Dynamic control for the task/posture coordination of humanoids : toward synthesis of complex activities

Joseph Salini

THÈSE de DOCTORAT

de

l'Université Pierre & Marie Curie

École Doctorale de Sciences Mécanique, Acoustique, Électronique et Robotique de Paris

Spécialité

Mécanique - Robotique

présentée par

Joseph SALINI

Commande dynamique pour la coordination tâche/posture des humanoïdes : vers la synthèse d'activités complexes

---

Soutenue le 15 Juin 2012

JURY

M.	P. Fraisse	Professeur à l'Université Montpellier 2	Rapporteur
M.	J.P. Laumond	Directeur de recherche au LAAS-CNRS	Rapporteur
M.	A. Micaelli	Directeur de recherche au CEA/LIST, Fontenay-aux-Roses	Examinateur
M.	O. Khatib	Professeur à l'Université de Stanford	Examinateur
M.	P.B. Wieber	Chargé de recherche à l'INRIA Rhône-Alpes	Examinateur
M.	P. Bidaud	Professeur à l'Université Pierre & Marie Curie	Directeur
M.	V. Padois	Maître de Conférences à l'Université Pierre & Marie Curie	Examinateur
M.	O. Sigaud	Professeur à l'Université Pierre & Marie Curie	Membre invité

A dissertation submitted for the degree of

Doctor of Philosophy

of the

Pierre and Marie Curie University

in

Mechanics and Robotics

presented by

Joseph SALINI

Dynamic control for the task/posture coordination of humanoids : toward synthesis of complex activities

---

Defended on June 15^{th}*, 2012*

Committee in charge

M.	P. Fraisse	Professor at University of Montpellier 2	Referee
M.	J.P. Laumond	Research Director at LAAS-CNRS	Referee
M.	A. Micaelli	Research Director at CEA/LIST, Fontenay-aux-Roses	Examiner
M.	O. Khatib	Professor at Stanford University	Examiner
M.	P.B. Wieber	Research scientist at the INRIA Rhône-Alpes	Examiner
M.	P. Bidaud	Professor at the Pierre & Marie Curie University	Adviser
M.	V. Padois	Associate professor at the Pierre & Marie Curie University	Examiner
M.	O. Sigaud	Professor at the Pierre & Marie Curie University	Guest member

Résumé

Les travaux de recherche développés dans le cadre de cette thèse traitent d'une manière générale du problème de la commande dynamique « orientée tâche » de systèmes sous-actionnés et redondants en considérant plus spécifiquement les systèmes robotiques humanoïdes ou encore les humains virtuels.

Nous avons cherché à apporter des contributions au problème de la « synthèse par la commande d'activités motrices » pour des systèmes contraints par leurs capacités intrinsèques et par leurs interactions notamment physiques avec l'environnement. Les problèmes plus spécifiques qui ont été traités sont relatifs à : 1) la commande dynamique des systèmes humanoïdes pour la réalisation d'activités motrices nécessitant la coordination tâche/posture perturbées par des interactions physiques, 2) l'enchaînement dynamique continu des activités d'un répertoire de coordinations motrices, 3) la planification et l'adaptation des activités élémentaires dans l'objectif d'un enchaînement automatique supervisé pour la réalisation de tâches complexes dans des contextes non déterministes.

Le premier chapitre de ce mémoire de thèse est consacré à la description d'un moteur de simulation physique et de lois de commande pour le développement de comportements dynamiques élémentaires de systèmes mécaniques poly-articulés. Ces derniers sont contraints par des liaisons binaires bilatérales ainsi que des liaisons unilatérales comme des contacts frictionnels. Nous développons dans un premier temps le formalisme supportant le développement de ce simulateur avant de décrire son implémentation et son fonctionnement à travers des exemples simples. Les équations du mouvement sont ici développées en s'appuyant sur les équations de Boltzmann-Hamel de systèmes mécaniques à base flottante, et la résolution numérique des contraintes s'effectue au travers d'un algorithme itératif de type Gauss-Seidel. L'intégration numérique du système se fait à pas de temps constant selon le schéma d'intégration Euler semi-implicite, et l'ensemble des différentes étapes du processus est illustré par un exemple simple.

Le deuxième chapitre propose une commande générique fondée sur la réalisation de plusieurs tâches sous contraintes. Ces contraintes sont principalement la représentation des limitations physiques du robot, à la fois internes comme les butées articulaires, et externes comme les contacts frictionnels lors des interactions avec l'environnement. Le formalisme s'inspire des « fonctions de tâches », et la résolution simultanée de plusieurs d'entre-elles est réalisée par une commande optimale quadratique multicritère. Elle s'appuie sur des méthodes d'optimisation convexe, et les bonnes propriétés des programmes quadratiques (QP) permettent un contrôle efficace (en temps, convergence, robustesse) de l'ensemble du corps pour la coordination tâche/posture. Une stratégie fondée sur les pondérations est utilisée pour la gestion de tâches concurrentes, cela permettant une grande souplesse du contrôle lors de transitions dans les ensembles de tâches pour obtenir la continuité des variables de commande.

Le troisième chapitre développe une implémentation de ce contrôleur générique sur des activités types au travers du modèle virtuel du robot iCub (http://www.robotcub.org/). Ces expériences mettent en évidence la capacité de la commande à réaliser des tâches complexes incluant des transitions lors de séquences de tâches ou lors de changement d'état dans l'ensemble des contraintes lors de la rupture des contacts par exemple. De plus, les performances du programme d'optimisation d'un point de vue temporel sont étudiées pour différentes expériences et différents formalismes.

Le quatrième chapitre développe un contrôleur haut-niveau permettant de planifier et d'adapter une séquence pour la matérialisation d'activités plus complexes en se basant sur un répertoire d'actions prédéterminées et la supervision de leur performances, tout en garantissant la continuité de la commande. Pour cela, un moteur de décision fondé sur la logique floue est mis en place à l'aide du logiciel « Spirops ». Les informations perçues par les capteurs servent d'entrées au moteur décisionnel qui utilise un ensemble de règles floues pour avoir, en sortie, des valeurs d'intérêt associées à chaque action que le robot peut exécuter. Ce choix s'est imposé à nous en raison des bonnes propriétés de cette technique, à savoir 1) sa robustesse et son efficacité, 2) l'utilisation d'une représentation plus « symbolique » pour le contrôle de systèmes robotiques, et 3) l'emploi de variables continues pour la modélisation d'événements discrets, et donc une interconnexion très simple avec le contrôleur générique bas-niveau. Ce schéma est validé par plusieurs expériences où le robot iCub génère des comportements relativement complexes en vérifiant l'état de ses actions et en s'adaptant aux conditions géométriques et dynamiques de l'environnement.

Mots Clés : *Simulateur Dynamique, Robot humanoïde, Contrôle Global du Mouvement, Programme Linéaire Quadratique, Transitions, Moteur de Décision.*

Abstract

The research developed in this PhD thesis deals with the general problem of dynamic control of "task-oriented" under-actuated and redundant systems considering more specifically the humanoid robotic systems or virtual humans.

This research brings contributions to the problem of motor activities synthesis for constrained systems by their intrinsic capacities and by particular physical interactions with the environment.

More specific issues that were addressed relate to : 1) the dynamic control of humanoid systems for carrying out basic activities requiring task/posture coordination perturbed by physical interactions, 2) the building of sequences of continuous dynamic activities based on a repertoire of motor coordination, 3) the planning and the adaptation of activities in the aim of a supervised automatic sequencing for complex non-deterministic tasks.

The first chapter of this thesis is devoted to the description of a physical simulation engine and control laws for the development of dynamic behaviors of elementary poly-articulated mechanical systems. These latter are constrained by binary relations and bilateral links as unilateral frictional contacts. We develop the formalism supporting the development of this simulator before describing its implementation and operation through simple examples. The equations of motion are developed here based on the Boltzmann-Hamel equations of mechanical systems with a free-floating base, and the numerical solution of the constraints is done through an iterative algorithm of Gauss-Seidel. The numerical integration of the system is in constant time step integration scheme according to the semi-implicit Euler method, and all the different steps of the process are illustrated by a simple example.

The second chapter provides a generic command based on the completion of several tasks under constraints. These constraints are mainly the representation of physical limitations of the robot, both internal, such as joint limits, and as external frictional contacts during interactions with the environment. The formalism is based on "task functions", the simultaneous resolution of several of them is realized by a quadratic multicriteria optimal control scheme. It is based on convex optimization methods, and the good properties of quadratic programming (QP) can efficiently perform the whole body control (in time, convergence, robustness) for the task/ posture coordination. A strategy based on weights is used to manage conflicting tasks. It allows a great flexibility of control during transitions in the sets of tasks and constraints for the continuity of the control variables.

The third chapter develops an implementation of this controller on generic activities through the virtual model of the iCub robot (http://www.robotcub.org/). These experiments verify the achievement of multi-task problems and performance of the controller for managing transitions in task sequences or when state change in the set of constraints, when a contact is broken, for example. In addition, program performance optimization of a temporal point of view are studied for different experiences and different formalisms.

The fourth chapter develops a high-level controller to plan and adapt a sequence for the realization of more complex activities based on a predetermined repertoire of actions and monitoring their performance, while ensuring the control variables continuity. For this, a decision engine based on fuzzy logic is implemented via "SpirOps" software. The information collected by sensors are used as inputs to the decision making engine that uses a set of fuzzy rules for the output values of interest associated with each action that the robot can execute. This choice was forced on us for the good properties of this technique, namely 1) its robustness and efficiency, 2) the use of a more symbolic for control of robotic systems, and 3) the use of continuous variables for modeling discrete event, and therefore a very simple interconnection with the generic low-level controller. This scheme is validated by several experiments where the robot iCub generates relatively complex behaviors by checking the status of its actions and adapting to the geometric and dynamic conditions of the environment.

