint motions on the real robot. The numbering of joints corresponds to the order in which they are listed in figure 6.5.

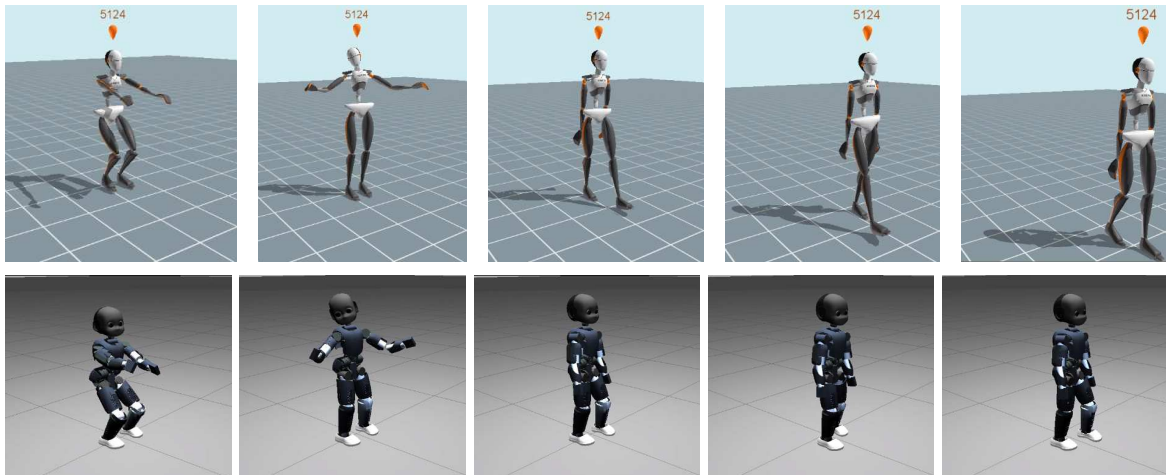
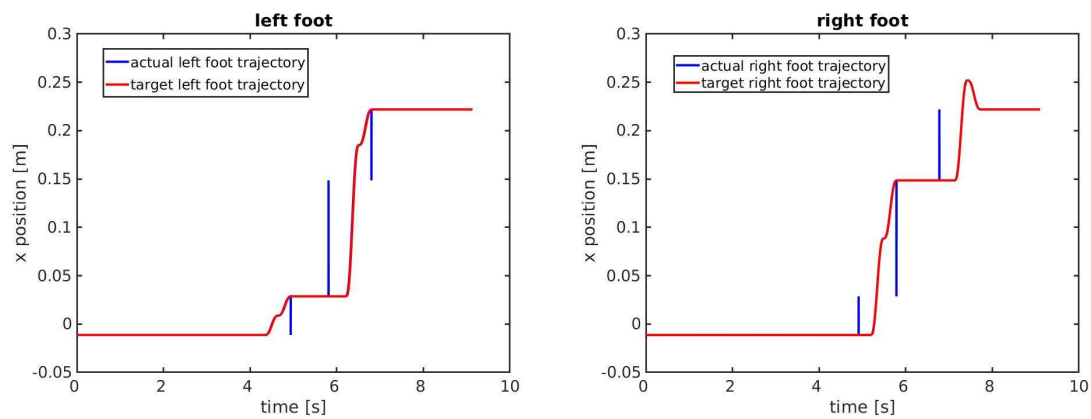
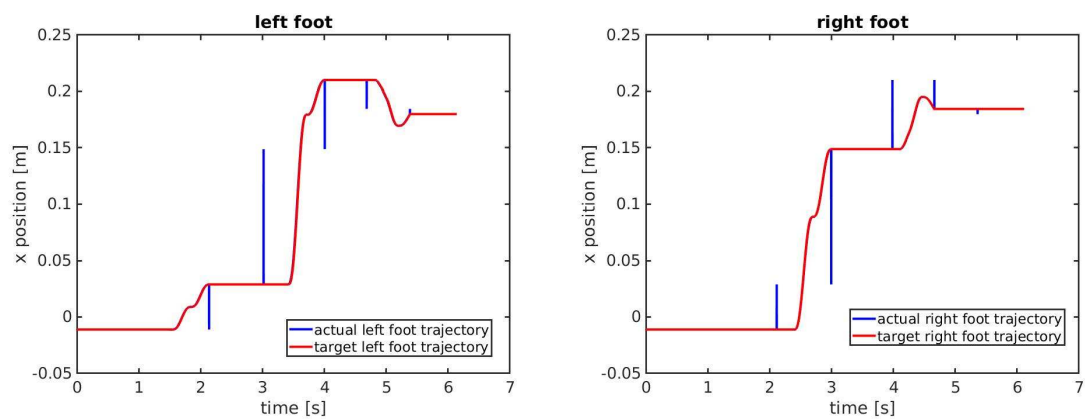


Figure 6.11 Teleoperation experiment 1: snapshots of squatting and walking movements taken with the human motion capture system and the simulated robot.



(a) Experiment 1



(b) Experiment 2

Figure 6.12 Teleoperation experiments 1 and 2: left and right feet motions achieved with the simulated robot, in the walking direction (x -axis). Above: experiment 1, below: experiment 2. The target length and direction of a footstep is determined from the teleoperator's step.

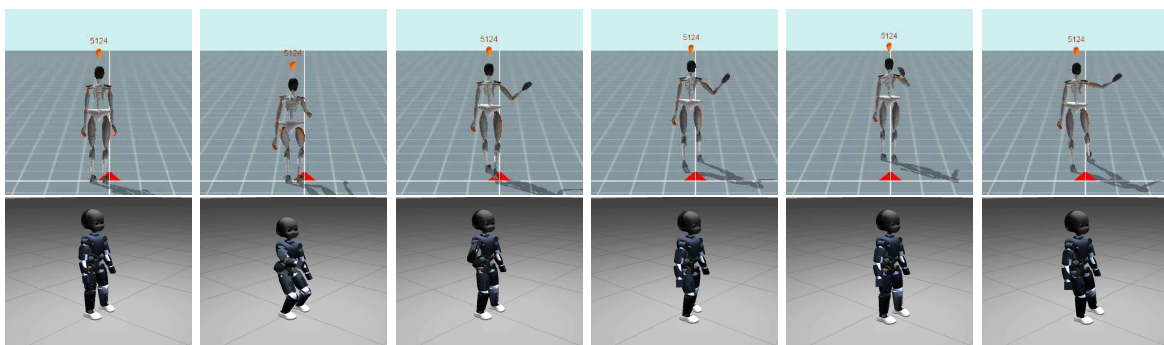
Experiment 2

The second experiment has been performed in simulation, and consists in the following sequence of movements.

1. Bring the right arm forward while performing a squat
2. Walk two steps forward while waving the right arm
3. Walk two steps backward while waving the right arm

Snapshots of squatting and walking steps, obtained with the virtual teleoperator and the simulated iCub, are superposed in figure 6.13. They show that the robot closely follows the movements of the teleoperator when squatting and moving the arm. Note that the robot walks more slowly than the teleoperator: the operator walked “normally” without waiting for the robot to complete each footstep. For this reason, the operator has finished waving the hand before the robot starts its second step, which explains the discrepancy in arm positions between the operator and robot.

The retargeted footstep lengths have been adapted to those of the teleoperator, as shown in figure 6.12. For example, the left foot of the robot performs a step backward in experiment 2; the last footstep then equalizes the position of the two feet. Moreover, the foot trajectories show to follow closely the desired trajectories; only at foot touchdown, an impulse can be observed in the measured feet positions, due to the impact with the ground.



(a) Initial pose (b) Squatting (c) Step (d) Step (e) Stop (f) Step back

Figure 6.13 Teleoperation experiment 2: snapshots of squatting, upper-body and walking movements taken with the human motion capture system and the simulated robot.

Experiment 3

The third experiment is performed as live teleoperation of the robot. It consists in the following sequence of movements.

1. Walk four steps forward
2. Wave the right arm and bring it back down
3. Wave the left arm and bring it back down
4. Lift both arms forward and drop arms down
5. Wave the right arm
6. Lift both arms forward
7. Perform two squats
8. Wave the left arm
9. Wave the right arm
10. Open arms in cross.

Figure 6.14 shows snapshots of the teleoperator and robot performing each of these movements. In particular, even though, as discussed above, the robot walks slower than the teleoperator, upper-body movements are still being retargeted in real time. As a result, while the robot is completing the number of steps performed by the teleoperator, it can be seen simultaneously walking and following the teleoperator's arm movements. This behavior is not ideal for legibility and can be corrected with an improved foot trajectory generator as suggested above. Nonetheless, this result is interesting because it shows that the whole-body control framework allows the robot to keep balance while effectively achieving upper-body and lower-body movements. In particular, the robot shows to be following the teleoperator's upper-body movements while walking, as shown in the top pictures of figure 6.14.

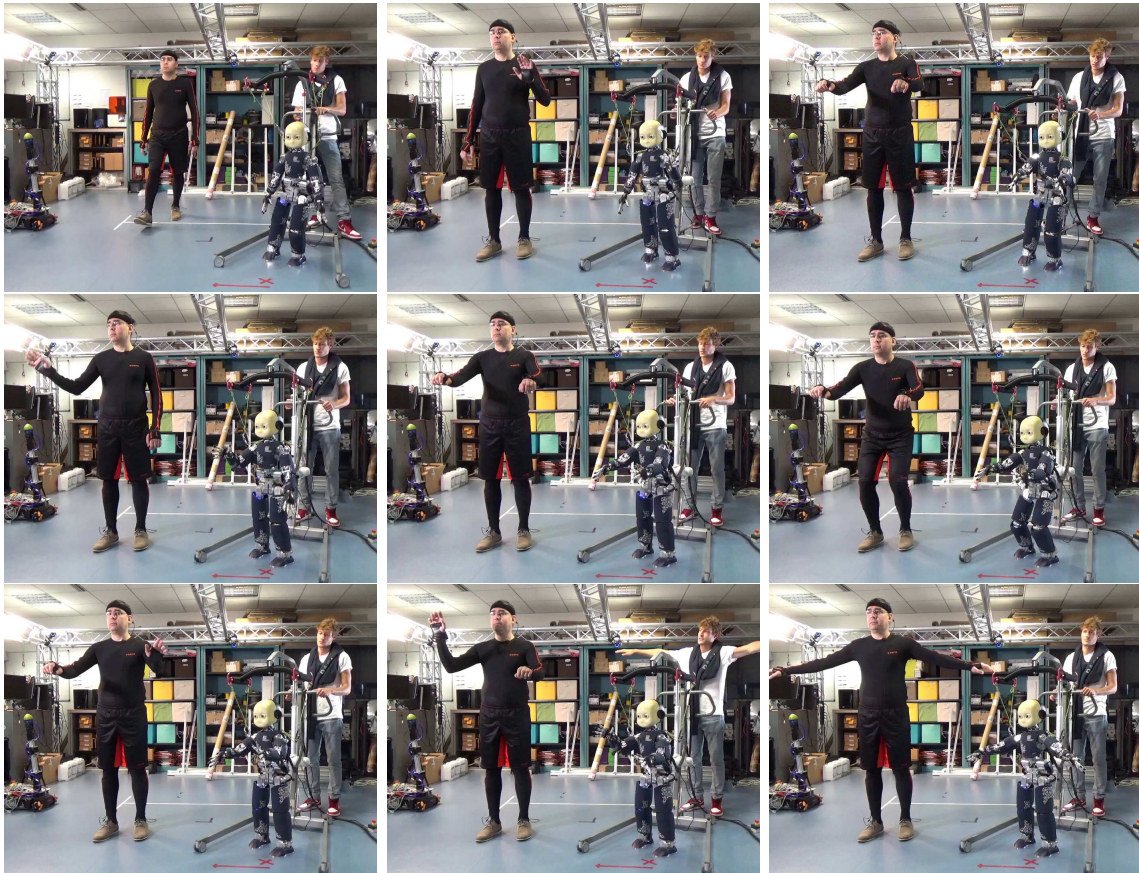


Figure 6.14 Teleoperation experiment 3: snapshots of whole-body teleoperation on the real robot: walking, waving arms, squatting, waving arms again and opening them in cross.