Keywords : *Dynamic Simulator, Humanoid Robot, Whole-Body Dynamic Control, Linear Quadratic Programming, Transitions, Decision Making Engine.*

À ma famille.

Ce que le monde nous apporte
Outre les bienfaits d'une nature comblée
Usinons le pour forger la grande porte
La porte qui mène à la fraternité

Eux les hommes que nous appelons nos frères
Usant de leurs dons pour servir l'humanité
Recevront de Dieu le cadeau le plus sacré
Celui de vivre heureux aux cotés de leurs pairs

Oublions les querelles du passé
Retrouvons le bonheur du présent
Si par l'idée ou le sang nous sommes liés à nos ainés
Ensemble nous resterons, et ce jusqu'à la fin des temps

"Ce qui est simple est faux. Ce qui ne l'est pas est inutilisable."
Paul Valéry

Contents

List of Figures

List of Symbols

Scalar are lower-case letters. Matrix are noted in upper-case letters. Vector are bold lower-case letters. The sign $<$ compares two scalars whereas $\prec$ compares two vectors term by term.

$\mathbb{R}^n$	n-dimensional vector space over the field of the real numbers
I_n	identity matrix of $\mathbb{R}^{n\times n}$
$\bullet^\mathsf{T}$	transpose
$\dot\bullet$ $\ddot\bullet$	time derivative and second time derivative
$\|\bullet\|$	Euclidean norm
Ψ	frame in space
$\bullet_{a,b}$	from Ψ_a to Ψ_b
$\bullet^a$	relative to Ψ_a
$SO(3)$	special orthogonal group of dimension 3
$SE(3)$	special Euclidean group of dimension 3
$\mathfrak{so}(3)$	Lie algebra of $SO(3)$
$\mathfrak{se}(3)$	Lie algebra of $SE(3)$
$\mathfrak{se}^*(3)$	dual vector space of $\mathfrak{se}(3)$
$\boldsymbol{p}$	point in space
$\boldsymbol{u}$	vector in space
R	rotation matrix
H	homogeneous matrix
$\widetilde{\bullet}$	homogeneous coordinate of considered vector
$\widehat{\bullet}$	skew-matrix according to considered vector
$\boldsymbol{v}$	linear velocity vector
$\boldsymbol{\omega}$	rotational velocity vector
$\boldsymbol{T}$	twist
$\boldsymbol{f}$	force vector
$\boldsymbol{\tau}$	torque vector or generalized force
$\boldsymbol{W}$	wrench
$\mathcal{W}$	mechanical work

Q	generalized coordinates matrix for **nonlinear** configuration space joint
$\boldsymbol{\nu}$	generalized velocity vector for **nonlinear** configuration space joint
$\boldsymbol{q}$	generalized coordinates vector for **linear** configuration space joint
$\dot{\boldsymbol{q}}$	generalized velocity vector for **linear** configuration space joint
$\ddot{\boldsymbol{q}}$	generalized acceleration vector for **linear** configuration space joint
DoF	degrees of freedom of joint or system
Ad	adjoint matrix
J	Jacobian matrix
$\dot{J}$	time derivative Jacobian matrix
m	mass of body
$\mathcal{I}$	inertia matrix of body
$\mathcal{L}$	Lagrangian
M	generalized mass matrix
N	generalized nonlinear effects matrix
$\boldsymbol{n}$	generalized nonlinear effects vector
B	generalized viscosity matrix
$\boldsymbol{g}$	generalized gravity vector
Z	mechanical impedance, for example such as $\boldsymbol{f} = Z\boldsymbol{v}$
Y	mechanical admittance, inverse of impedance
`py`	in this font, class/instance/snippet of code in python language
LP	linear program
LQP	linear quadratic program
SOCP	second-order cone program
SDP	semidefinite program
LMI	linear matrix inequality
$\boldsymbol{x}$	unknown of primal optimization problem
$\boldsymbol{y}$	unknown of dual optimization problem
$\bullet^*$	optimal solution
A	equality constraints matrix
$\boldsymbol{b}$	equality constraint vector ($A\boldsymbol{x} = \boldsymbol{b}$)
G	inequality constraints matrix
$\boldsymbol{h}$	inequality constraint vector ($G\boldsymbol{x} \preceq \boldsymbol{h}$)
S	Selection matrix

$\boldsymbol{\chi}$	action variable, composed of input torque vector and contact forces
$\mathbb{X}$	dynamic variable, $\mathbb{X} = [\ddot{\boldsymbol{q}}\ \boldsymbol{\chi}]$
$\mathbf{t}$	subset of twist vector
$\mathbf{w}$	subset of wrench vector
E	coefficient matrix of the controlled part of task
$\mathbf{f}$	the objective part of task
ω	weight (importance) of considered task
$\|\bullet\|_M$	Euclidean M-weighted norm
Λ	projection of inertia matrix onto task space
CoM	center of mass
ZMP	zero moment point

Introduction

Since antiquity, people have tried to reproduce their behavior through artificial beings. The first examples can be found in mythology with the assistants created by Hephaestus, and it has continued until automatons development in the nineteenth century. These entirely mechanical systems were sometimes even programmable, like those designed by Jaquet-Droz such as the automated writer or musician. With the advent of computers in the mid-twentieth century, it became possible to control complex systems and to produce much more sophisticated programmable behaviors. This opened the area of the modern robotics and the first industrial robot, the Unimate, was integrated into car assembly lines in 1961. Robots also offer the ability to adapt their behavior by the richness of their mechanical potential and the exploitation of data from sensors that are integrated, especially data on their evolution or working environment.

Their complexity can now allow to consider interventions with some degree of autonomy in non-structured and dynamic environments by taking advantage of mechanical systems with high mobility, with the means of visual, tactile, auditory perceptions, and the capacity to plan their activities based on reasoning mechanisms taking into account uncertainties, non-determinism, etc.

Humanoid robots inherently offer a great potential to mimic the human in a number of these activities. They focus today an important number of research activities relatively diverse in nature (Fig. 0.1). This ambition to create robots like humans led to quite challenging and unique research problems compared to other systems. Beyond the philosophical concept to create an artificial being at the image of man, the complexity of these systems assume that issues are dealt with the control of highly redundant and under-actuated systems under multiple and variable constraints. The latter represent for instance motor coordination in sensorimotor optimal control and learning, multi-modal perception including visual, acoustic, otolithic and somesthetic perception as well as the physical and non-physical interaction through collaborative activities with humans.

The control of these systems must allow to produce full-body motions, obviously compatible with their physics, but also beyond that to take into account specific constraints as the dynamic of unstable equilibrium in systems with unilateral joints subject to expected or/and unexpected perturbations coming from interactions with the physical working and evolving environment. Activities combining basic tasks (such as grasping, manipulating, pushing) and postural balance must be controlled so as to meet different objectives of basic tasks, but they must also be linked smoothly to build more comprehensive activities. This sequence of elementary functions and settings for each of them should be appropriate for the state of the system, its perception of tasks (current state, performance), and the knowledge of the inner system capacities.

Some of these issues have already been addressed in several previous works. For instance, Sentis [Sentis2007] has presented an analytical solution on dynamic whole-body control where a hierarchical set of tasks is performed using null space projectors. He pro-

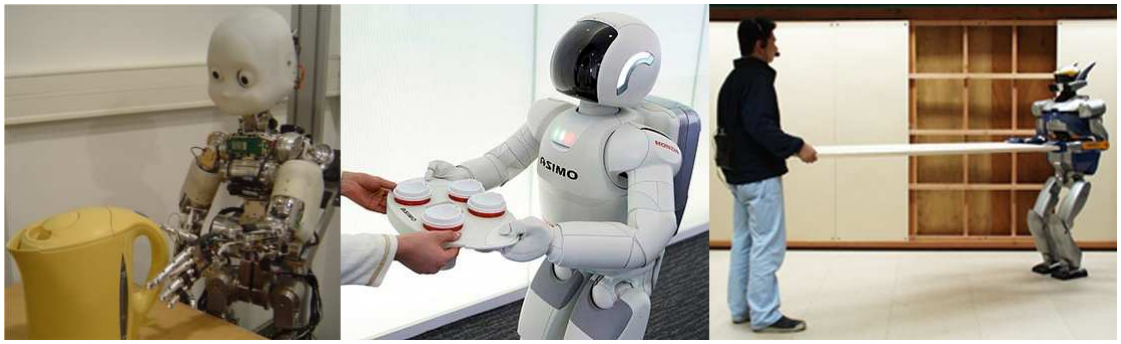


FIGURE 0.1.: Some humanoid robots realizing tasks from grabbing to co-manipulation. – Left : the iCub robot developed by the Robocub consortium [RobotCub]. Center : Asimo, developed by Toyota. Right : HRP-2 from Kawada Industries Inc. & AIST [KawadaHRP2].

posed a framework that requires each task to be "dynamically consistent". However this technique cannot take into account inequalities, which is necessary when robot motion has to satisfy very restrictive inequality constraints.

The use of kinematic redundancy in the control of robotic systems is the source of several works that began in the late 1970s, particularly with Liegeois [Liegeois1977]. They were then developed to include dynamic coupling between the movements produced to perform simultaneously various tasks by the same system [Sentis2005]. In parallel, works based on optimal control techniques [Shieh1988, Dorato1995] have yielded proposals for understanding the optimal trajectories or equilibrium issues, including laws governing particular human movements [Flash1985, Uno1989].