6.3 Conclusion

In summary, we have proposed a whole-body teleoperation framework that allows the robot to perform generic whole-body motions, such as upper-body motions while moving the lower-body, i.e., walking or squatting. The human-likeness of the robot motion is achieved thanks to a complete retargeting of the human upper-body joints, while the lower-body retargeting is formulated to guarantee the stability of the robot. The effectiveness of our approach has been demonstrated by teleoperating the iCub while performing the aforementioned movements. Results achieved with the proposed approach show that teleoperation of the upper-body can be successfully performed while walking. Results in simulation also show that the approach could be used for generic walking steps.

The main limitation of our framework is the velocity of the robot walking movements. This is due to limitations of the walking pattern generator we use, and more specifically to the use of the minimum jerk trajectory generator, which produces smooth trajectories for the legs. In next iterations, the framework shall be upgraded to achieve state of the art dynamic walking, including for example stability constraints, as well as MPC-based methods. Indeed, state of the art MPC approaches have proven to be intrinsically stable and to make a teleoperated robot compliant with respect to unexpected external interactions [Scianca et al., 2016], [Scianca et al., 2017].

From another perspective, the walking movement has been teleoperated only in a limited way, by scaling trajectories of the feet with respect to those of the teleoperator. It could then be interesting to investigate ways to achieve a more complete teleoperation of the human walking behavior. For instance, to achieve a more human-like walking, the retargeting of specific joints on the legs may be included, such as in [Hu et al., 2014]. To make the controller even more general, we may also consider, for example, single stance motions (retargeting swing leg movements) into the framework, and extending the footstep retargeting to generic footstep trajectories including changes in orientation.

Regarding the stack-of-tasks, experiments have shown the importance of considering both feet with the highest priority, as opposed to the stance foot only (as described in section 6.1.3), since this was beneficial for the swing foot to be well posed at the moment of touchdown. The formulation of the stack-of-tasks is likely at cause here, since only one stance foot is considered at a time: the swing foot is properly considered as a stance foot only when the swing and stance feet are switched, at the end of a step. In reality however, both feet should be considered as stance feet when they are both in contact with the ground. This is something which could easily be improved in the next iteration of the controller by defining as stance foot any foot which is currently in contact with the ground, and as swing foot any foot which is not currently in contact with the ground. Note also that, in the implementation of the controller, the head posture is considered with equal priority as the feet, when in the double support state (but not in walking states). This appears to be simply due to a quick implementation, and can also be fixed in a next iteration.

Finally, a more serious limitation of the controller is the fact that it does not consider contact constraints, which may allow the feet of the robot to slip on the ground. This is however not a desired behavior when walking. The most sensible solution in this

case would be to eventually transfer the control problem to a torque-controller. Future works in this direction should include the adaptation of OpenSoT for walking tasks with torque-control. Another possibility could be to adopt an alternative library such as the open-source Efficient Task Space Inverse Dynamics (TSID) library [Stack Of Tasks development team, 2019].

Nonetheless, the formulation of the framework, making use of a finite state machine and QP-based controller to achieve whole-body motions, allows for substantial flexibility and is worth pursuing further.

In particular, an interesting property of the proposed framework is that it can be used with state of the art QP-based whole-body controllers. This makes it possible to seamlessly switch between autonomous and teleoperated whole-body control, since the same controller can be used in both situations. Such a feature can become extremely useful in teleoperation scenarios where the communication between operator and robot can suffer losses or unexpected interruptions. Future work shall delve deeper into the integration of teleoperated and autonomous behaviors into the framework.

In conclusion, as outlined in the paragraphs above, our work on whole-body teleoperation of humanoid robots presented in this last chapter appears to be only a beginning: the promising results achieved with our method open many possibilities for exciting future developments.

Chapter 7

Conclusion

This thesis has explored subjects related to whole-body control, and proposes methods to increase the robustness of humanoid robots. Advances in this direction are particularly important, because achieving robust, autonomous and complex behaviors is essential to the development of service robots that can effectively deal with the real world.

While based on whole-body control, contributions have been developed along three more axes: joint limit avoidance, parameter tuning, and generalizing the whole-body motions that can be achieved by a controller.

7.1 Whole-body balancing torque-control

While developing a new whole-body torque-controller for the iCub, we have found that treating Cartesian and postural tasks with soft priorities helps in achieving smoother behaviors of the robot. However, many different formulations of a controller can actually lead to similar results, for instance using different strict and soft task hierarchy combinations, finite state machines or feedback control policies. It is up to the control expert to carefully design the controller, in order to make its implementation easier, but this most likely unfolds as an incremental process.

Additionally, results achieved with the framework developed in chapter 2 show Cartesian and postural tasks to be suitable for balancing and walking of a humanoid robot through torque-control. For instance, further results achieved using this framework,

given reference trajectories obtained from a receding horizon controller, successfully achieved walking of the robot in [Dafarra et al., 2018].

Nonetheless, additional methods addressing the robustness and tuning of the controllers would improve performance. This is what the rest of the thesis has then been concerned with.

7.2 Joint limit avoidance

Defining a parametrization of the feasible joint space of the robot has allowed to develop a theoretically guaranteed method for joint limit avoidance, which is adapted for torque-control. It has been shown to allow the robot to remain compliant, while resisting external perturbations in the case where joint limits are approached. When applied to whole-body control, it also shows to be beneficial for the robustness of the controller and produces smoother reactions to external perturbations.

Clearly, this approach is limited by the maximal torques which actuators may actually apply, and by proper tuning of the feedback gains. The fact that the parametrization creates a discontinuity when a joint limit is actually reached must however be addressed, but it can be as simple as introducing saturation.

7.3 Tuning parameters of whole-body controllers

While controllers developed in chapter 2 have shown to require particular efforts in the tuning of their parameters, it was not clear how much of a general issue it was, with controllers based on quadratic programming. A survey sent out to the robotics community has allowed to confirm that it indeed is, in a more formal way than simple anecdotal evidence. It has also confirmed that tuning is generally done by hand, through a process which can be described as tedious and time consuming, and there is a need to develop tools to make it easier.

In this case, our method for learning task priorities, taking advantage of domain randomization, has shown in chapter 5 to be highly promising. It automatically adjusts task priorities in simulation, while allowing to increase the capability of the controller to cope with disturbances, changes in working conditions and different platforms.

Ultimately, using the optimized task priorities has shown to allow transferring results between different robot models, without the need to re-tune the controller. It can potentially be of great help in bridging the reality gap.

So far, the proposed method is adapted for learning task priorities in simulation, but other parameters also require tuning. For instance, gains used for computing feedback terms may also have an important impact on the performance of a controller. Nonetheless, since robot simulation models are not perfect reflections of the reality (for example, motor backlash and stiction may be overlooked), it is possible that gains need different values between simulated and real-world robots. However, using the proposed method directly on the robot is potentially risky. Additional measures to finely tune controller parameters directly on the robot may eventually be useful as well.

7.4 Whole-body control for generic motions

Due to the tedious manual process of tuning controller parameters, controllers are generally well defined to achieve a specific action of the robot. For example, controllers used in chapters 2, 3 and 5 are specialized for balancing on a robot's feet and walking by moving the feet and center of mass in a predefined way. It remains, nevertheless, that envisioned applications for humanoid robots do not confine them to such limited tasks. Sooner or later, humanoid robots will be called to perform actions with their upper-body, while walking. The control framework presented in chapter 6 is a first step in this direction, allowing to achieve generic footsteps, as well as generic and human-like upper-body movements. In particular, it has shown to allow teleoperating upper-body movements in real-time, while the robot is simultaneously walking, taking footsteps adjusted according to those of the teleoperator.

Robustness to the various trajectories generated by the teleoperation has been achieved by controlling the center of mass such that it remains in a stable position. In the implementation presented in chapter 6, the feet trajectories are not yet retargeted in real-time from the motion of the teleoperator. While walking slower might help stability during static walking, the walk of the robot appears slow compared to the movements of the upper-body. Further developments for the teleoperation of the lower-body would clearly improve the capabilities of the robot.

7.5 Closing remarks

In conclusion, exciting directions for future research may include (but are far from being limited to) testing joint limit avoidance in more challenging scenarios with the real robot, addressing the tuning of feedback gains and exploring real-time walking teleoperation. Eventually, exploring additional approaches that allow to increase even more the robustness achieved with whole-body controllers, promise to be highly beneficial. Not only will it save time and effort when deploying controllers, but it shall have a significant impact on the autonomy of robots.

References

- Antonova, R., Cruciani, S., Smith, C., and Kragic, D. (2017). Reinforcement learning for pivoting task. *CoRR*, abs/1703.00472.
- Arnold, D. V. and Hansen, N. (2012). A (1+1)-cma-es for constrained optimisation. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO '12*, pages 297–304.
- Assal, S. F. M., Watanabe, K., and Izumi, K. (2005). Neural network learning from hint for the inverse kinematics problem of redundant arm subject to joint limits. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1477–1482.
- Atawnih, A., Papageorgiou, D., and Doulgeri, Z. (2016). Kinematic control of redundant robots with guaranteed joint limit avoidance. *Robotics and Autonomous Systems*, 79:122 – 131.
- Ayusawa, K. and Yoshida, E. (2017). Motion retargeting for humanoid robots based on simultaneous morphing parameter identification and motion optimization. *IEEE Transactions on Robotics*, 33(6):1343–1357.
- Bhat, S. P. and Bernstein, D. S. (2000). A topological obstruction to continuous global stabilization of rotational motion and the unwinding phenomenon. *Systems & Control Letters*, 39(1):63 – 70.
- Bischoff, B., Nguyen-Tuong, D., van Hoof, H., McHutchon, A., Rasmussen, C. E., Knoll, A., Peters, J., and Deisenroth, M. P. (2014). Policy search for learning robot control using sparse data. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3882–3887.
- Boston Dynamics, i. (2018). Atlas - the world’s most dynamic humanoid. <https://www.bostondynamics.com/atlas>. [Online; accessed 19 December 2018].
- Brygo, A., Sarakoglou, I., Tsagarakis, N., and Caldwell, D. G. (2014). Tele-manipulation with a humanoid robot under autonomous joint impedance regulation and vibrotactile balancing feedback. In *2014 IEEE-RAS 14th International Conference on Humanoid Robotics (Humanoids)*, pages 862–867.
- Caron, S., Pham, Q., and Nakamura, Y. (2015). Stability of surface contacts for humanoid robots: Closed-form formulae of the contact wrench cone for rectangular support areas. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5107–5112.