Considering physical constraints acting on the system while designing control schemes in order to get, at the end, a physically realizable controlled system, different control strategies have been investigated based on rigorous mathematical optimization techniques, among which quadratic control problem (QP) with inputs and/or state constraints. Nowadays, numerical methods to derive optimal control or predictive control take advantage of advances in optimization techniques, from linear programming to nonlinear convex optimization methods. With the advent of these powerful and efficient computational algorithms, real-time solutions to constrained optimal control problems are nearing a reality.

These direct methods in optimal control that transform optimal control problems into nonlinear programming problems by discretization are particularly well adapted for dealing with real world applications that have been approached using multi-objective or sequential methods.

Multi-objective optimization, which also refers to multi-attribute optimization or multi-objective programming, is the process of simultaneously optimizing two or more conflicting objectives subject to certain constraints. Successive Linear/Quadratic Programming (SLP, SQP) algorithms solve nonlinear optimization problems via a sequence of linear or quadratic programs. They have been introduced in the early 1960s, and since then several applications and variants of SLP and SQP have appeared. They are fairly easy to implement for solving non-separable as well as separable large problems [Zhang1985].

A number of control problems in robotics can be effectively solved by methods of linear, quadratic or convex programming. For instance, in [Barthelemy2008], some balance

performance measures are extracted thanks to linear programs, and active balancing is presented in [Park2007] through second-order cone programming (SOCP). Furthermore, the robot posture can be optimized with QP to reject perturbations and keep balance as presented in [Collette2009], and [Kanoun2009b] has developed the whole-body kinematic control of humanoid robots with SQP, and integrates "inequality tasks", *i.e.* tasks that may become constraints when some inequalities are not respected. In dexterous manipulation, many works intend to improve the grasping task by optimizing force distribution with Linear Programming (LP) [Liu1999], SOCP [Liu2004], and even Semi-Definite Programming (SDP) [Buss1997].

One of the most widespread modern control principles which is the discrete-time method Model Predictive Control (MPC, also called receding horizon control) exploits the solution of quadratic programming problems where constraints on control signals and states can easily be handled [Bemporad2002]. In each time step, MPC requires the solution of a quadratic program minimizing a cost function over a receding horizon. Recently, the interest focuses on the use of MPC for controlling systems containing a mix of continuous and integer unknown variables. Unfortunately, when this problem is formulated as an optimization problem, the result is no longer a QP problem but a Mixed Integer Quadratic Programming (MIQP) problem whose solution is more difficult to obtain.

These optimal control processes must work in conjunction with some planning algorithm to perform the synthesis of complex behaviors in nondeterministic environments. Tasks sequencing, which is required for the achievement of complex missions, has to be defined from dynamics and logical rules based on supervisors that analyze tasks statuses and achievements. In industry, some tools allow to control sequences of dynamic events, for instance by using finite-state machine [Kolen2001], or by using "Graphcet" to clearly separate control and supervision [Charbonnier1995]. In robotics, decision making based on perceptual information to achieve a sequence of tasks is done in [Cambon2009]. A planning method based on symbolic actions is proposed to perform tasks and reach goals in quasi-static frameworks. The work of [Yoshida2005] describes the dynamic motion planning of humanoid robots to perform manipulation and locomotion tasks without colliding obstacles, and to adjust trajectories when dynamic effects lead to infeasible motions.

The objective of the work developed in this thesis aims to address the problem of "synthesis" of dynamic behaviors when controlling humanoid robots or virtual humans carrying out activities with physical interactions with the environment, or in others words we try to establish in real-time optimal trajectory control methods, by optimizing the system behavior with respect to a given goal while satisfying the dynamics, as well as the state and actuation constraints.

In this aim, we addressed the following problems :

- The control problem is formulated and solved as a quadratic programming problem with equality and inequality constraints to find the nonlinear mapping between desired torques and positions. It leads to the design of a generic whole-body dynamic controller based on optimization programs.
- We consider the coordination and synthesis of complex activities by the decomposition of the problem into sub-problems, while ensuring the continuity of solutions. This is a very general issue that can be considered either by successive or mul-

ticriteria algorithms. Here we pay a particular attention to task transition when changes happen in the set of tasks and constraints, using a weighing strategy and soft evolutions, to ensure proper control.
– Finally, we consider the problem of dynamic planning of motor activity for solving problems defined as overall functional objectives. We address this issue by incorporating a means of supervision and decision on actions.

Hence, this work is divided into four chapters. In the first one, we provide a formulation of the equations of motion of constrained multi-body systems and control systems. This formulation is guided by the specification of the implementation that led to the software Arboris-python.

The second chapter focuses on optimization problems. The internal and external robotic limitations are established and the appropriate program is selected, according to the constraints type. Then, we recall how to solve unconstrained and constrained convex problems. This serves to understand the formulations and resolution methods of LQP which is the optimization program in the core of our controller. This LQP optimizes a cost function based on a set of tasks, sometimes conflicting, so it has to make a choice or a trade-off to perform them at the same time. We focus on the building of this cost function with a homogeneous cost, and the means to manage transitions in a sequence of tasks.

In the third chapter, we present some experiments which are carried out on the simulator Arboris-python with the virtual humanoid robot iCub, to validate the techniques described throughout this thesis. The LQP-based low-level controller is tested through some sequences where several tasks are performed simultaneously using relatively complex methods to interact with the world, and their performances in term of precision and computation time are detailed through three simulations.

In the last chapter, the low-level controller is connected to a high-level decision making engine. Decisions are based on some relatively simple rules which deal with high-level operations. The simplicity and efficiency of this connection are demonstrated through some experiments where geometric parameters and dynamic effects are not known *a priori*.

Finally, the conclusion recalls the contributions, and some perspectives are suggested to continue this work according to some lines of research.

1. Modeling and simulating mechanical systems

Contents

In this chapter, a dynamic simulator is presented for prototyping and validating the methods proposed in this work, and first we introduce the formalism used to describe the dynamics of rigid-body mechanical systems.

Nowadays, the advances in information technology allow to simulate and animate virtual characters or mechanisms with even more realism. For instance, visual effects using computer graphics have become more and more important in movies of the last decades because they can render physically consistent mechanics, fluids, explosions, without any expensive special effects or devices. Their rendering is possible thanks to a graphics engine on one hand, and a physics engine on the other hand, the first by processing lights,

shadows and textures frame-by-frame, and the second by computing the scene state at the next frame. Here we focus on this second type of engine for simulating mechanical systems.

Obviously, there are several different engines available but all are not suitable for a particular problem. Typically, the most interested domains are the cinema, the video game industry and the research departments for mechanical systems. In the video game industry, the some popular physics engines are Havok™ [Havok] (non-free), PhysX [PhysX] (free for non-commercial use), Newton Game Dynamics [NewtonDyn], ODE [ODE] and Bullet [Bullet] (open sources). Generally, this field requires real-time processing to get good gameplay and good rendering with no latency, so the physical consistency of the generated motions can be lost to fasten the algorithms by simplifying models, collision detections, etc. In movies, the purpose is a little different because it is a matter of aesthetics, one can cheat with physics for entertaining reasons, so motions can be readjusted off-line in post-production. When more realistic rendering is necessary, some movies use Bullet, especially computer-animated films. Finally, researchers in mechanics and robotics have many tools available to design their structures and controllers, and in addition to ODE and Bullet there is also Adams [Adams], Microsoft Robotics Studio [MRS], OpenHRP3 [OpenHRP], Webots (based on ODE [1]) [Webots] and XDE [XDE]. Note also that many specific dynamic simulators have been developed in robotics, for instance SAI [Khatib2002, SAI], GraspIt ! [Miller2004] to manage grasping tasks with complex objects, or the simulator integrated in AMELIF [Chardonnet2009] for interactive simulations.

Although most of simulators exploit the equations of motion derived from the Newton-Euler or Lagrange formalisms, the ways of modeling these systems vary with the kind of application and the associated limitations, for instance *global coordinates* permit to deal with the whole set of parameters, independent or not, and *generalized coordinates* focus on the subset of parameters representing the configuration of a constrained system (for instance by some joints). The first case is employed in many dynamic simulators because no assumption is made on the links connecting the different parts (Adams, Bullet, ODE) and the resolution of all the constraints is done simultaneously ([Baraff1996] uses the constraint matrix sparsity for efficient resolutions). The second case reduces the problem size by integrating the holonomic joint constraints directly in the equations of motion, so the forward dynamics can be computed by some very efficient algorithms, as employed in XDE. Furthermore, the generalized acceleration computation can be done with the Featherstone algorithm [Featherstone1983, Mirtich1996] whose complexity is proportional to the number of joints, which is very interesting for ragdoll physics in video games or real-time robotic control with OpenHRP3 (AIST edition) [2].

Nevertheless, many constraints cannot be integrated in the equations of motion, for instance non-holonomic or unilateral constraints, and solving the dynamic equations requires some specific algorithms.

These constraints are generally replaced by some constraint forces, and when they are properly computed they ensure that the dynamics equations of motion are satisfied. A way to make this calculation is to use a penalty method, as in Adams or XDE,

1. iCub Simulator [iCubSim] is a simulator specifically designed for the iCub platform [Sandini2007], also based on ODE.

2. Two version of OpenHRP3 have been developed in parallel, the AIST and Tokyo university editions. For more information : http://www.openrtp.jp/openhrp3/en/about.html

which returns some forces according to the distance which measures the constraints violation. It is a simple method to quickly find a solution, but it does not always satisfy all the constraints (for instance, some little interpenetrations may occur when dealing with contacts) and may lead to oscillating behaviors. One can also use direct methods which solve the problem in a finite number of iterations, for example conjugate gradient methods in Newton Game Dynamics, or iterative methods to obtain the solution with a given tolerance, for example Gauss-Seidel methods in Bullet. Note that XDE can also use a Gauss-Seidel-like method [3] to solve the constraints [Merlhiot2011]. Finally, one can directly solve the Linear Complementary Problem (LCP) where these constraints are replaced by some Lagrange multipliers [Baraff1996], as it is done in ODE or OpenHRP3 (Tokyo edition) [4].

In this chapter we propose a new dynamic simulator, Arboris-python, to quickly and easily design robots and robotic controller. Whereas most of the programs described above are written in C++, Arboris-python focuses on the prototyping of the algorithms by using the python language which is weakly typed and does not require any compilation. Arboris-python is very handy to quickly model and simulate multibody systems composed of rigid bodies interconnected in a tree-structure by joints, and it gives access to modeling, controllers and constraints. The dynamics of these systems are computed in a form similar to the Boltzmann-Hamel equations [Greenwood2003, Duindam2007]. Using time-stepping and a semi-implicit Euler integration scheme, a first-order approximation of the model is also computed. This allows for additional constraints such as contacts and kinematic loops to be solved using a Gauss-Seidel algorithm [Albu2002, Liu2005].

Many works have already focused on the description of rigid-body dynamics, from single body to multiple bodies systems, composed of unilateral and bilateral joints, holonomic or not, etc. In this work, we recall the equations of rigid bodies mechanisms in a way suitable to the design of Arboris-python, meaning that we look for the matrix forms of the relations which describe the dynamic behavior of the system. This serves as theoretical basis for understanding how Arboris-python is designed and the ways to model, control, integrate the world and solve the constraints.

The first part of this chapter aims to extract the kinematic and dynamic equations that give information on the future robot state. It focuses on the matrix equations that rule the dynamics of rigid body systems. The second part describes the algorithms needed to simulate mechanical systems subject to several constraints. It focuses on the theoretical issues through the methods implemented in Arboris-python, which represent the core of the simulator. More details about practical implementation, modeling and control are given in Appendix A.1.

1.1. Modeling mechanical systems

Constrained kinematic or dynamic mechanical system models are the basis for controller design, particularly for model-based control system analysis and design, where explicit dynamic equations of motion are required. General methods were formulated for this purpose more than 200 years ago and have been actively and continuously worked

3. XDE use also penalty method, but it is mainly employed to get real-time applications for get proper haptic feedbacks (about 1 kHz).

4. In [Chardonnet2009], it is stated that the AIST edition of OpenHRP2 uses penalty method.

on since by many engineers, mathematicians and scientists. Analytical mechanics, which essentially make use of variational principles to obtain the explicit formulation of equations of motion for mechanical systems including constrained systems, have had a fruitful association with control theory from its birth. In the past decade, the emergence of a geometric theory for nonlinear control systems closely linked to the modern geometric formulation of analytical mechanics. The geometrical approaches to the rigid multi-body systems dynamics in [Stramigioli2001b, Chevallier2006, Merlhiot2009] exploit the matrix representation of $SE(3)$ and its Lie group structure. A Lie-group formulation for the kinematics and dynamics of holonomic constrained mechanical systems (CMS) is presented in [Muller2003], where the group properties are implicitly used to algebraically derive the Lagrangian rigid multi-body systems equations of motion.

In the purpose of this thesis, we restrain our study to matrix representations in order to get the means to quickly implement these equations in Arboris-python. So the mechanics description is conducted as follows. First, the geometric problem of position and orientation representation in space is addressed with the Special Orthogonal and Euclidean groups $SO(3)$ and $SE(3)$, as well as a redundant description of configurations in space through homogeneous representation. Second, the system evolution over time is introduced with the exponential representation of applications belonging to $SO(3)$ and $SE(3)$, which leads to the description of velocities with twists. Then, the dynamic of a simple body is described through its inertial properties and external wrenches in order to obtain the equations of motion. The last section gives the principles of rigid-body mechanical systems by explaining the representation and the dynamics of tree-structures.

1.1.1. Coordinates in space

The study of mechanical systems evolution describes the motion of any point with minimal data thanks to the relations which exist between the different particles of the whole system. In robotics, most of mechanisms are composed of parts considered as rigid, whose relative motions are constrained by joints. Besides, all the particles of the same part perform similar motions. Considering that these parts have infinite stiffness, the application which describes their relative positions is called **a rigid transformation**.

Before dealing with entire complex systems, one needs to know how to represent its basic elements. First, a point in space $\boldsymbol{p}$ is defined by 3 real components $\boldsymbol{p} = (x, y, z) \in \mathbb{R}^3$ which stand for its projection onto the three axes of a Cartesian frame. Second, a vector $\boldsymbol{u} \in \mathbb{R}^3$ is the difference between two points, $\boldsymbol{p}$ and $\boldsymbol{q}$, such as $\boldsymbol{u} = \boldsymbol{p} - \boldsymbol{q}$. Finally, a frame Ψ_a is defined by a point $\boldsymbol{p}_a$ and three orthogonal unit vectors $\boldsymbol{x}_a, \boldsymbol{y}_a, \boldsymbol{z}_a$ with a right-handed representation, meaning that they satisfy $\boldsymbol{x}_a \times \boldsymbol{y}_a = \boldsymbol{z}_a$ where $\times$ represents the cross product of two vectors in $\mathbb{R}^3$.

From this, the objective is to characterize the motion of any point $\boldsymbol{p}$ with respect to the time t such as $\boldsymbol{p}(t) = (x(t), y(t), z(t))$ using the fact that particles of a rigid body have no relative motion, *i.e.* the distance between any couple of points remains constant. In this purpose, a rigid transformation is defined as an application $f : \mathbb{R}^3 \to \mathbb{R}^3$ which preserves distance and cross-product, and leads to the following properties

- $\|f(\boldsymbol{p}) - f(\boldsymbol{q})\| = \|\boldsymbol{p} - \boldsymbol{q}\|$ for all points $\boldsymbol{p}, \boldsymbol{q}$,
- $f(\boldsymbol{u} \times \boldsymbol{v}) = f(\boldsymbol{u}) \times f(\boldsymbol{v})$ for all vectors $\boldsymbol{u}, \boldsymbol{v}$.

All these applications transform any right-handed coordinate frame into another right-handed coordinate frame, so it gives a way to relate all the points of the same body, and to describe their motions with a minimum knowledge.

1.1.1.1. Orientation with the special orthogonal group $SO(3)$

Rotation transformations can be represented for instance through rotation matrices. Let $\Psi_a = (\boldsymbol{p}_a, \boldsymbol{x}_a, \boldsymbol{y}_a, \boldsymbol{z}_a)$ be an inertial coordinate frame, and $\Psi_b = (\boldsymbol{p}_b, \boldsymbol{x}_b, \boldsymbol{y}_b, \boldsymbol{z}_b)$ any frame in space. The vectors $\boldsymbol{x}_{a,b}, \boldsymbol{y}_{a,b}, \boldsymbol{z}_{a,b} \in \mathbb{R}^3$ are respectively the coordinates of the orthonormal vectors $\boldsymbol{x}_b, \boldsymbol{y}_b, \boldsymbol{z}_b$ relative to Ψ_a, and the rotation matrix $R_{a,b} \in \mathbb{R}^{3\times3}$ defines the transformation from Ψ_a to Ψ_b such as

$$R_{a,b} = \begin{bmatrix} \boldsymbol{x}_{a,b} & \boldsymbol{y}_{a,b} & \boldsymbol{z}_{a,b} \end{bmatrix}$$

Given a rotation matrix $R = [\boldsymbol{r}_x, \boldsymbol{r}_y, \boldsymbol{r}_z]$, its rows and columns are orthonormal, which is expressed in the following equalities

$$\begin{aligned} \boldsymbol{r}_i^\mathsf{T} \boldsymbol{r}_i &= 1 \quad \forall i \\ \boldsymbol{r}_i^\mathsf{T} \boldsymbol{r}_j &= 0 \quad \forall i \neq j \end{aligned}$$

and give the following assertions

$$RR^\mathsf{T} = R^\mathsf{T} R = I_3 \tag{1.1}$$

$$R^{-1} = R^\mathsf{T} \tag{1.2}$$

$$\det(R) = \pm 1 \tag{1.3}$$

where I_3 is the identity matrix in $\mathbb{R}^3$. As the previous statement asserts that the coordinate frames orientation must be "right-handed", the last equality becomes $\det(R) = 1$.

Associated to matrix multiplication, the set of matrices in $\mathbb{R}^{3\times3}$ which have these properties is denoted by $SO(3)$ for the **Special Orthogonal group** in the 3-dimensional space [Murray1994, Mahony2009] with the following definition

$$SO(3) = \left\{ R \in \mathbb{R}^{3\times3} \mid R^{-1} = R^\mathsf{T},\ \det(R) = 1 \right\}. \tag{1.4}$$

Hence, the representation of any vector in different coordinate frames is done with elements in $SO(3)$. Given a vector $\boldsymbol{u}$, its coordinates can be written $\boldsymbol{u}^a = (u_{xa}\boldsymbol{x}_a + u_{ya}\boldsymbol{y}_a + u_{za}\boldsymbol{z}_a)$ relative to the frame Ψ_a and $\boldsymbol{u}^b = (u_{xb}\boldsymbol{x}_b + u_{yb}\boldsymbol{y}_b + u_{zb}\boldsymbol{z}_b)$ relative to the frame Ψ_b. The relation that links these coordinates is given by the rotation matrix $R_{a,b}$ as $\boldsymbol{u}^a = R_{a,b}\boldsymbol{u}^b$. Furthermore, given a third frame Ψ_c, the coordinates $\boldsymbol{u}^c = (u_{xc}\boldsymbol{x}_c + u_{yc}\boldsymbol{y}_c + u_{zc}\boldsymbol{z}_c)$ are related to $\boldsymbol{u}^b$ with the rotation matrix $R_{b,c}$ as $\boldsymbol{u}^b = R_{b,c}\boldsymbol{u}^c$, and the association with the equation above gives $\boldsymbol{u}^a = R_{a,c}\boldsymbol{u}^c$ with

$$R_{a,c} = R_{a,b}R_{b,c}. \tag{1.5}$$

By extension, it is possible to get a succession of frames where each frame is expressed with respect to the previous one, and Eq.(1.5) is then called **the composition rule for rotations**.