- Chan, T. F. and Dubey, R. V. (1995). A weighted least-norm solution based scheme for avoiding joint limits for redundant joint manipulators. *IEEE Transactions on Robotics and Automation*, 11(2):286–292.
- Charbonneau, M., Modugno, V., Nori, F., Oriolo, G., Pucci, D., and Ivaldi, S. (2018). Learning robust task priorities of qp-based whole-body torque-controllers. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9.
- Charbonneau, M., Nori, F., and Pucci, D. (2016). On-line joint limit avoidance for torque controlled robots by joint space parametrization. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 899–904.
- Charbonneau, M., Penco, L., Nori, F., Pucci, D., and Ivaldi, S. (2019). A comprehensive framework for qp-based whole-body teleoperation. Manuscript submitted for publication to 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).
- Chen, J.-L. and Liu, J.-S. (2002). Avoidance of obstacles and joint limits for end-effector tracking in redundant manipulators. In *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.*, volume 2, pages 839–844 vol.2.
- Chen, L. and Guo, Y. (2006). Hierarchical nonholonomic path planning of dual-arm space robot systems with joint limits. In *2006 6th World Congress on Intelligent Control and Automation*, volume 2, pages 8862–8865.
- Clever, D., Harant, M., Mombaur, K. D., Naveau, M., Stasse, O., and Endres, D. (2017). Cocomopl: A novel approach for humanoid walking generation combining optimal control, movement primitives and learning and its transfer to the real robot HRP-2. *IEEE Robotics and Automation Letters*, 2(2):977–984.
- Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, 521:503–507.
- Dafarra, S., Nava, G., Charbonneau, M., Guedelha, N., Andradel, F., Traversaro, S., Fiorio, L., Romano, F., Nori, F., Metta, G., and Pucci, D. (2018). A control architecture with online predictive planning for position and torque controlled walking of humanoid robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9.
- Dariush, B., Gienger, M., Arumbakkam, A., Zhu, Y., Jian, B., Fujimara, K., and Goerick, C. (2009). Online transfer of human motion to humanoids. *International Journal of Humanoid Robotics*, 06(02):265–289.
- Dehio, N., Reinhart, R. F., and Steil, J. J. (2015). Multiple task optimization with a mixture of controllers for motion generation. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6416–6421.
- Del Prete, A. and Mansard, N. (2016). Robustness to joint-torque-tracking errors in task-space inverse dynamics. *IEEE Transactions on Robotics*, 32(5):1091–1105.

- Dietrich, A., Wimböck, T., and Albu-Schäffer, A. (2011). Dynamic whole-body mobile manipulation with a torque controlled humanoid robot via impedance control laws. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3199–3206.
- Dragan, A. D., Lee, K. C., and Srinivasa, S. S. (2013). Legibility and predictability of robot motion. In *Proceedings of the 8th ACM/IEEE International Conference on Human-robot Interaction, HRI '13*, pages 301–308.
- Elobaid, M., Hu, Y., Babic, J., and Pucci, D. (2018). Telexistence and teleoperation for walking humanoid robots. *arXiv preprint arXiv:1809.01578*.
- Escande, A., Mansard, N., and Wieber, P. B. (2010). Fast resolution of hierarchized inverse kinematics with inequality constraints. In *2010 IEEE International Conference on Robotics and Automation*, pages 3733–3738.
- euRobotics (2018). eurobotics mailing list used for dissemination related to robotics. <https://www.eu-robotics.net/eurobotics/newsroom/ mailing-list/index.html>. [Online; accessed 19 December 2018].
- Fava, A. D., Bouyarmane, K., Chappellet, K., Ruffaldi, E., and Kheddar, A. (2016). Multi-contact motion retargeting from human to humanoid robot. In *2016 IEEE-RAS 16th International Conference on Humanoid Robotics (Humanoids)*, pages 1081–1086.
- Feng, S., Whitman, E., Xinjilefu, X., and Atkeson, C. G. (2014). Optimization based full body control for the atlas robot. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 120–127.
- Ferreau, H., Kirches, C., Potschka, A., Bock, H., and Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363.
- Fiacco, F. and Luca, A. D. (2013). Fast redundancy resolution for high-dimensional robots executing prioritized tasks under hard bounds in the joint space. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2500–2506.
- Fritsche, L., Unverzag, F., Peters, J., and Calandra, R. (2015). First-person teleoperation of a humanoid robot. In *2015 IEEE-RAS 15th International Conference on Humanoid Robotics (Humanoids)*, pages 997–1002.
- Fukumoto, Y., Nishiwaki, K., Inaba, M., and Inoue, H. (2004). Hand-centered whole-body motion control for a humanoid robot. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 2, pages 1186–1191 vol.2.
- Fumagalli, M., Ivaldi, S., Randazzo, M., Natale, L., Metta, G., Sandini, G., and Nori, F. (2012). Force feedback exploiting tactile and proximal force/torque sensing. *Auton. Robots*, 33(4):381–398.

- Fumagalli, M., Randazzo, M., Nori, F., Natale, L., Metta, G., and Sandini, G. (2010). Exploiting proximal f/t measurements for the icub active compliance. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1870–1876.
- GdR-Robotique (2018). Groupement de recherche (gdr) en robotique, an important robotics mailing list in france. <http://www.gdr-robotique.org/annonces/>. [Online; accessed 19 December 2018].
- Guedelha, N., Kuppuswamy, N., Traversaro, S., and Nori, F. (2016). Self-calibration of joint offsets for humanoid robots using accelerometer measurements. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 1233–1238.
- Ha, S. and Liu, C. K. (2016). Evolutionary optimization for parameterized whole-body dynamic motor skills. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1390–1397.
- Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195.
- Heremans, F., der Noot, N. V., Ijspeert, A. J., and Ronsse, R. (2016). Bio-inspired balance controller for a humanoid robot. In *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 441–448.
- Herzog, A., Righetti, L., Grimminger, F., Pastor, P., and Schaal, S. (2014). Balancing experiments on a torque-controlled humanoid with hierarchical inverse dynamics. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 981–988.
- Herzog, A., Rotella, N., Mason, S., Grimminger, F., Schaal, S., and Righetti, L. (2016). Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots*, 40(3):473–491.
- Hoffman, E. M., Clément, B., Zhou, C., Tsagarakis, N. G., Mouret, J.-B., and Ivaldi, S. (2018). Whole-Body Compliant Control of iCub: first results with OpenSoT. In *IEEE/RAS ICRA Workshop on Dynamic Legged Locomotion in Realistic Terrains*, Brisbane, Australia.
- Hoffman, E. M., Rocchi, A., Laurenzi, A., and Tsagarakis, N. G. (2017). Robot control for dummies: Insights and examples using opensot. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 736–741.
- Hopkins, M. A., Hong, D. W., and Leonessa, A. (2015). Compliant locomotion using whole-body control and divergent component of motion tracking. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5726–5733.
- Hu, K., Ott, C., and Lee, D. (2014). Online human walking imitation in task and joint space based on quadratic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3458–3464.

- Hyon, S. H., Hale, J. G., and Cheng, G. (2007). Full-body compliant human-humanoid interaction: Balancing in the presence of unknown external forces. *IEEE Transactions on Robotics*, 23(5):884–898.
- Igel, C., Suttrop, T., and Hansen, N. (2006). A computational efficient covariance matrix update and a (1+1)-cma for evolution strategies. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06*, pages 453–460.
- Isenberg, D. R., Mclain, M. A., and Kakad, Y. P. (2010). Contact force measurement noise in the partial feedback linearization control of humanoid robots. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pages 257–262.
- Ishiguro, Y., Kojima, K., Sugai, F., Nozawa, S., Kakiuchi, Y., Okada, K., and Inaba, M. (2017). Bipedal oriented whole body master-slave system for dynamic secured locomotion with lip safety constraints. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 376–382.
- Ishiguro, Y., Kojima, K., Sugai, F., Nozawa, S., Kakiuchi, Y., Okada, K., and Inaba, M. (2018). High speed whole body dynamic motion experiment with real time master-slave humanoid robot system. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7.
- Ito, M., Kawatsu, K., and Shibata, M. (2010). Maximal admission of joint range of motion based on redundancy resolution for kinematically redundant manipulators. In *Proceedings of SICE Annual Conference 2010*, pages 778–782.
- Ivaldi, S., Babič, J., Mistry, M., and Murphy, R. (2016). Special issue on whole-body control of contacts and dynamics for humanoid robots. *Autonomous Robots*, 40(3):425–428.
- Jamone, L., Damas, B., Santos-Victor, J., and Takanishi, A. (2013). Online learning of humanoid robot kinematics under switching tools contexts. In *2013 IEEE International Conference on Robotics and Automation*, pages 4811–4817.
- Kanajar, P., Caldwell, D. G., and Kormushev, P. (2017). Climbing over large obstacles with a humanoid robot via multi-contact motion planning. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 1202–1209.
- Kawada Industries, i. (2018). Humanoid robot hrp-4. <http://global.kawada.jp/mechatronics/hrp4.html>. [Online; accessed 19 December 2018].
- Kermorgant, O. and Chaumette, F. (2011). Avoiding joint limits with a low-level fusion scheme. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 768–773.
- Khalil, H. (2002). *Nonlinear Systems*. Pearson Education. Prentice Hall.
- Kim, D., You, B.-J., and Oh, S.-R. (2013). *Whole Body Motion Control Framework for Arbitrarily and Simultaneously Assigned Upper-Body Tasks and Walking Motion*, pages 87–98. Springer Berlin Heidelberg, Berlin, Heidelberg.

- Koenemann, J., Burget, F., and Bennewitz, M. (2014). Real-time imitation of human whole-body motions by humanoids. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2806–2812.
- Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2149–2154.
- Koryakovskiy, I., Kudruss, M., Vallery, H., Babuška, R., and Caarls, W. (2018). Model-plant mismatch compensation using reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):2471–2477.
- Kuindersma, S., Deits, R., Fallon, M., Valenzuela, A., Dai, H., Permenter, F., Koolen, T., Marion, P., and Tedrake, R. (2016). Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455.
- Kuindersma, S., Permenter, F., and Tedrake, R. (2014). An efficiently solvable quadratic program for stabilizing dynamic locomotion. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2589–2594.
- Lee, S.-H. and Goswami, A. (2012). A momentum-based balance controller for humanoid robots on non-level and non-stationary ground. *Autonomous Robots*, 33(4):399–414.
- Liegeois, A. (1977). Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(12):868–871.
- Liu, M., Tan, Y., and Padois, V. (2015). Generalized hierarchical control. *Autonomous Robots*, 40(1):17–31.
- Lober, R. (2017). *Task Compatibility and Feasibility Maximization for Whole-Body Control*. Theses, UPMC.
- Luo, R. C., Perng, Y.-W., Shih, B.-H., and Tsai, Y.-H. (2013). Cartesian position and force control with adaptive impedance/compliance capabilities for a humanoid robot arm. In *2013 IEEE International Conference on Robotics and Automation*, pages 496–501.
- Mansard, N., Stasse, O., Evrard, P., and Kheddar, A. (2009). A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In *2009 International Conference on Advanced Robotics*, pages 1–6.
- Marey, M. and Chaumette, F. (2010). New strategies for avoiding robot joint limits: Application to visual servoing using a large projection operator. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6222–6227.
- Marsden, J. E. and Ratiu, T. S. (2010). *Introduction to Mechanics and Symmetry: A Basic Exposition of Classical Mechanical Systems*. Springer Publishing Company, Incorporated.