1.1.1.2. Configuration with the special Euclidean group $SE(3)$

The previous section focuses on orientation representation and rotational transformation, but the rigid body configuration in space also needs some information about the body position. Given two frames Ψ_a and Ψ_b, their locations in space are given by their

origins $\boldsymbol{p}^a$ and $\boldsymbol{p}^b$, so with Ψ_a the reference frame, the coordinates of $\boldsymbol{p}^b$ expressed in Ψ_a are denoted by $\boldsymbol{p}_{a,b} \in \mathbb{R}^3$, and their relative configuration in space is defined by the couple $(R_{a,b}, \boldsymbol{p}_{a,b})$.

These transformations represent **rigid motions**. Given a point $\boldsymbol{p}$ and its coordinates $\boldsymbol{p}^a = (p_{xa}\boldsymbol{x}_a + p_{ya}\boldsymbol{y}_a + p_{za}\boldsymbol{z}_a)$ and $\boldsymbol{p}^b = (p_{xb}\boldsymbol{x}_b + p_{yb}\boldsymbol{y}_b + p_{zb}\boldsymbol{z}_b)$ expressed respectively in Ψ_a and Ψ_b, the transformation equation from the first one to second one is given as follows (see Fig. 1.1)

$$\boldsymbol{p}^a = \boldsymbol{p}_{a,b} + R_{a,b}\boldsymbol{p}^b. \tag{1.6}$$

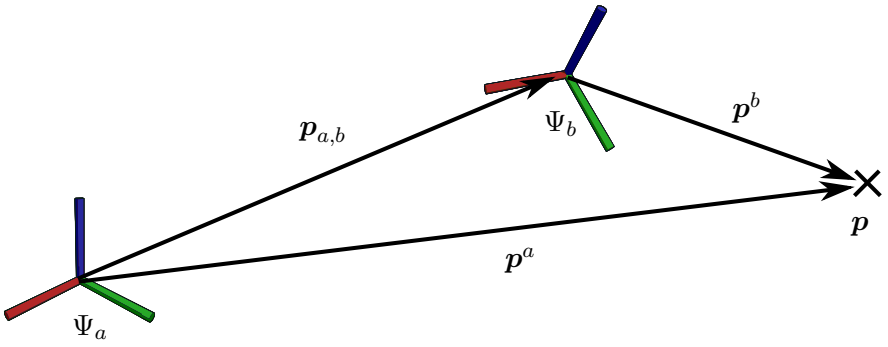


FIGURE 1.1.: transformation from $\boldsymbol{p}$ expressed in Ψ_a to $\boldsymbol{p}$ expressed in Ψ_b.

However, even if it allows to describe very easily any transformation in space, one has to pay attention to the nature of the transformed items. Equation (1.6) describes the method for a point, but the coordinates transformation of a vector $\boldsymbol{u}$, which is the difference between two points $\boldsymbol{u} = \boldsymbol{p} - \boldsymbol{q}$, is a little different. Denoting by $\boldsymbol{u}^a$ and $\boldsymbol{u}^b$ its coordinates in frames Ψ_a and Ψ_b, the transformation only needs the rotation matrix, as shown below

$$\begin{aligned} \boldsymbol{u}^a &= \boldsymbol{p}^a - \boldsymbol{q}^a \\ &= \boldsymbol{p}_{a,b} + R_{a,b}\boldsymbol{p}^b - (\boldsymbol{p}_{a,b} + R_{a,b}\boldsymbol{q}^b) \\ &= R_{a,b}\boldsymbol{u}^b \end{aligned} \tag{1.7}$$

which indicates that a distinction must be done between points and vectors transformations.

This differentiation of treatment may become a problem if the nature of a triplet is unknown. Fortunately, this issue can be overcome with the use of homogeneous coordinates which integrate the affine component in their representations. So, given a point $\boldsymbol{p}^a$ and a vector $\boldsymbol{u}^a$ expressed in Ψ_a, their respective homogeneous coordinates $\widetilde{\boldsymbol{p}}^a$ and $\widetilde{\boldsymbol{u}}^a$ yield in $\mathbb{R}^4$ such as

$$\widetilde{\boldsymbol{p}}^a = \begin{bmatrix} p_{xa} \\ p_{ya} \\ p_{za} \\ 1 \end{bmatrix} \qquad \widetilde{\boldsymbol{u}}^a = \begin{bmatrix} v_{xa} \\ v_{ya} \\ v_{za} \\ 0 \end{bmatrix}.$$

Thus, it becomes very easy to concatenate the information of a rigid motion from the frame Ψ_a to the frame Ψ_b characterized by the couple $(R_{a,b} = [\ \boldsymbol{x}_{a,b} \quad \boldsymbol{y}_{a,b} \quad \boldsymbol{z}_{a,b}\ ],\ \boldsymbol{p}_{a,b})$. The rotational components are vectors, the translation component is a point, and the

gathering of these elements in a single matrix written in homogeneous coordinates becomes

$$H_{a,b} = \left[\begin{array}{cccc} \widetilde{\boldsymbol{x}}_{a,b} & \widetilde{\boldsymbol{y}}_{a,b} & \widetilde{\boldsymbol{z}}_{a,b} & \widetilde{\boldsymbol{p}}_{a,b} \end{array} \right] = \left[\begin{array}{cc} R_{a,b} & \boldsymbol{p}_{a,b} \\ \mathbf{0} & 1 \end{array} \right], \tag{1.8}$$

where $H_{a,b}$ is called **the homogeneous transformation matrix** from Ψ_a to Ψ_b.

Hence, similarly to $SO(3)$, it is possible to define $SE(3)$ as the **Special Euclidean Group** in the 3-dimensional space [Murray1994] which regroups the applications that describe translation and rotation transformations in space. Associated to matrix multiplication, it is written as

$$SE(3) = \left\{ H = \left[\begin{array}{cc} R & \boldsymbol{p} \\ \mathbf{0} & 1 \end{array} \right] \mid R \in SO(3),\ \boldsymbol{p} \in \mathbb{R}^3 \right\}. \tag{1.9}$$

Hence, elements in $SE(3)$ can specify orientation in space, but also translation transformation from a frame to another.

The linear relations defined in Eqs.(1.6)–(1.7) can be written with the same element belonging to $SE(3)$, and thanks to homogeneous coordinates the following properties occur

- $\widetilde{\boldsymbol{u}}^a = H_{a,b}\widetilde{\boldsymbol{u}}^b$ for all vectors $\boldsymbol{u}$,
- $\widetilde{\boldsymbol{p}}^a = H_{a,b}\widetilde{\boldsymbol{p}}^b$ for all points $\boldsymbol{p}$,
- given two points $\widetilde{\boldsymbol{p}}^a, \widetilde{\boldsymbol{q}}^a$, the difference $\widetilde{\boldsymbol{u}}^a = \widetilde{\boldsymbol{p}}^a - \widetilde{\boldsymbol{q}}^a$ is a vector,
- given two vectors $\widetilde{\boldsymbol{u}}^a, \widetilde{\boldsymbol{v}}^a$, the sum/difference $\widetilde{\boldsymbol{w}}^a = \widetilde{\boldsymbol{u}}^a \pm \widetilde{\boldsymbol{v}}^a$ is a vector,
- given a point $\widetilde{\boldsymbol{p}}^a$ and a vector $\widetilde{\boldsymbol{u}}^a$, the sum $\widetilde{\boldsymbol{q}}^a = \widetilde{\boldsymbol{p}}^a + \widetilde{\boldsymbol{u}}^a$ is a point.

It follows that **homogeneous matrix composition** shown in Fig. 1.2 can be expressed similarly to the rotation matrix composition, and let $\widetilde{\boldsymbol{p}}$ be the homogeneous coordinates of a point in space, it follows that (the same applies to vectors by replacing $\widetilde{\boldsymbol{p}}$ with $\widetilde{\boldsymbol{u}}$)

$$\widetilde{\boldsymbol{p}}^a = H_{a,b}\widetilde{\boldsymbol{p}}^b \qquad \widetilde{\boldsymbol{p}}^b = H_{b,c}\widetilde{\boldsymbol{p}}^c \qquad \widetilde{\boldsymbol{p}}^a = H_{a,c}\widetilde{\boldsymbol{p}}^c$$

with

$$H_{a,c} = H_{a,b}H_{b,c} = \left[\begin{array}{cc} R_{a,b}R_{b,c} & R_{a,b}\boldsymbol{p}_{b,c} + \boldsymbol{p}_{a,b} \\ \mathbf{0} & 1 \end{array} \right]. \tag{1.10}$$

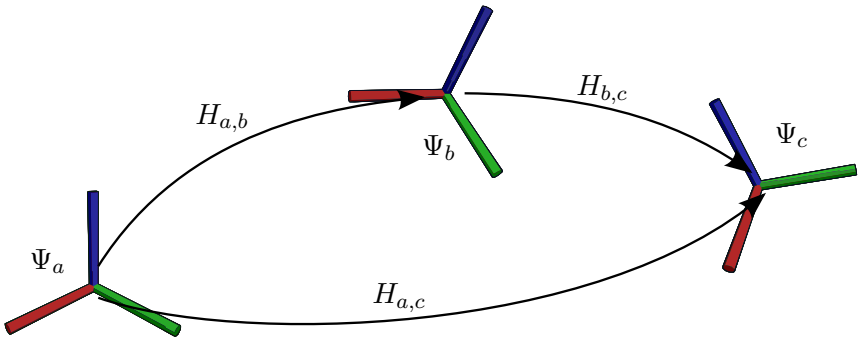


FIGURE 1.2.: Composition rule for homogeneous matrices.

As a group, every element lying in $SE(3)$ is invertible, the identity element being represented by I_4, and the inverse homogeneous matrix can be deduced from the last

equation as follows

$$H_{a,b}^{-1} = H_{b,a} = \begin{bmatrix} R_{b,a} & \boldsymbol{p}_{b,a} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R_{a,b}^{\mathsf{T}} & -R_{a,b}^{\mathsf{T}}\boldsymbol{p}_{a,b} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{1.11}$$

1.1.2. Rigid motion velocity

The previous section describes the position and orientation of any parts of the system with rigid motions. This can be viewed as a "snapshot" of the world at one instant, and the location and orientation of coordinate frames from a reference is deduced through the applications of the group $SE(3)$. The objective of this section is to define the evolution of the system over time by extending the previous methods. This is done through the exponential representation of the components lying in the groups $SO(3)$ and $SE(3)$ and by the introduction of the twist representation.

1.1.2.1. Exponential representation in $SO(3)$

Rotation matrix, which belongs to $\mathbb{R}^{3\times3}$, gives a representation method for rotation in space, but its elements are not all independent and it is not the only way to represent rotation transformation. In fact, it exists several ways to represent rotations in space, for example one can use Euler angles, quaternion, or axis-angle representation, each of them having their advantages and drawbacks. This section describes the exponential representation of rotation. The main interest is to implicitly link the orientation of a coordinate frame with its rotational velocity. This leads to the description of its evolution over time.

For now, we only consider the rotational motion of a vector $\boldsymbol{u}$. Given two frames Ψ_o (for origin) and Ψ_e (for end) where Ψ_e rotates with respect to Ψ_o with a constant velocity, the coordinates of vector $\boldsymbol{u}$ are constant in Ψ_e but vary in Ψ_o as shown in Fig. 1.3. The evolution of $\boldsymbol{u}$ over time is described by the derivative of the equation between its coordinates in the two frames, which is

$$\begin{aligned} \frac{d}{dt}(\boldsymbol{u}^o(t)) &= \frac{d}{dt}(R_{o,e}(t)\boldsymbol{u}^e) \\ \dot{\boldsymbol{u}}^o(t) &= \dot{R}_{o,e}(t)\boldsymbol{u}^e \\ \dot{\boldsymbol{u}}^o(t) &= \dot{R}_{o,e}(t)R_{o,e}^{\mathsf{T}}(t)\boldsymbol{u}^o(t) \end{aligned} \tag{1.12}$$

Thus, a vector linked with its derivative over time through a simple linear expression. But the expression of the matrix $\dot{R}_{o,e}(t)$ is unknown and may make the development of this last equation impossible.

The idea is to focus on the product $\dot{R}_{o,e}R_{o,e}^{\mathsf{T}}$. This multiplication recalls Eq.(1.1) about the orthogonality of rotation matrix. Thus, by computing the derivative of this product with respect to time, it follows

$$\begin{aligned} \dot{R}_{o,e}R_{o,e}^{\mathsf{T}} + R_{o,e}\dot{R}_{o,e}^{\mathsf{T}} &= 0 \\ \dot{R}_{o,e}R_{o,e}^{\mathsf{T}} &= -\left(\dot{R}_{o,e}R_{o,e}^{\mathsf{T}}\right)^{\mathsf{T}} \end{aligned}$$

which proves that this multiplication is antisymmetric. It implies that the 3 components of the diagonal are necessarily null because they are equal to their opposite, and that the

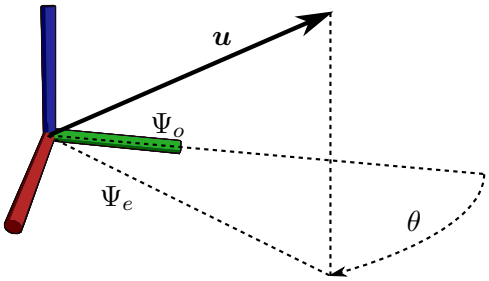


FIGURE 1.3.: Rotation of vector $\boldsymbol{u}$ linked to Ψ_e about Ψ_o with an angle θ.

three terms of the upper triangular part are constrained to be the opposite of the three other terms of the lower triangular part. Hence, the matrix $\dot{R}_{o,e}R_{o,e}^{\mathsf{T}}$ is only composed of 3 independent terms, and it is possible to find a vector $\boldsymbol{\omega} \in \mathbb{R}^3$ and its related skew-symmetric matrix $\widehat{\boldsymbol{\omega}}$ such as [5]

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \qquad \widehat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \qquad \widehat{\boldsymbol{\omega}} = \dot{R}_{o,e}R_{o,e}^{\mathsf{T}}. \tag{1.13}$$

All these skew-symmetric matrices $\widehat{\boldsymbol{\omega}}$ belong to the vector space $\mathfrak{so}(3)$ which is **the Lie algebra** of $SO(3)$. The Lie algebra of a Lie group can be defined as the tangent space of the group at the identity element. In this case, $\mathfrak{so}(3)$ represents the vector space tangent to the group of rotations, that is the rotational velocities.

Returning to Eq.(1.12), omitting the subscript and superscript for clarity, the instantaneous velocity of $\boldsymbol{u}$ becomes

$$\dot{\boldsymbol{u}}(t) = \widehat{\boldsymbol{\omega}}\boldsymbol{u}(t) \tag{1.14}$$

which is a time invariant linear differential equation with a well-know solution. Given its initial value $\boldsymbol{u}(0)$, the result of this differential equation is

$$\boldsymbol{u}(t) = e^{\widehat{\boldsymbol{\omega}}t}\boldsymbol{u}(0) \qquad \text{with} \qquad e^{\widehat{\boldsymbol{\omega}}t} = \sum_{i=0}^{\infty} \frac{(\widehat{\boldsymbol{\omega}}t)^i}{i!} = I_3 + \widehat{\boldsymbol{\omega}}t + \frac{(\widehat{\boldsymbol{\omega}}t)^2}{2!} + \frac{(\widehat{\boldsymbol{\omega}}t)^3}{3!} + \cdots$$

Of course, the expression of the last term is quite complicated, and a brute force computation can be imprecise and time consuming. To handle it, one has to point out that the successive powers of the matrix $\widehat{\boldsymbol{\omega}}$ only depend on $\widehat{\boldsymbol{\omega}}$ and $\widehat{\boldsymbol{\omega}}^2$. After some steps, the final formulation of the exponential representation of rotation is [Murray1994]

$$e^{\widehat{\boldsymbol{\omega}}t} = I_3 + \frac{\sin(\|\boldsymbol{\omega}\|\, t)}{\|\boldsymbol{\omega}\|}\widehat{\boldsymbol{\omega}} + \frac{(1 - \cos(\|\boldsymbol{\omega}\|\, t))}{\|\boldsymbol{\omega}\|^2}\widehat{\boldsymbol{\omega}}^2. \tag{1.15}$$

This equation is also true for $\boldsymbol{\omega} \to \mathbf{0}$. This particular case occurs when there is no velocity, which implies no rotation and a transformation equal to the identity, that is $e^{\widehat{\boldsymbol{\omega}}t} = I_3$.

Hence, a relation exists between exponential representation and rotation matrices. Indeed, while rotating about a constant rotational velocity $\boldsymbol{\omega}$ during a time t, the point

5. Note that the cross product between two vectors is equivalent to the left multiplication of the second one with the related skew-symmetric matrix of the first one, *i.e.* $\boldsymbol{a} \times \boldsymbol{b} \Leftrightarrow \widehat{\boldsymbol{a}}\boldsymbol{b}$.

$\boldsymbol{p}$ moves through a circular path with an angle $\theta = \|\boldsymbol{\omega}\| t$. So if $\boldsymbol{u}$ represents the unit vector collinear to $\boldsymbol{\omega}$ with the same direction, the last equation becomes

$$R(\boldsymbol{u}, \theta) = e^{\widehat{\boldsymbol{u}}\theta} = I_3 + \sin(\theta)\widehat{\boldsymbol{u}} + (1 - \cos(\theta))\widehat{\boldsymbol{u}}^2. \tag{1.16}$$

1.1.2.2. Exponential representation in $SE(3)$

Similarly to the previous section, the exponential representation of rigid motion is a way to relate the coordinates of a point in space with its velocity and to describe its evolution over time.

Here, the reference frame is Ψ_o and the moving frame Ψ_e moves relative to Ψ_o with constant rotational and translational velocities as shown in Fig. 1.4. The point $\boldsymbol{p}$ is rigidly linked with Ψ_e and the expression of its evolution over time is given by the derivative of its coordinates in the two frames, which is written as follows

$$\begin{aligned} \dot{\boldsymbol{p}}^o(t) &= \dot{R}_{o,e}(t)\boldsymbol{p}^e + \dot{\boldsymbol{p}}_{o,e}(t) \\ &= \dot{R}_{o,e}(t)\left(R_{o,e}^{\mathsf{T}}(t)\boldsymbol{p}^o(t) - R_{o,e}^{\mathsf{T}}(t)\boldsymbol{p}_{o,e}(t)\right) + \dot{\boldsymbol{p}}_{o,e}(t) \end{aligned} \tag{1.17}$$

This equation is linear and links the velocity $\dot{\boldsymbol{p}}^o$ of point $\boldsymbol{p}^o$ with its coordinates in the reference frame.