- Mason, S., Righetti, L., and Schaal, S. (2014). Full dynamics lqr control of a humanoid robot: An experimental study on balancing and squatting. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 374–379.
- Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The icub humanoid robot: An open platform for research in embodied cognition. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pages 50–56.
- Modugno, V., Chervet, U., Oriolo, G., and Ivaldi, S. (2016a). Learning soft task priorities for safe control of humanoid robots with constrained stochastic optimization. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 101–108.
- Modugno, V., Nava, G., Pucci, D., Nori, F., Oriolo, G., and Ivaldi, S. (2017). Safe trajectory optimization for whole-body motion of humanoids. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 763–770.
- Modugno, V., Neumann, G., Rueckert, E., Oriolo, G., Peters, J., and Ivaldi, S. (2016b). Learning soft task priorities for control of redundant robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 221–226.
- Moro, F. L., Gienger, M., Goswami, A., Tsagarakis, N. G., and Caldwell, D. G. (2013). An attractor-based whole-body motion control (wbmc) system for humanoid robots. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 42–49.
- Na, M., Yang, B., and Jia, P. (2008). Improved damped least squares solution with joint limits, joint weights and comfortable criteria for controlling human-like figures. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 1090–1095.
- Nava, G., Romano, F., Nori, F., and Pucci, D. (2016). Stability analysis and design of momentum-based controllers for humanoid robots. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 680–687.
- Ngo, K. B. and Mahony, R. (2006). Bounded torque control for robot manipulators subject to joint velocity constraints. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 7–12.
- Norton, A., Ober, W., Baraniecki, L., McCann, E., Scholtz, J., Shane, D., Skinner, A., Watson, R., and Yanco, H. (2017). Analysis of human–robot interaction at the darpa robotics challenge finals. *The International Journal of Robotics Research*, 36(5-7):483–513.
- Olfati-Saber, R. (2001). *Nonlinear Control of Underactuated Mechanical Systems with Application to Robotics and Aerospace Vehicles*. PhD thesis, Massachusetts Institute of Technology, Cambridge.
- Otani, K. and Bouyarmane, K. (2017). Adaptive whole-body manipulation in human-to-humanoid multi-contact motion retargeting. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 446–453.

- Otani, K., Bouyarmane, K., and Ivaldi, S. (2018). Generating assistive humanoid motions for co-manipulation tasks with a multi-robot quadratic program controller. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3107–3113.
- Ott, C., Lee, D., and Nakamura, Y. (2008). Motion capture based human motion recognition and imitation by direct marker control. In *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, pages 399–405.
- Ott, C., Roa, M. A., and Hirzinger, G. (2011). Posture and balance control for biped robots based on contact force optimization. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 26–33.
- Pan, S. and Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.
- Parmiggiani, A., Maggiali, M., Natale, L., Nori, F., Schmitz, A., Tsagarakis, N., Victor, J., Becchi, F., Sandini, G., and Metta, G. (2012). The design of the icub humanoid robot. *International Journal of Humanoid Robotics*, 9(4).
- Pattacini, U., Nori, F., Natale, L., Metta, G., and Sandini, G. (2010). An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1668–1674.
- Penco, L., Clément, B., Modugno, V., Mingo Hoffman, E., Nava, G., Pucci, D., Tsagarakis, N. G., Mouret, J. B., and Ivaldi, S. (2018). Robust real-time whole-body motion retargeting from human to humanoid. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 425–432.
- Prajna, S. and Jadbabaie, A. (2004). Safety verification of hybrid systems using barrier certificates. In Alur, R. and Pappas, G. J., editors, *Hybrid Systems: Computation and Control*, pages 477–492, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Prete, A. D. (2018). Joint position and velocity bounds in discrete-time acceleration/torque control of robot manipulators. *IEEE Robotics and Automation Letters*, 3(1):281–288.
- Pucci, D., Nava, G., and Nori, F. (2016a). Automatic gain tuning of a momentum based balancing controller for humanoid robots. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 158–164.
- Pucci, D., Romano, F., Traversaro, S., and Nori, F. (2016b). Highly dynamic balancing via force control. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 141–141.
- Righetti, L., Buchli, J., Mistry, M., Kalakrishnan, M., and Schaal, S. (2013). Optimal distribution of contact forces with inverse-dynamics control. *The International Journal of Robotics Research*, 32(3):280–298.

- Righetti, L. and Schaal, S. (2012). Quadratic programming for inverse dynamics with optimal distribution of contact forces. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 538–543.
- robotics worldwide (2018). robotics-worldwide mailing list, used for announcements of general value to the robotics community. <http://duerer.usc.edu/mailman/listinfo.cgi/robotics-worldwide>. [Online; accessed 19 December 2018].
- Robotis Co., L. (2018). Robotis op-2. <http://www.robotis.us/robotis-op2-us/>. [Online; accessed 19 December 2018].
- Rocchi, A., Hoffman, E. M., Caldwell, D. G., and Tsagarakis, N. G. (2015). Opensot: A whole-body control library for the compliant humanoid robot coman. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6248–6253.
- Romano, F., Nava, G., Azad, M., Čamernik, J., Dafarra, S., Dermý, O., Latella, C., Lazzaroni, M., Lober, R., Lorenzini, M., et al. (2018). The codyco project achievements and beyond: Toward human aware whole-body controllers for physical human robot interaction. *IEEE Robotics and Automation Letters*, 3(1):516–523.
- Romano, F., Traversaro, S., Pucci, D., Eljaik, J., Del Prete, A., and Nori, F. (2017). A whole-body software abstraction layer for control design of free-floating mechanical systems. In *IEEE Int. Conf. on Robotic Computing (IRC)*, pages 148–155.
- Romualdi, G., Dafarra, S., Hu, Y., and Pucci, D. (2018). A benchmarking of dcm based architectures for position and velocity controlled walking of humanoid robots. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9.
- Saab, L., Ramos, O. E., Keith, F., Mansard, N., Souères, P., and Fourquet, J. Y. (2013). Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics*, 29(2):346–362.
- Salini, J., Padois, V., and Bidaud, P. (2011). Synthesis of complex humanoid whole-body behavior: A focus on sequencing and tasks transitions. In *2011 IEEE International Conference on Robotics and Automation*, pages 1283–1290.
- Scianca, N., Cagnetti, M., Simone, D. D., Lanari, L., and Oriolo, G. (2016). Intrinsically stable mpc for humanoid gait generation. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 601–606.
- Scianca, N., Modugno, V., Lanari, L., and Oriolo, G. (2017). Gait generation via intrinsically stable mpc for a multi-mass humanoid model. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 547–552.
- Shin, S. Y. and Kim, C. (2010). On-line human motion transition and control for humanoid upper body manipulation. In *IROS*, pages 477–482.
- Sian, N. E., Yokoi, K., Kajita, S., Kanehiro, F., and Tanie, K. (2002). Whole body teleoperation of a humanoid robot - development of a simple master device using joysticks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2569–2574 vol.3.

- Siciliano, B. and Khatib, O. (2007). *Springer Handbook of Robotics*. Springer-Verlag, Berlin, Heidelberg.
- Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2008). *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition.
- Silvério, J., Calinon, S., Rozo, L. D., and Caldwell, D. G. (2018). Learning competing constraints and task priorities from demonstrations of bimanual skills. *CoRR*, abs/1707.06791.
- SoftBank Robotics, C. (2018). Nao. <https://www.softbankrobotics.com/emea/en/nao>. [Online; accessed 19 December 2018].
- Spitz, J., Bouyarmane, K., Ivaldi, S., and Mouret, J. B. (2017). Trial-and-error learning of repulsors for humanoid qp-based whole-body control. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 468–475.
- Stack Of Tasks development team (2019). Efficient task space inverse dynamics (tsid). <https://github.com/stack-of-tasks/tsid>.
- Stephens, B. J. and Atkeson, C. G. (2010). Dynamic balance force control for compliant humanoid robots. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1248–1255.
- Stilman, M., Nishiwaki, K., and Kagami, S. (2008). Humanoid teleoperation for whole body manipulation. In *2008 IEEE International Conference on Robotics and Automation*, pages 3175–3180.
- Su, Y., Wang, Y., and Kheddar, A. (2018). Sample-efficient learning of soft task priorities through bayesian optimization. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–6.
- Tassa, Y., Mansard, N., and Todorov, E. (2014). Control-limited differential dynamic programming. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175.
- Tee, K. P., Ge, S. S., and Tay, E. H. (2009). Barrier lyapunov functions for the control of output-constrained nonlinear systems. *Automatica*, 45(4):918 – 927.
- Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30.
- Traversaro, S. (2017). *Modelling, Estimation and Identification of Humanoid Robots Dynamics*. PhD thesis, Italian Institute of Technology.
- Traversaro, S., Del Prete, A., Ivaldi, S., and Nori, F. (2015). Inertial parameters identification and joint torques estimation with proximal force/torque sensing. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2105–2110.

- Wieland, P. and Allgöwer, F. (2007). Constructive safety using control barrier functions. *IFAC Proceedings Volumes*, 40(12):462 – 467. 7th IFAC Symposium on Nonlinear Control Systems.
- Xsens (2017). Xsens the leading innovator in 3d motion tracking technology. <http://www.xsens.com/products/xsens-mvn/>. [Online; accessed 19 December 2018].
- Yamane, K. and Hodgins, J. (2009). Simultaneous tracking and balancing of humanoid robots for imitating human motion capture data. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2510–2517.
- Yamane, K. and Nakamura, Y. (2003). Dynamics filter - concept and implementation of online motion generator for human figures. *IEEE Trans. Robotics and Automation*, 19(3):421–432.
- Zhang, Y., Wang, J., and Xia, Y. (2003). A dual neural network for redundancy resolution of kinematically redundant manipulators subject to joint limits and joint velocity limits. *IEEE Transactions on Neural Networks*, 14(3):658–667.

Appendix A

Additional material for the soft tasks controller #1

This appendix provides additional details concerning the soft tasks controller #1 presented in section 2.3.2. In particular, it details the finite state machine in section A.1, before stating the parameters used by the finite state machine, as well as the gains used by the feedback control policies, for simulation experiments in section A.2, and for experiments with the real robot in section A.3.

A.1 Finite state machine for the soft tasks controller #1

A finite state machine is used by the control framework to output desired setpoints for Cartesian and postural tasks, in function of the state of the robot. It is also used for gain scheduling, outputting proportional-derivative gains used in the computation of feedback control policies for the Cartesian and postural tasks. In this case, the state machine is applied to the whole-body motion of stepping in place, and is divided into 11 states for this purpose, as described in the following paragraphs.

1. **Balancing on two feet** is the initial state, where the robot is assumed to be standing on both feet. At this point, the Cartesian and postural task values are set to their initial values p_{CoM_0} , T_{left_0} , T_{right_0} , R_{root_0} , q_0 , and both feet are set to be in contact with the ground. The state machine moves on to state 2, if

the error on the CoM position is below a threshold e_{CoM} and a delay t_{min} has elapsed.

2. **Transition to left foot** is used to move the center of mass above the left foot.
 - The projection of the CoM onto the $x - y$ plane is set to be coincident with the origin of the left foot, plus a user-defined distance δ_{CoM} .
 - The desired postural task values are set to user-defined joint positions q_{des} .

The state machine moves on to state 3 when the error on the CoM position is below a threshold e_{CoM} and the vertical force measured at the right foot is smaller than a threshold $f_{liftoff}$.

3. **Left foot support** is used to lift the right foot above the ground, while the left foot remains in place.
 - The right foot is set to not be in contact with the ground anymore.
 - The projection of the CoM onto the $x - y$ plane is set to be coincident with the origin of the left foot frame, plus a user-defined distance δ_{CoM} .
 - The desired postural task values are set to user-defined joint positions q_{des} . In particular, the right knee is made to bend.
 - The desired position of the origin of the right foot frame is set to be at a user-defined distance δ_{right} from the left foot frame.

The state machine moves on to state 4 if the norm of the errors on the left leg joint positions is smaller than a threshold e_{legc} , the norm of the errors on the right leg joint positions is smaller than a threshold e_{leg} , the error on right foot position is smaller than a threshold e_{foot} , and the time elapsed in this state is over $t_{balancing}$.

4. **Preparing for right foot touchdown** is used to bring the right foot back down towards the ground.
 - The right foot remains not in contact with the ground.
 - The projection of the CoM onto the $x - y$ plane is set to be coincident with the origin of the left foot frame, plus a user-defined distance δ_{CoM} .
 - The desired postural task values are set to user-defined joint positions q_{des} .
 - The desired position of the origin of the right foot frame is set to be at a user-defined distance δ_{right} from the left foot frame.

The state machine moves on to state 5 if the norm of the errors on the left leg joint positions is smaller than a threshold e_{legc} , the norm of the errors on the right leg joint positions is smaller than a threshold e_{leg} , and the error on right foot position is smaller than a threshold e_{foot} .

5. **Right foot touchdown** is used to bring the right foot back in contact with the ground.
 - The right foot continues to be considered not in contact with the ground.
 - The projection of the CoM onto the $x - y$ plane is set to be coincident with the origin of the left foot frame, plus a user-defined distance δ_{CoM} .
 - The desired postural task values are set to user-defined joint positions q_{des} .
 - The desired position of the origin of the right foot frame is set to be at a user-defined distance δ_{right} from the left foot frame.

The state machine moves on to state 6 when the vertical force measured at the right foot is higher than a threshold $f_{touchdown}$.

6. **Transition to initial position** is used to bring back the robot as it was at the initialization of the state machine.
 - Both feet are set to be in contact with the ground.
 - The CoM position is set back to p_{CoM_0}
 - The desired postural task values are set back to q_0 .
 - The right foot pose is set back to T_{right_0} .

The state machine moves on to state 7 when the error on the CoM position is below a threshold e_{CoM} .

7. **Transition to right foot** is used to move the center of mass above the right foot.
 - The projection of the CoM onto the $x - y$ plane is set to be coincident with the origin of the right foot, plus a user-defined distance δ_{CoM} .
 - The desired postural task values are set to user-defined joint positions q_{des} .

The state machine moves on to state 8 when the error on the CoM position is below a threshold e_{CoM} and the vertical force measured at the left foot is smaller than a threshold $f_{liftoff}$.

8. **Right foot support** is used to lift the left foot above the ground, while the right foot remains in place.

- The left foot is set to not be in contact with the ground anymore.
- The projection of the CoM onto the $x - y$ plane is set to be coincident with the origin of the right foot frame, plus a user-defined distance δ_{CoM} .
- The desired postural task values are set to user-defined joint positions q_{des} . In particular, the left knee is made to bend.
- The desired position of the origin of the left foot frame is set to be at a user-defined distance δ_{right} from the right foot frame.

The state machine moves on to state 9 if the norm of the errors on the right leg joint positions is smaller than a threshold e_{legc} , the norm of the errors on the left leg joint positions is smaller than a threshold e_{leg} , the error on left foot position is smaller than a threshold e_{foot} , and the time elapsed in this state is over $t_{balancing}$.

9. **Preparing for left foot touchdown** is used to bring the left foot back down towards the ground.

- The left foot remains not in contact with the ground.
- The projection of the CoM onto the $x - y$ plane is set to be coincident with the origin of the right foot frame, plus a user-defined distance δ_{CoM} .
- The desired postural task values are set to user-defined joint positions q_{des} .
- The desired position of the origin of the left foot frame is set to be at a user-defined distance δ_{right} from the right foot frame.

The state machine moves on to state 10 if the norm of the errors on the right leg joint positions is smaller than a threshold e_{legc} , the norm of the errors on the left leg joint positions is smaller than a threshold e_{leg} , and the error on left foot position is smaller than a threshold e_{foot} .

10. **Left foot touchdown** is used to bring the left foot back in contact with the ground.

- The left foot continues to be considered not in contact with the ground.
- The projection of the CoM onto the $x - y$ plane is set to be coincident with the origin of the right foot frame, plus a user-defined distance δ_{CoM} .

- The desired postural task values are set to user-defined joint positions q_{des} .
- The desired position of the origin of the left foot frame is set to be at a user-defined distance δ_{right} from the right foot frame.

The state machine moves on to state 11 when the vertical force measured at the left foot is higher than a threshold $f_{touchdown}$.

11. **Transition to initial position** is used to bring back the robot as it was at the initialization of the state machine.

- Both feet are set to be in contact with the ground.
- The CoM position is set back to p_{CoM_0}
- The desired postural task values are set back to q_0 .
- The left foot pose is set back to T_{left_0} .

The state machine moves on to state 2 when the error on the CoM position is below a threshold e_{CoM} .

Note that across all states, the desired orientation of the root link is left untouched, remaining at R_{root_0} . The same is true for the orientation of the left and right feet.

Also, at each state, a different set of feedback gains, used for the computation of feedback terms for the Cartesian and postural tasks as in equations (2.11) and (2.13), is output by the state machine.

In order to smooth the transition of signal values (task setpoints or gains) from one state to the next, we use the minimum jerk trajectory generator developed in [Pattacini et al., 2010], which provides instantaneous desired positions, as well as desired velocities and accelerations in the case of task setpoints, given a user-defined smoothing time t_{smooth} . In this case, t_{smooth} is also attributed different values by the state machine, in function of the current state.

A.2 Parameter values defined for the implementation of the soft tasks controller #1 in simulation

This section details the values defined for implementing the soft tasks controller #1 presented in section 2.3.2, in simulation. In particular, it defines the parameters used by the finite state machine, as well as the gains used by the feedback control policies.

Table A.1 Parameters for the stabilization of contact tasks **in simulation experiments**: $f_{kz_{min}}$ is the minimal vertical reaction force for which a contact is considered to exist, n_v is the number of vertices used in the approximation of the friction cone, μ_c and μ_t are approximations of the Coulomb and torsional static friction coefficients associated to the contact surfaces.

parameter	value
$f_{kz_{min}}$	10 N
n_v	4
μ_c	1
μ_t	$\frac{1}{75}$

Table A.2 Task weights w used with the soft tasks controller #1 for simulation experiments

w_Υ	w_s	w_τ
100	0.3	1e-4

Table A.3 Parameters of the finite state machine used by the soft tasks controller #1 for simulation experiments. Parameters are introduced in A.1

parameter	value
e_{CoM}	0.01 m
e_{foot}	0.01 m
e_{leg}	5 deg
e_{legc}	5 deg
$f_{liftoff}$	60 N
$f_{touchdown}$	4 N
τ_{max}	60 N
$\dot{\tau}_{max}$	50 N/s
t_{min}	2 s
$t_{balancing}$	5 s
t_{smooth}	2 s

Table A.4 Proportional (P) feedback gains defined for Cartesian tasks **in simulation experiments**, along the x -, y - and z -axes for feedback on position, and about the same axes for feedback on rotation (denoted by $\theta_x, \theta_y, \theta_z$). For all P gains, the associated derivative (D) feedback gains are obtained as $D = 2\sqrt{P}$.

Task	Gain	State										
		1	2	3	4	5	6	7	8	9	10	11
CoM	P_x	50	50	50	50	50	50	50	50	50	50	50
	P_y	60	60	60	60	60	60	60	60	60	60	60
	P_z	50	50	50	50	50	50	50	50	50	50	50
left foot	P_x	20	0	0	0	0	0	0	20	20	20	0
	P_y	20	0	0	0	0	0	0	20	20	20	0
	P_z	20	0	0	0	0	0	0	20	20	20	0
	P_{θ_x}	5	5	5	5	5	5	5	10	10	10	5
	P_{θ_y}	5	5	5	5	5	5	5	10	10	10	5
	P_{θ_z}	5	5	5	5	5	5	5	10	10	10	5
right foot	P_x	20	0	20	20	20	0	0	0	0	0	0
	P_y	20	0	20	20	20	0	0	0	0	0	0
	P_z	20	0	20	20	20	0	0	0	0	0	0
	P_{θ_x}	5	5	10	10	10	5	5	5	5	5	5
	P_{θ_y}	5	5	10	10	10	5	5	5	5	5	5
	P_{θ_z}	5	5	10	10	10	5	5	5	5	5	5
root link	P_{θ_x}	5	10	10	5	10	15	10	10	5	10	15
	P_{θ_y}	5	10	10	5	10	15	10	10	5	10	15
	P_{θ_z}	5	10	10	5	10	15	10	10	5	10	15

Table A.6 Displacement of the Cartesian tasks, defined to achieve a stepping motion, along the x -, y - and z -axes **for simulation experiments**

δ	State										
	1	2	3	4	5	6	7	8	9	10	11
δ_{CoM_x}	-	0	0.01	0.01	0.01	-	0	0.01	0.01	0.01	-
δ_{CoM_y}	-	0.01	0	0	-2.01	-	-0.01	0	0	2.0054	-
δ_{CoM_z}	-	0	0	0	-0.86	-	0	0	0	-0.86	-
δ_{left_x}	-	-	-	-	-	-	0	0	0	0	-
δ_{left_y}	-	-	-	-	-	-	0	0.13	0.13	0.13	-
δ_{left_z}	-	-	-	-	-	-	0	0.05	0	0	-
δ_{right_x}	-	0	0	0	0	-	-	-	-	-	-
δ_{right_y}	-	0	-0.13	-0.13	-0.13	-	-	-	-	-	-
δ_{right_z}	-	0	0.05	0	0	-	-	-	-	-	-

Table A.7 Joint position values defined for simulation experiments in degrees according to the pitch-roll-yaw convention: θ is pitch, ψ is roll, and ϕ is yaw, and used by the postural task

Joint	State											
	1	2	3	4	5	6	7	8	9	10	11	
Torso	θ	-	-2.01	4.93	-2.01	-4.93	-	2.01	-4.93	2.01	4.93	-
	ψ	-	4.47	1.49	4.47	1.49	-	-4.47	-1.49	-4.47	-1.49	-
	ϕ	-	2.46	0.86	2.46	0.86	-	2.46	0.86	2.46	0.86	-
Left shoulder	θ	-	-8.55	7.18	-8.55	7.18	-	-8.55	3.23	-8.55	3.23	-
	ψ	-	49.16	46.61	49.16	46.61	-	49.16	38.90	49.16	38.90	-
	ϕ	-	13.96	17.48	13.96	17.48	-	13.96	19.14	13.96	19.14	-
Left elbow	θ	-	49.90	45.42	49.90	45.42	-	49.90	35.60	49.90	35.60	-
Right shoulder	θ	-	-8.55	3.23	-8.55	3.23	-	-8.55	7.18	-8.55	7.18	-
	ψ	-	49.16	38.90	49.16	38.90	-	49.16	46.61	49.16	46.61	-
	ϕ	-	13.96	19.14	13.96	19.14	-	13.96	17.48	13.96	17.48	-
Right elbow	θ	-	49.90	35.60	49.90	35.60	-	49.90	45.42	49.90	45.42	-
Left hip	θ	-	0	0	0	0	-	0	18	0	0	-
	ψ	-	-6.35	-6.35	-6.35	-4.25	-	4.54	4.54	4.54	1.29	-
	ϕ	-	0	0	0	0	-	0	0	0	0	-
Left knee	θ	-	0	0	0	0	-	0	-45	0	0	-
Left ankle	θ	-	0	0	0	1.43	-	0	-20	0	0	-
	ψ	-	6.88	6.88	6.88	6.88	-	-6.59	-6.59	-6.59	-1.59	-
Right hip	θ	-	0	18	0	0	-	0	0	0	0	-
	ψ	-	4.54	4.54	4.54	1.29	-	-6.35	-6.35	-6.35	-4.25	-
	ϕ	-	0	0	0	0	-	0	0	0	0	-
Right knee	θ	-	0	0	0	0	-	0	0	0	0	-
Right ankle	θ	-	0	0	0	0	-	0	0	0	1.43	-
	ψ	-	-6.59	-6.59	-6.59	-1.59	-	6.88	6.88	6.88	6.88	-

A.3 Parameter values defined for the implementation of the soft tasks controller #1 on the iCub

This section details the values defined for implementing the soft tasks controller #1 presented in section 2.3.2, for real-world experiments with the iCub. In particular, it defines the parameters used by the finite state machine, as well as the gains used by the feedback control policies.