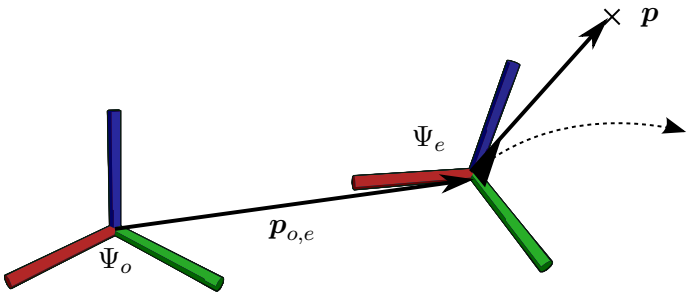


FIGURE 1.4.: Displacement of point $\boldsymbol{p}$ linked to frame Ψ_e relative to Ψ_o with a constant velocity.

This formulation is close to rotation transformation of Eq.(1.12), so the expression can be modified to lead to an exponential solution. This is done through homogeneous representation. To do so, the vector and affine parts of the equation are separated into two components, the first one being $\widehat{\boldsymbol{\omega}} = \dot{R}_{o,e}R_{o,e}^{\mathsf{T}} \in \mathbb{R}^{3\times3}$ and the second one being $\boldsymbol{v} = -\widehat{\boldsymbol{\omega}}\boldsymbol{p}_{o,e} + \dot{\boldsymbol{p}}_{o,e} \in \mathbb{R}^3$. In this way, their concatenation in a matrix $\widehat{\boldsymbol{T}}$ allows to rewrite Eq.(1.17) with homogeneous coordinates such as (again, the subscripts and superscripts are omitted for clarity)

$$\begin{bmatrix} \dot{\boldsymbol{p}} \\ 0 \end{bmatrix} = \begin{bmatrix} \widehat{\boldsymbol{\omega}} & \boldsymbol{v} \\ \boldsymbol{0} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} \quad \Leftrightarrow \quad \dot{\widetilde{\boldsymbol{p}}} = \widehat{\boldsymbol{T}}\widetilde{\boldsymbol{p}}. \tag{1.18}$$

The matrices $\widehat{\boldsymbol{T}}$ belong to $\mathfrak{se}(3)$, the Lie algebra of $SE(3)$. Similarly to $\mathfrak{so}(3)$, it represents in this case the vector space tangent to the group of configuration in space, that is the velocities in space.

Returning to Eq.(1.18), the desired time invariant linear differential equation with respect to $\widetilde{\boldsymbol{p}}$ is obtained. The notation of matrix $\widehat{\boldsymbol{T}}$ is explained by the fact that it plays the same role as $\widehat{\boldsymbol{\omega}}$ which is the conversion of the vector $\boldsymbol{\omega}$ composed of 3 independent parameters to a matrix expressed in $\mathbb{R}^{3\times 3}$. Similarly, the matrix $\widehat{\boldsymbol{T}} \in \mathfrak{se}(3)$ is related to a vector composed of 6 independent parameters denoted by $\boldsymbol{T}$, and named a twist. This vector is more detailed in the next section.

So, given the initial conditions of the point $\widetilde{\boldsymbol{p}}(0)$, the solution of Eq.(1.18) is

$$\widetilde{\boldsymbol{p}}(t) = e^{\widehat{\boldsymbol{T}}t}\widetilde{\boldsymbol{p}}(0). \tag{1.19}$$

This matrix-exponential form is quite difficult to obtain [JonghoonPark2005], one needs to notice the good properties of $\widehat{\boldsymbol{T}}$ and its successive powers [Murray1994]. It follows that solution of Eq.(1.19) is obtained by the development of the exponential representation $e^{(\widehat{\boldsymbol{T}}t)}$ such as

$$\underset{\boldsymbol{\omega}\neq\mathbf{0}}{e^{(\widehat{\boldsymbol{T}}t)}} = \begin{bmatrix} e^{(\widehat{\boldsymbol{\omega}}t)} & \left((I_3 - e^{(\widehat{\boldsymbol{\omega}}t)})\widehat{\boldsymbol{\omega}} + \boldsymbol{\omega}\boldsymbol{\omega}^{\mathsf{T}}t\right)\frac{\boldsymbol{v}}{\|\boldsymbol{\omega}\|^2} \\ \mathbf{0} & 1 \end{bmatrix} \qquad \underset{\boldsymbol{\omega}=\mathbf{0}}{e^{(\widehat{\boldsymbol{T}}t)}} = \begin{bmatrix} I_3 & \boldsymbol{v} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{1.20}$$

which is an element of the group $SE(3)$.

1.1.2.3. Twist

The exponential representation of rigid motion expressed in Eq.(1.20) depends on $\boldsymbol{\omega}$ and $\boldsymbol{v}$, which give information about the relative rotational and translational velocities between two coordinate frames. These particular values are independent but the matrix $\widehat{\boldsymbol{T}}$ used to compute Eq.(1.18) is overabundant. So, **the twist** is introduced as the vector $\boldsymbol{T}^o_{o,e}$ (and the related matrix $\widehat{\boldsymbol{T}}^o_{o,e} \in \mathfrak{se}(3)$) which is the concatenation of $\boldsymbol{\omega}^o_{o,e}$ and $\boldsymbol{v}^o_{o,e}$, such as

$$\boldsymbol{T}^o_{o,e} = \begin{bmatrix} \boldsymbol{\omega}^o_{o,e} \\ \boldsymbol{v}^o_{o,e} \end{bmatrix} \qquad \widehat{\boldsymbol{T}}^o_{o,e} = \begin{bmatrix} \widehat{\boldsymbol{\omega}}^o_{o,e} & \boldsymbol{v}^o_{o,e} \\ \mathbf{0} & 0 \end{bmatrix} \qquad \dot{\widetilde{\boldsymbol{p}}}^o = \widehat{\boldsymbol{T}}^o_{o,e}\widetilde{\boldsymbol{p}}^o.$$

As the order of $\boldsymbol{\omega}$ and $\boldsymbol{v}$ in twists is not very important, we have arbitrarily chosen in this thesis to follow the convention adopted by [Stramigioli2001a]. Twist can also be computed from the time derivative of the related homogeneous transformation as follows

$$\widehat{\boldsymbol{T}}^c_{a,b} = H_{c,a}\dot{H}_{a,b}H_{b,c} \qquad \widehat{\boldsymbol{T}}^o_{o,e} = \dot{H}_{o,e}H^{-1}_{o,e} \qquad \widehat{\boldsymbol{T}}^e_{o,e} = H^{-1}_{o,e}\dot{H}_{o,e} \tag{1.21}$$

The twist denoted by $\boldsymbol{T}^c_{a,b}$ is defined as the vector describing the rotational and translational velocities of Ψ_c rigidly linked to Ψ_b which moves relative to Ψ_a. Using twists in the velocity description allows to characterize all particles velocities in the same body at the same time. Indeed, if one has the expression of a twist expressed in Ψ_a, it is straightforward to get the twist expressed in Ψ_b when these two frames are rigidly linked. The displacement of this twist from Ψ_a to Ψ_b is done through the following equations

$$\boldsymbol{\omega}^a_{o,e} = R_{a,b}\boldsymbol{\omega}^b_{o,e} \tag{1.22}$$

$$\boldsymbol{v}^a_{o,e} = R_{a,b}\boldsymbol{v}^b_{o,e} + \widehat{\boldsymbol{p}}_{a,b}R_{a,b}\boldsymbol{\omega}^b_{o,e}. \tag{1.23}$$

These equations can be expressed in a matrix form, and **the adjoint matrix** $Ad_{a,b}$ which depends on the homogeneous matrix $H_{a,b}$ is defined to describe the displacement from Ψ_a to Ψ_b in the following manner

$$Ad_{a,b} = \begin{bmatrix} R_{a,b} & 0 \\ \widehat{\boldsymbol{p}}_{a,b} R_{a,b} & R_{a,b} \end{bmatrix} \qquad \boldsymbol{T}^a_{o,e} = Ad_{a,b} \boldsymbol{T}^b_{o,e}. \tag{1.24}$$

Twists are also useful to describe the composition of velocities between several bodies. The idea is similar to the composition of rotation or homogeneous matrix where the relation from a frame to another is built from the links which compose the chain. Given two twists $\boldsymbol{T}^d_{a,b}$ and $\boldsymbol{T}^d_{b,c}$, the twist $\boldsymbol{T}^d_{a,c}$ is computed such as

$$\boldsymbol{T}^d_{a,c} = \boldsymbol{T}^d_{a,b} + \boldsymbol{T}^d_{b,c}. \tag{1.25}$$

From this simple equation, some simple assertions can be highlighted. For example, a rigid body has no relative motion with itself, so for all frames Ψ_a, Ψ_b and Ψ_d one can write that $\boldsymbol{T}^d_{a,b} = -\boldsymbol{T}^d_{b,a}$. Furthermore, two frames Ψ_b and Ψ_c belonging to the same body have no relative motion, which implies that for all frames Ψ_a and Ψ_d one gets $\boldsymbol{T}^d_{a,b} = \boldsymbol{T}^d_{a,c}$.

For the rest of the thesis, the twist denoted by $\boldsymbol{T}^a$ refers to the twist $\boldsymbol{T}^a_{0,a}$ where the frame Ψ_0 is the Cartesian frame (the origin). The same goes for vectors $\boldsymbol{\omega}^a$ and $\boldsymbol{v}^a$.

1.1.3. Dynamics

This section focuses on the systems dynamic equations. The previous sections give methods to describe the configuration of coordinate frames over time, but it does not explain how to affect their evolutions. Actually, this is done through forces and torques. The following section explains how these forces are represented and how they act on rigid body motions.