Table A.8 Parameters for the stabilization of contact tasks **in real world experiments**: $f_{k_{z_{min}}}$ is the minimal vertical reaction force for which a contact is considered to exist, n_v is the number of vertices used in the approximation of the friction cone, μ_c and μ_t are approximations of the Coulomb and torsional static friction coefficients associated to the contact surfaces.

parameter	value
$f_{k_{z_{min}}}$	1 N
n_v	4
μ_c	$\frac{1}{3}$
μ_t	$\frac{1}{75}$

Table A.9 Task weights w used by the soft tasks controller #1 **in real-world experiments**

w_Υ	w_s	w_τ
100	0.5	1e-4

Table A.10 Parameters of the finite state machine used by the soft tasks controller #1 in real-world experiments. Parameters are introduced in A.1

parameter	value
e_{CoM}	0.013 m
e_{foot}	0.02 m
e_{leg}	24 deg
e_{legc}	24 deg
$f_{liftoff}$	60 N
$f_{touchdown}$	14 N
τ_{max}	70 N
$\dot{\tau}_{max}$	100 N/s
t_{min}	5 s
$t_{balancing}$	15 s
t_{smooth}	2 s

Table A.11 Proportional (P) feedback gains defined for Cartesian tasks **in real-world experiments**, along the x -, y - and z -axes for feedback on position, and about the same axes for feedback on rotation (denoted by $\theta_x, \theta_y, \theta_z$). For all P gains, the associated derivative (D) feedback gains are obtained as $D = 2\sqrt{P/40}$.

Task	Gain	State										
		1	2	3	4	5	6	7	8	9	10	11
CoM	P_x	50	50	50	50	50	70	50	50	50	50	70
	P_y	60	60	60	60	60	60	60	60	60	60	60
	P_z	50	50	50	50	50	50	50	50	50	50	50
left foot	P_x	0	0	0	0	0	0	0	90	90	50	30
	P_y	0	0	0	0	0	0	0	20	40	20	50
	P_z	20	0	0	0	0	0	0	70	20	50	30
	P_{θ_x}	5	5	5	5	5	5	5	20	10	10	10
	P_{θ_y}	5	5	5	5	5	5	5	20	10	10	10
	P_{θ_z}	5	5	5	5	5	5	5	20	10	10	10
right foot	P_x	0	0	90	90	50	30	0	0	0	0	0
	P_y	0	0	20	40	20	50	0	0	0	0	0
	P_z	0	0	70	20	50	30	0	0	0	0	0
	P_{θ_x}	5	5	20	10	10	10	5	5	5	5	5
	P_{θ_y}	5	5	20	10	10	10	5	5	5	5	5
	P_{θ_z}	5	5	20	10	10	10	5	5	5	5	5
root link	P_{θ_x}	5	5	10	10	10	5	5	10	10	10	5
	P_{θ_y}	5	5	10	10	10	5	5	10	10	10	5
	P_{θ_z}	5	5	10	10	10	5	5	10	10	10	5

Table A.12 Proportional (P) feedback gains defined for the postural task **in real-world experiments**. For all P gains, the associated derivative (D) feedback gains are all set to $D = 0$, the mechanics of each joint motor already providing damping. Gains for the shoulder and elbow have the same values for both right and left arms. Joints are denoted according to the pitch-roll-yaw convention, where θ is pitch, ψ is roll, and ϕ is yaw.

Joint		State										
		1	2	3	4	5	6	7	8	9	10	11
Torso	P_θ	40	40	40	40	40	40	40	40	40	40	40
	P_ψ	40	40	40	40	40	40	40	40	40	40	40
	P_ϕ	40	40	40	40	40	40	40	40	40	40	40
Shoulder	P_θ	45	45	45	45	45	45	45	45	45	45	45
	P_ψ	45	45	45	45	45	45	45	45	45	45	45
	P_ϕ	45	45	45	45	45	45	45	45	45	45	45
Elbow	P_θ	45	45	45	45	45	45	45	45	45	45	45
Left hip	P_θ	35	32	32	400	375	295	35	300	0	300	35
	P_ψ	35	35	35	35	35	35	35	35	35	35	35
	P_ϕ	25	25	25	25	25	25	25	25	25	25	25
Left knee	P_θ	50	50	50	100	50	50	50	400	600	600	50
Left ankle	P_θ	50	100	100	50	50	50	50	0	0	50	50
	P_ψ	75	100	100	75	75	75	75	0	0	75	75
Right hip	P_θ	35	35	300	0	300	35	35	35	300	300	50
	P_ψ	35	35	35	35	35	35	100	100	35	35	35
	P_ϕ	25	25	25	25	25	25	25	25	25	25	25
Right knee	P_θ	50	50	200	400	200	200	50	50	50	50	50
Right ankle	P_θ	50	50	0	0	50	50	100	100	50	50	50
	P_ψ	75	75	0	0	75	75	100	100	75	75	75

Table A.13 Displacement of the Cartesian tasks **for real-world experiments**, defined to achieve a stepping motion, along the x -, y - and z -axes.

δ	State										
	1	2	3	4	5	6	7	8	9	10	11
δ_{CoM_x}	-	0.01	0.01	0.01	0.01	-	0.01	0.01	0.01	0.01	-
δ_{CoM_y}	-	0	0	0	-0.02	-	0	0	0	0.02	-
δ_{CoM_z}	-	-0.005	-0.005	-0.005	-0.01	-	-0.005	-0.005	-0.005	-0.01	-
δ_{left_x}	-	-	-	-	-	-	0	0	0	0	-
δ_{left_y}	-	-	-	-	-	-	0	0.13	0.13	0.13	-
δ_{left_z}	-	-	-	-	-	-	0	0.05	0	0	-
δ_{right_x}	-	0	0	0	0	-	-	-	-	-	-
δ_{right_y}	-	0	-0.13	-0.13	-0.13	-	-	-	-	-	-
δ_{right_z}	-	0	0.05	0	0	-	-	-	-	-	-

Table A.14 Joint position values defined for real-world experiments in degrees according to the pitch-roll-yaw convention: θ is pitch, ψ is roll, and ϕ is yaw, and used for the postural task

Joint	State											
	1	2	3	4	5	6	7	8	9	10	11	
Torso	θ	-	-2.01	4.93	-2.01	-4.93	-	2.01	-4.93	2.01	4.93	-
	ψ	-	4.47	1.49	4.47	1.49	-	-4.47	-1.49	-4.47	-1.49	-
	ϕ	-	2.46	0.86	2.46	0.86	-	2.46	0.86	2.46	0.86	-
Left shoulder	θ	-	-8.55	7.18	-8.55	7.18	-	-8.55	3.23	-8.55	3.23	-
	ψ	-	49.16	46.61	49.16	46.61	-	49.16	38.90	49.16	38.90	-
	ϕ	-	13.96	17.48	13.96	17.48	-	13.96	19.14	13.96	19.14	-
Left elbow	θ	-	49.90	45.42	49.90	45.42	-	49.90	35.60	49.90	35.60	-
Right shoulder	θ	-	-8.55	3.23	-8.55	3.26	-	-8.55	7.18	-8.55	7.18	-
	ψ	-	49.16	38.90	49.16	38.90	-	49.16	46.61	49.16	46.61	-
	ϕ	-	13.96	19.14	13.96	19.14	-	13.96	17.48	13.96	17.48	-
Right elbow	θ	-	49.90	35.60	49.90	35.60	-	49.90	45.42	49.90	45.42	-
Left hip	θ	-	0	0	0	0	-	0	18	0	0	-
	ψ	-	-6.35	-6.35	-6.35	-4.25	-	4.54	4.54	4.54	1.29	-
	ϕ	-	0	0	0	0	-	0	0	0	0	-
Left knee	θ	-	0	0	0	0	-	-5.73	-45	-5.73	-5.73	-
Left ankle	θ	-	0	0	0	1.43	-	0	-20	0	0	-
	ψ	-	10.40	10.40	10.40	10.40	-	0	0	0	0	-
Right hip	θ	-	0	18	0	0	-	0	0	0	0	-
	ψ	-	4.54	4.54	4.54	1.2892	-	-6.35	-6.35	-6.35	-4.25	-
	ϕ	-	0	0	0	0	-	0	0	0	0	-
Right knee	θ	-	-10.40	-45	-10.40	-10.40	-	0	0	0	0	-
Right ankle	θ	-	0	-20.00	0	0	-	0	0	0	1.43	-
	ψ	-	-6.59	-6.59	-6.59	-1.59	-	10.40	10.40	10.40	10.40	-

Appendix B

Additional material for the soft tasks controller #2

This appendix provides additional details concerning the soft tasks controller #2 presented in section 2.3.3. In particular, it details the finite state machine in section B.1, before stating the parameters used by the finite state machine, as well as the gains used by the feedback control policies, for simulation experiments, in section B.2.

B.1 Finite state machine for the soft tasks controller #2

A finite state machine is used by the control framework to output desired setpoints for Cartesian and postural tasks, in function of the state of the robot. In this case, the state machine is applied to the whole-body motion of stepping in place, and is divided into 5 states for this purpose, as described in the following paragraphs.

1. **Initial balancing on two feet** is the initial state in which the controller begins. It assumes that the origin of the world frame coincides with the base frame. It also assumes the robot to be standing on two feet, and signals that both feet are in contact with the ground. It is used to register the initial CoM position, feet pose, neck orientation and joint positions, and it sets their desired values to the measured initial values. After a user-defined delay t_{min} , the state machine moves on to state 2.

2. **Move CoM above the stance foot** is used to set the desired position of the CoM above the stance foot (i.e. moving the CoM laterally along the y -axis to be aligned with the current foot frame y -axis, keeping the initial height and position along the x -axis). An additional displacement (or correction) of the CoM δ_{CoM} can be defined by the user, along the x - and y -axes. The state machine moves on to state 3 when the error on the CoM position is smaller than a user-defined threshold $e_{CoM_{max}}$, once more than a given time delay t_{min} has elapsed, or when a larger given time delay t_{max} has been exceeded.
3. **Stance foot balancing** is used to lift the swing foot above the ground. It signals that the swing foot is not in contact with the ground anymore, and sets the desired pose of the swing foot to a user-defined displacement $\delta_{swing_{foot}}$ with respect to its initial pose. At the same time, the CoM desired position is kept above the stance foot, with an additional displacement δ_{CoM} (or correction) of the CoM that can be defined by the user. The state machine moves on to state 4 when the error on feet position is smaller than a threshold $e_{feet_{max}}$, or when a given time delay t_{max} has been exceeded.
4. **Prepare for foot touchdown** is used to move the foot back towards the ground. It sets the desired pose of the swing foot to its initial pose, while the CoM is again kept above the stance foot, with an additional displacement (or correction) δ_{CoM} of the CoM that can be defined by the user. The state machine moves on to state 5 when the error on feet position is smaller than a threshold $e_{feet_{max}}$, once more than a given time delay t_{min} has elapsed, or when a larger given time delay t_{max} has been exceeded.
5. **Two feet balancing** is used to bring the CoM back to its initial position. It also signals that both feet are in contact with the ground, once the vertical force measured at the swing foot is above a user-defined threshold $F_{z_{min}}$. The state machine swaps the swing and stance feet, and moves on to state 2 when the error on CoM position is smaller than a threshold $e_{CoM_{max}}$, once more than a given time delay t_{min} has elapsed, or when a larger given time delay t_{max} has been exceeded.

The neck orientation and joint positions are left untouched by states, allowing their desired values to remain equal to their initial values. Furthermore, at each time instant, the desired values output from the state machine are sent to a minimum

jerk trajectory generator [Pattacini et al., 2010], that provides instantaneous desired positions, velocities and accelerations to the controller, for smooth trajectories, given a user-defined smoothing time t_{smooth} .

According to the current state, feedback gains used in equations (2.11) and (2.13) may be updated to values defined by the user for each state. In this specific application, however, the gain values are defined to be the same across all states.

Furthermore, remark that in this controller, the purpose of the postural task is solely to stabilize redundant degrees of freedom, and desired joint positions are simply defined as the initial posture of the robot. In order to encourage that the reference postural task accelerations input to the controller are consistent with reference Cartesian task accelerations, we attempt taking into account feedback terms on Cartesian tasks $\dot{\nu}_{\mathcal{T}}^*$ into the computation of the postural feedback term \ddot{s}^* , which was previously computed as in equation (2.13).

The idea of the modification we propose here is to compute joint accelerations that allow a tradeoff between the achievement of the desired Cartesian accelerations, and remaining close to the desired postural accelerations computed with a PD feedback control policy.

For doing so, $\dot{\nu}_{\mathcal{T}}$, the robot acceleration that Cartesian task accelerations $\dot{\nu}_{\mathcal{T}}^*$ may induce, is computed using the following expression. The use of the Jacobian pseudo-inverse is considered acceptable in the context where we are only concerned with stabilizing redundant DOFs, and we do not need precision.

$$\dot{\nu}_{\mathcal{T}} = J_{\mathcal{T}}^{\dagger} \left(\dot{J}_{\mathcal{T}} \nu - \dot{\nu}_{\mathcal{T}}^* \right) \quad (\text{B.1})$$

As for $\dot{\nu}_s$, the robot acceleration that postural task accelerations may induce, it is computed using the following expression, where a feedback term on the base velocity $\dot{\nu}_{\mathcal{B}}^*$ is used to ensure convergence of the base acceleration due to postural accelerations to zero.

$$\dot{\nu}_s = \left[\dot{\nu}_{\mathcal{B}}^{*\top} \quad \ddot{s}^{ref\top} \right]^{\top} \quad (\text{B.2})$$

From there, the following optimization problem is defined, to compute the desired postural acceleration, which will then be used by the controller.

$$\ddot{s}^* = \arg \min_{\ddot{s}} w_s \left| \tilde{\nu}_s(\ddot{s}) \right|^2 + \sum_{\mathcal{T}} w_{\mathcal{T}} \left| \tilde{\nu}_{\mathcal{T}}(\ddot{s}) \right|^2 \quad (\text{B.3})$$

In the above, the error on robot acceleration is computed with

$$\tilde{\nu}_{\mathcal{T}}(\ddot{s}) = \begin{bmatrix} \dot{v}_{\mathcal{B}} \\ \ddot{s} \end{bmatrix} - \dot{\nu}_{\mathcal{T}} \quad (\text{B.4})$$

for Cartesian tasks, and with

$$\tilde{\nu}_s(\ddot{s}) = \begin{bmatrix} \dot{v}_{\mathcal{B}} \\ \ddot{s} \end{bmatrix} - \dot{\nu}_s \quad (\text{B.5})$$

for the postural task.

B.2 Parameter values defined for the implementation of the soft tasks controller #2

This section details the values defined for implementing the soft tasks controller #2 presented in section 2.3.3, for simulation experiments.

Table B.1 Parameters for the stabilization of contact tasks: $f_{k_{z_{min}}}$ is the minimal vertical reaction force for which a contact is considered to exist, n_v is the number of vertices used in the approximation of the friction cone, μ_c and μ_t are approximations of the Coulomb and torsional static friction coefficients associated to the contact surfaces.

parameter	value
$f_{k_{z_{min}}}$	10 N
n_v	4
μ_c	1
μ_t	$\frac{1}{75}$

Table B.2 Parameters of the finite state machine used by the soft tasks controller #2. Parameters are introduced in B.1, and displacements are given along the axes [x, y, z].

parameter	value
δ_{CoM}	[0, 0, 0] m
$\delta_{swing_{foot}}$	[-0.025, 0, 0.025] m
$e_{CoM_{max}}$	0.025 m
$e_{feet_{max}}$	0.025 m
t_{min}	2 s
t_{max}	6 s
$F_{z_{min}}$	0.1 N
t_{smooth}	2 s

Table B.3 Task weights w used with the soft tasks controller #2

w_{CoM}	w_{stance}	w_{swing}	w_{neck}	w_s	w_τ
1	1	1	1	1e-3	1e-4

Table B.4 Proportional (P) and derivative (D) feedback gains defined for Cartesian tasks of the soft tasks controller #2, along the x -, y - and z -axes for feedback on position, and about the same axes for feedback on rotation (denoted by $\theta_x, \theta_y, \theta_z$)

Task	Gain	x	y	z	θ_x	θ_y	θ_z
CoM	P	5	5	5	-	-	-
	D	$2\sqrt{5}$	$2\sqrt{5}$	$2\sqrt{5}$	-	-	-
swing foot	P	10	10	10	6	6	6
	D	$2\sqrt{10}$	$2\sqrt{10}$	$2\sqrt{10}$	$2\sqrt{6}$	$2\sqrt{6}$	$2\sqrt{6}$
stance foot	P	10	10	10	6	6	6
	D	$2\sqrt{10}$	$2\sqrt{10}$	$2\sqrt{10}$	$2\sqrt{6}$	$2\sqrt{6}$	$2\sqrt{6}$
neck	P	-	-	-	1.5	1.5	1.5
	D	-	-	-	$2\sqrt{1.5}$	$2\sqrt{1.5}$	$2\sqrt{1.5}$

Table B.5 Proportional (P) and derivative (D) feedback gains defined for the postural task of the soft tasks controller #2

	Torso			Shoulder			Elbow	Hip			Knee	Ankle	
	pitch	roll	yaw	pitch	roll	yaw	pitch	pitch	roll	yaw	pitch	pitch	roll
P	20	20	20	10	10	10	8	30	30	30	60	10	10
D	$2\sqrt{20}$	$2\sqrt{20}$	$2\sqrt{20}$	$2\sqrt{10}$	$2\sqrt{10}$	$2\sqrt{10}$	$2\sqrt{8}$	$2\sqrt{30}$	$2\sqrt{30}$	$2\sqrt{30}$	$2\sqrt{60}$	$2\sqrt{10}$	$2\sqrt{10}$

Appendix C

Additional material for the parametrization-based joint limit avoidance approach of chapter 3

C.1 Proof of lemma 3

This section presents the proof of lemma 3, introduced in section 3.4.1 for joint limit avoidance using the exogenous parameter ξ . As a recall, the lemma states that substituting ξ in a passivity-based control law allows for joint limit avoidance, as well as the asymptotic stability of the closed-loop dynamics equilibrium point.

Consider the following candidate Lyapunov function:

$$V := \frac{1}{2} \dot{\xi}^T M_\xi \dot{\xi} + \frac{1}{2} \tilde{\xi}^T K_P \tilde{\xi} \quad (\text{C.1})$$

Observe that

$$V = 0 \iff (\dot{\xi}, \tilde{\xi}) = (0_n, 0_n) \quad (\text{C.2})$$

Note that K_P being a positive definite matrix, and in view of Property 3, then

$$V(\tilde{\xi}, \dot{\xi}, t) > 0 \quad \forall (\tilde{\xi}, \dot{\xi}) \neq \{0\} \quad (\text{C.3})$$

Now, recall that M_ε tends to zero when $\tilde{\xi}$ tends to infinity. Despite this fact, one shows that the candidate Lyapunov function is radially unbounded, i.e.

$$|(\tilde{\xi}, \dot{\tilde{\xi}})| \rightarrow \infty \Rightarrow V \rightarrow \infty \quad (\text{C.4})$$

which is a sufficient condition for obtaining global stability results associated with a candidate Lyapunov function [Khalil, 2002, p. 152]. This is the main point of the proof, where it differs consistently from the proof of the passivity-based controller (3.3).

Then, in view of Property 4, the time derivative of V along the closed loop system (3.17)-(3.18) is given by

$$\dot{V} = -\dot{\tilde{\xi}}^T K_D \dot{\tilde{\xi}} \leq 0 \quad (\text{C.5})$$

which implies the stability of the equilibrium point

$$(\tilde{\xi}, \dot{\tilde{\xi}}) = (0, 0) \quad (\text{C.6})$$

and boundedness of the system trajectories

$$(\tilde{\xi}, \dot{\tilde{\xi}})(t) \quad (\text{C.7})$$

for any initial condition.

Now, observe that the closed-loop system (3.17)-(3.18) is time varying, and this implies that LaSalle's lemma cannot be applied to determine that \dot{V} tends to zero. To show this, we have to apply Barbalat's lemma, and thus we have to show that \ddot{V} is bounded. By using the fact that the trajectories of the system $(\tilde{\xi}, \dot{\tilde{\xi}})(t)$ are bounded, one shows that \ddot{V} is bounded. Then, \dot{V} tends to zero, and this implies that $\dot{\tilde{\xi}}$ tends to zero. To show that also $\tilde{\xi}$ tends to zero, we have to show first that $\ddot{\tilde{\xi}}$ tends to zero. This latter fact can be shown by using again Barbalat's lemma, i.e. one shows that $\ddot{\tilde{\xi}}$ is bounded using the fact that the system trajectories are bounded. Then, one has $\dot{\tilde{\xi}} \rightarrow 0$ and $\ddot{\tilde{\xi}} \rightarrow 0$. By using these facts in the closed loop dynamics (3.17)-(3.18), one has that $\tilde{\xi}$ tends to zero.

Appendix D

Additional material related to the survey on tuning of QP-based controllers

This appendix provides additional material with respect to the survey on parameter tuning of QP-based controllers, presented in chapter 4. Section D.1 contains a copy of the email sent to mailing lists [euRobotics, 2018; GdR-Robotique, 2018; robotics worldwide, 2018] in order to distribute our survey. Then, section D.2 reproduces the form of the survey in its entirety. Finally, section D.3 reports the entirety of the free text comments provided by the respondents of the survey, in answers to open-ended questions.

D.1 Invitation to participate to the survey on QP-based controllers

Dear robotics community members,

We are working on automatic tuning of QP controllers for humanoid robots, and we need your help!

If you have at any point worked with controllers based on quadratic programming (QP), we would like to invite you to participate in an online survey on this subject:

<https://goo.gl/forms/M4LC64l14fIKw0503>

Please fill it out to help us know about your experience with QP-based controllers, and if a tool for their tuning would be useful to the community.

The survey is anonymous, and should take approximately 10 minutes of your time.

Many thanks in advance for your help!

Marie Charbonneau

Invited PhD student

Team Larsen

INRIA Nancy Grand-Est

France

email:marie.charbonneau@loria.fr

D.2 Questionnaire of the survey on tuning of QP-based controllers

Tuning QP controllers for robots

Thank you for participating to this anonymous survey!

We are leading this investigation to know if a possible tool for the tuning of controllers based on quadratic programming (designated "QP controllers" below) would be useful to the community. What is your experience on this topic?

Please help us out!

The survey is organized into five sections:

- 1) introduction (current section)
- 2) what kind of robot you work with
- 3) in which way you use QP controllers
- 4) your experience with the tuning of QP controllers
- 5) info about you (don't worry: the survey is anonymous, it's just for stats)

There will be space to tell us freely about your experience with QP controllers. Rest assured that the survey is anonymous, so have your say!

If you have any problem filling the questionnaire, please write to: marie.charbonneau@loria.fr

Ready? Go!

* Required

Are you using or have you been using QP controllers? *

- Yes
- No

NEXT

Page 1 of 5

Figure D.1 Survey page 1

Tuning QP controllers for robots

* Required

Defining your robot

Which type of robot are you developing your QP controller with?

*

- Manipulator
- Wheeled mobile robot
- Humanoid robot
- Legged robot (non humanoid)
- Aerial robot
- Underwater robot
- Other: _____

BACK

NEXT

Page 2 of 5

Figure D.2 Survey page 2

Tuning QP controllers for robots

* Required

Defining the QP problem

In which way are you using QP controllers? *

- Inverse dynamics
- Inverse kinematics
- Offline trajectory generation / motion planning
- Real-time trajectory generation / motion planning
- Position control
- Torque/force control
- Other: _____

When multiple tasks are considered, various ways can be used to define a QP. What is your approach in this sense? *

- We only optimize over a single task
- Stack of tasks: a cascade of QPs is solved, with the lowest priority task acting in the nullspace of the highest priority tasks
- High priority tasks are set as constraints of the QP, while low priority tasks are considered into the cost function
- Soft/weighted tasks: the cost function is computed from a weighted sum of values associated to each task
- Other: _____

If possible, you may indicate a link to a reference regarding the QP controller(s) you work(ed) with, or to the code

Your answer

BACK

NEXT

Page 3 of 5

Figure D.3 Survey page 3

Tuning QP controllers for robots

* Required

Tuning of QP

In your experience, how important is it to tune the following parameters associated to a QP controller? *

	1 (no important impact on results)	2	3	4	5	6	7 (crucial for successful results)
gains of the controller (e.g. PD gains of a given task)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
desired task trajectories	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
task weights/priorities	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
constraints of the QP	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
intrinsic parameters of the QP (e.g. length of MPC horizon, thresholds)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
external parameters associated to the tasks (e.g. contact properties)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Is there another parameter which you think is important to tune, but which we forgot in the question above? Please provide a description

Your answer

Do you have comments to add with respect to the penultimate question?

Your answer

Figure D.4 Survey page 4, part 1

From your experience, and please be honest: how much time and effort do you generally spend tuning the following parameters for a new QP controller? *

	We do not tune those	< 1 day	< 1 week	< 1 month	< 6 months	It can take longer than 6 months to get the right parameter	Tuning is a continuous process which always needs to be (re)done
gains of the controller (e.g. PD gains of a given task)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
desired task trajectories	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
task weights/priorities	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
constraints of the QP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
intrinsic parameters of the QP (e.g. length of MPC horizon)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
external parameters associated to the tasks (e.g. contact properties)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Is there another parameter which you spend time and effort tuning, but which we forgot in the question above? Please provide a description

Your answer

Do you have comments to add with respect to the penultimate question?

Your answer

Have you developed or are you using any tools or techniques for making the tuning process easier? *

Your answer

How tedious do you find parameter tuning to be?

	1	2	3	4	5	6	7	
It is so not tedious, I find it rather enjoyable to tune parameters for QP controllers	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	It is tedious enough to make me dislike tuning parameters

Figure D.5 Survey page 4, part 2

For what reason?

Your answer

How important do you think it is to make parameter tuning less tedious? *

1 2 3 4 5 6 7

Not important Crucial

For what reason?

Your answer

How would you rate your interest in an eventual tool for the tuning for your QP controller? *

1 2 3 4 5 6 7

I am not interested in using tools for that I am very excited to try it out!

What would be the ideal tool in your opinion?

Your answer

Is there information based on your experience that you would like to share, in order to help us better understand your issues related to the tuning of QP controllers (if you have any)?

Your answer

Would you like to share further information, comments, recommendations or feedback related to this topic?

Your answer

BACK

NEXT

Page 4 of 5

Figure D.6 Survey page 4, part 3

About you

Your experience in robotics *

< 1 year

1-3 years

4-7 years

8-10 years

> 10 years

In which field are you working, more specifically? *

Control

Estimation

Identification

Machine learning

Mechanical design

Modelling

Planning

Software development

Other: _____

Your gender *

Female

Male

Other: _____

Are you a robot?

Yes

No

Maybe

Thank you for taking the time to fill this form!


 Page 5 of 5

Figure D.7 Survey page 5

D.3 Answers to open-ended questions

Answers to “If possible, you may indicate a link to a reference regarding the QP controller(s) you work(ed) with, or to the code”

<https://github.com/robotology/whole-body-controllers>

<https://orca-controller.readthedocs.io/en/dev/>

https://github.com/kuka-isir/cart_opt_ctrl

Controller source code: <https://github.com/jrl-umi3218/Tasks>

<https://hal.archives-ouvertes.fr/hal-01276931/document>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7433460>

Home-made

<https://projects.coin-or.org/qpOASES>

<https://ieeexplore.ieee.org/document/6482266/>

https://github.com/semroco/giskard_core and github.com/semroco/giskardpy

<http://www.roboticsproceedings.org/rss14/p54.pdf>

The QP used on HRP4 at LIRMM montpellier and

<https://hal.archives-ouvertes.fr/hal-01735462v1>

OpenSoT and MC-RTC

https://github.com/robotology/walking-controllers/blob/master/modules/Walking_module/src/WalkingQPInverseKinematics_qpOASES.cpp

<https://github.com/jrl-umi3218/Tasks/>

Contrôleur développé par Joseph Salini (ISIR) pour XDE

Answers to “Is there another parameter which you think is important to tune, but which we forgot in the question above? Please provide a description”

Regularization parameters (e.g. tolerances, saturations...)

Regularization terms

aleatory

Sampling rate of the trajectory

Answers to “Do you have comments to add with respect to the penultimate question?”, referring to “In your experience, how important is it to tune the following parameters associated to a QP controller?”

Trajectories should not be tuned but should be the outcome!

You should specify 'don't know' answer to the questions above (I am not sure about all)

Answers to “Is there another parameter which you spend time and effort tuning, but which we forgot in the question above? Please provide a description”

What took the longest in terms of getting all of these things implemented was when you see some undesired behavior (like a foot bouncing off the ground on touch down) it can be difficult to figure out which part of the system to fix. It could be the foot trajectory, the trajectory following due to low gains, the trajectory following due to a bad damping estimate that gets fed in through inverse dynamics, maybe the objective weighting is too low, maybe we are switching from swing to stance too early/late. While this framework is incredibly flexible and useful it does inherently couple everything together so it can be difficult to tease things apart.

Answers to “Do you have comments to add with respect to the penultimate question?”, referring to “From your experience, and please be honest: how much time and effort do you generally spend tuning the following parameters for a new QP controller?”

I assumed a person is working on the robot every day, full time.

The crucial parts are to get the constraints right. And that changes with every task. However, if done smartly, they can be (partly) reused. Some parameters follow iterations (eg contact properties), and they can be short (weeks or months for modifying a manipulator’s end effector) or long (up to a year or longer for building a new robot). Many parameters are also dependent, so changing e.g. the compliance of the actuator triggers changes in other parameters. Therefore, most are continuously evolving and good sets vary from task to task.

Tuning time can vary extremely depending of the scenario. It can be really fast on some mono robot manipulation scenario but can be really tedious for multi robot scenario.

I don’t tune the parameters, Phd students do, so do not consider my answers.

I don’t consider "desired task trajectories" as part of QP tuning, but rather as part of motion planning/MPC.

Answers to “Have you developed or are you using any tools or techniques for making the tuning process easier?”

No

no

No.

Not at the moment. We are evaluating everything directly on the hardware since it is in front of us anyway.

magic

Yes see my thesis

Matlab Hybrid MPC toolbox

We developed tools on top of RVIZ to graphically tune controller gain and tasks trajectory.

Basic zero-pole graphs or simulated time plots while tuning the low-level (PID) gains.

We are learning a mix of the controller -so best case, its parameters require no tuning- and the task jacobian which takes care of some of the other parameters.

Yes

Not yet

NA

No. We are just getting around to defining performance metrics such that you can quantify improvement while tuning... but currently it is all based on trying it on the robot and see how it looks.

No tuning, just Mismatch Learning :)

Developed by the team.

Answers to “For what reason?”, referring to “How tedious do you find parameter tuning to be?”

A complex dynamical system may have too many parameters to tune, the possible combinations are endless and the combined effect of all parameters is not easy to predict.

Parameters should have a physical meaning and should therefore be easy to tune. If that is not the case, your model is bullshit. Of course, it is not always easy to find the actual numbers for the physical parameters, but at least you can apply common sense to their ranges.

Application specific and mostly heuristic

QP controller can be really tedious to tune if you use it with bad trajectory planning and try to play with tasks weights and gains to overcome this issue.

Sometimes work for the configuration of the robot used during tuning, then fails for a different robot configuration

Once you work with it a while it certainly gets easier.

Trial and error tuning is too long and often tuning one parameter will affect other tuned parameters

I usually find that tuning only the scale-of-order is enough to get a first set of working QP settings (this applies to both PD gain and weight tunings).

Temps consommé imprévisible et toujours » temps de développement du code

Answers to “For what reason?”, referring to “How important do you think it is to make parameter tuning less tedious?”

It is tedious because it often takes a huge amount of time. If less tedious = less time, then I think it is really important.

See answer above.

Save time and make QP's useful for production systems

No one wants to tune parameters every time with a new platform.

Easier implementation on industrial robots

In our work the QP part was the most robust. Compared to the contact planning part it was really easy.

It's never tuned correctly and/or for all applications

I think the main problem with parameter tuning is not that it might be tedious. The main problem is that one never knows when the current tuning is good enough.

I see potential in there.

Answers to “What would be the ideal tool in your opinion?”

It should take as input parameters that have a clear and predictable effect on the robot, then map these parameters to the ones that people usually tune, whose effect is neither clear or predictable

Easy to use, good GUI.

Automatically move the robot and find the best PID values

The ideal tool would not need to any major adaptation on the part of the controller itself.

It should rely on a dynamic simulator and on some machine learning (better if supervised learning, even if automatically supervised). The simulator replays the experiment systematically and tunes the gain. Some manual tuning will also be required on the real robot, but only at the end.

An easy to set-up tool that can be easily applied to different QP frameworks, and auto-tune the parameters quickly+safely

Hmm, let's see. The tool needs a way to guess what I'm looking for in terms of whole-body behavior. At first, I see two ways to do that:

1 — The tool assumes the instantaneous QP cost function is the specification. It then goes on to play a motion following my desired tasks, and looks at the integral of this cost function over a given run. Then changes QP parameters, and tries again for a different run. This way, we know how to generate a dataset (parameters, costs); then we can follow a data-driven approach (a.k.a. "learning")! Expected pro: user can do something else while the computer is working. Expected con: computation time may be prohibitive; only applies to PD-gain tuning of a weighted QP.

2 — The tool does not assume a specification; rather, it generates a trajectory and asks me every time which one I find better. Expected pro: applies to both PD-gain and weight tunings; specification is implicit. Expected con: user spends brain time in the process, dataset will thus be smaller.

These thoughts being sketched, the ideal tool in my opinion looks like this: the user specifies all of its cost terms (for a weighted QP: the expressions that are weighted; for a lexicographic QP: same across all layers), but not weights nor PD

gains. The tool will generate several trajectories for, say, at most a week. As a final outcome, the user is presented with a cost-function design GUI:

- Input: cost function, i.e. weights on each cost terms and/or lexicographic separation between weighted layers
 - Output: recommended set of PD gains, and statistics over each cost term: min/max/average/standard deviation of the cost value over a trajectory.
- ... well, you asked for the ideal ;-)

Answers to “Is there information based on your experience that you would like to share, in order to help us better understand your issues related to the tuning of QP controllers (if you have any)?”

I think having more robustness w.r.t. a change of task is important. Usually, the hand-made tuning is really task-dependent.

Check out my thesis : <https://tel.archives-ouvertes.fr/tel-01685182/en>

A graphic "tutorial" on QP parameters could be useful for people getting started with QP controllers

PD-gain tuning is not so hard. The problem is, the user does not really know what she/he wants. Hence my hunch that the point is not helping users tune their QPs, but helping QPs tune their users!

Answers to “Would you like to share further information, comments, recommendations or feedback related to this topic?”

Refer to works on machine learning for transferring from simulation to real robot.

Good luck!

Answers to “Are you a robot?”