1.1.3.1. Wrench

Wrenches represent the forces and torques acting on rigid bodies. Given the force $\boldsymbol{f}^o_{o,e} \in \mathbb{R}^3$ (the linear component) and the moment of force $\boldsymbol{\tau}^o_{o,e} \in \mathbb{R}^3$ (the rotational component) which are the actions of body Ψ_o on Ψ_e expressed in Ψ_o, the related wrench denoted by $\boldsymbol{W}^o_{o,e}$ is the concatenation of the two elements such as

$$\boldsymbol{W}^o_{o,e} = \begin{bmatrix} \boldsymbol{\tau}^o_{o,e} \\ \boldsymbol{f}^o_{o,e} \end{bmatrix}. \tag{1.26}$$

Wrenches belong to $\mathfrak{se}^*(3)$, the dual vector space of $\mathfrak{se}(3)$. Indeed, the order of the parameters is not arbitrary, it depends upon the order given to the twists parameters, so each component is located at the same position as its respective twist counterpart. Similarly to twist, the wrench denoted by $\boldsymbol{W}^a$ refers to the $\boldsymbol{W}^a_{0,a}$ where Ψ_0 is the Cartesian frame, and the same goes for $\boldsymbol{\tau}^a$ and $\boldsymbol{f}^a$.

The duality between wrench and twist allows to compute the infinitesimal work $\delta\mathcal{W}$ and the total work between times t_1 and t_2 acting on a body as follows

$$\delta\mathcal{W} = (\boldsymbol{W}^a)^\mathsf{T} \boldsymbol{T}^a = (\boldsymbol{\tau}^a)^\mathsf{T} \boldsymbol{\omega}^a + (\boldsymbol{f}^a)^\mathsf{T} \boldsymbol{v}^a \qquad \mathcal{W} = \int_{t_1}^{t_2} (\boldsymbol{W}^a(t))^\mathsf{T} \boldsymbol{T}^a(t)\, dt\ .$$

Similarly to twist, the expression of a wrench at one frame is linked with its expression on another frame of the same rigid body. However, if the adjoint defined in Eq.(1.24) allows to displace twists, the wrench displacement is a little different. To do so, one has to notice that the infinitesimal work $\delta\mathcal{W}$ does not depend upon the frame of the body where it is computed, so it implies that

$$\begin{aligned}\delta\mathcal{W} &= (\boldsymbol{W}^a)^\mathsf{T}\boldsymbol{T}^a = (\boldsymbol{W}^a)^\mathsf{T}(Ad_{a,b}\boldsymbol{T}^b) = (\boldsymbol{W}^b)^\mathsf{T}\boldsymbol{T}^b \\ \boldsymbol{W}^b &= Ad_{a,b}^\mathsf{T}\boldsymbol{W}^a. \end{aligned} \tag{1.27}$$

The reader should be aware of the dual relation it exists between twist and wrench, resulting in the use of similar operators, but in different ways ($\boldsymbol{T}^a = Ad_{a,b}\boldsymbol{T}^b$ but $\boldsymbol{W}^b = Ad_{a,b}^\mathsf{T}\boldsymbol{W}^a$). Finally, when several wrenches act simultaneously on one body, they can be concatenated in a unique acting wrench. For example, if two wrenches $\boldsymbol{W}^a$ and $\boldsymbol{W}^b$ act on the frames Ψ_a and Ψ_b (rigidly linked), the global wrench is computed in Ψ_c as

$$\boldsymbol{W}^c_{body} = Ad_{a,c}^\mathsf{T}\boldsymbol{W}^a + Ad_{b,c}^\mathsf{T}\boldsymbol{W}^b.$$

1.1.3.2. Dynamic equation of rigid body in space

This section determines how wrenches provide the possibility to change body motion. The Newton's laws of motion give a part of the answer, especially the second one which relates the force acting on a point to its acceleration, the well-known formula $\boldsymbol{f} = m\boldsymbol{a}$ where m is the mass of the point. This section extends this relation to a rigid body from the Newton-Euler equations.

Dynamic equation The rigid-body equation of motion arises from the derivative of the momentum with respect to time. The computation is done by separating translational and rotational parts [Murray1994] and leads to the following equations

$$M^b\dot{\boldsymbol{T}}^b + N^b\boldsymbol{T}^b = \boldsymbol{W}^b \tag{1.28}$$

$$M^b = \begin{bmatrix} \mathcal{I}^b & m_b\widehat{\boldsymbol{r}} \\ m_b\widehat{\boldsymbol{r}}^\mathsf{T} & m_bI_3 \end{bmatrix} \qquad N^b = \begin{bmatrix} \widehat{\boldsymbol{\omega}}^b\mathcal{I}^b & m_b\widehat{\boldsymbol{r}}\widehat{\boldsymbol{\omega}}^b \\ m_b\widehat{\boldsymbol{\omega}}^b\widehat{\boldsymbol{r}}^\mathsf{T} & m_b\widehat{\boldsymbol{\omega}}^b \end{bmatrix}$$

where Ψ_b is the body frame, M^b and N^b are respectively the generalized mass matrix [6] and the generalized nonlinear effects matrix of body b expressed in Ψ_b, $\boldsymbol{\omega}^b$ is the rotational part of $\boldsymbol{T}^b$, $\boldsymbol{r}$ (retrieved from M^b) represents the center of mass location relative to Ψ_b, $m_b(\in \mathbb{R}^{+*})$ is the body mass, and $\mathcal{I}^b(\in \mathbb{R}^{3\times3})$ is the body inertia matrix expressed in Ψ_b.

In a general manner, $\mathcal{I}^b$ is computed as follows

$$\mathcal{I}^b = -\int_V \rho(\boldsymbol{p}^b)\widehat{\boldsymbol{p}^b}^2\, dV = \begin{bmatrix} \mathcal{I}_{xx} & \mathcal{I}_{xy} & \mathcal{I}_{xz} \\ \mathcal{I}_{xy} & \mathcal{I}_{yy} & \mathcal{I}_{yz} \\ \mathcal{I}_{xz} & \mathcal{I}_{yz} & \mathcal{I}_{zz} \end{bmatrix} \tag{1.29}$$

where V is the volume of the body, ρ is the local density, $\boldsymbol{p}^b$ is the integration variable (a point) locating the infinitesimal volume in V, and $\widehat{\boldsymbol{p}^b}$ is the related skew-matrix.

6. also called kinetic energy matrix.

Although this equation is not quite simple at first sight, inertia is easy to compute when Ψ_b **coincides with the body principal axes of inertia** Ψ_c. In this case, it becomes diagonal $\mathcal{I}^c = \text{diag}([\ \mathcal{I}_{xx} \quad \mathcal{I}_{yy} \quad \mathcal{I}_{zz}\ ])$ and $\mathcal{I}_{xx}, \mathcal{I}_{yy}, \mathcal{I}_{zz} \in \mathbb{R}^{+*}$ are called the principal moments of inertia. Then, the computation of the related kinetic energy Ek_b of body b is obtained as follows

$$\begin{aligned} Ek_b &= \frac{1}{2} m_b (\boldsymbol{v}^c)^\mathsf{T} \boldsymbol{v}^c + \frac{1}{2} (\boldsymbol{\omega}^c)^\mathsf{T} \mathcal{I}^c \boldsymbol{\omega}^c \\ &= \frac{1}{2} (\boldsymbol{T}^c)^\mathsf{T} M^c \boldsymbol{T}^c \\ M^c &= \begin{bmatrix} \mathcal{I}^c & 0 \\ 0 & m_b I_3 \end{bmatrix}. \end{aligned} \tag{1.30}$$

We retrieve the generalized mass matrix M^c, this time expressed in Ψ_c. The computation is frame-independent, so the expression of the generalized mass matrix M^a relative to another frame Ψ_a belonging to the same body is given as follows

$$Ek_b = \frac{1}{2} (\boldsymbol{T}^c)^\mathsf{T} M^c \boldsymbol{T}^c = \frac{1}{2} (\boldsymbol{T}^a)^\mathsf{T} M^a \boldsymbol{T}^a$$

with

$$M^a = Ad_{c,a}^\mathsf{T} M^c Ad_{c,a}. \tag{1.31}$$

1.1.4. Representation of tree-structure

The previous sections describe the frames and bodies representation, but even if rigid motion is discussed, no information is given about the relative velocities of different bodies, and for now they only have free motions in space. This section deals with the representation of dynamic multibody systems in robotics.

1.1.4.1. Open kinematic chain

When two rigid bodies are linked, their relative velocities are constrained through the use of *joints*. Each body has parents and children bodies connected with joints, which implicitly leads to represent mechanical structures called **kinematic chains** as graphs composed of rigid bodies as nodes, and joints as links. This definition is extended to **open kinematic chains** where no kinematic cycle occurs, *i.e.* the graph is a tree, as shown in Fig. 1.5. If no detail is given, the following relations refer to this last definition.

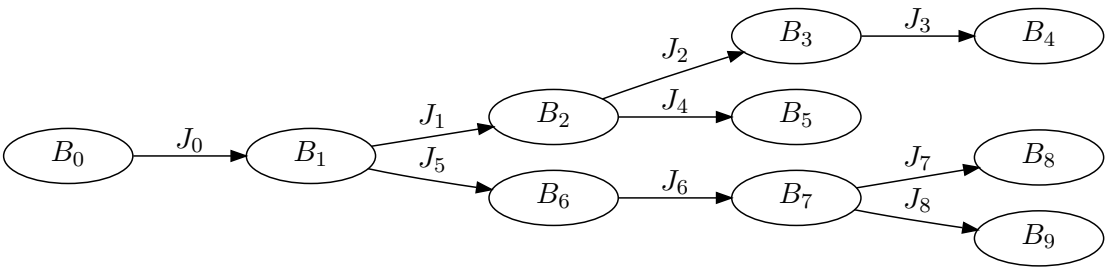


FIGURE 1.5.: Graph representing an open kinematic chain. There is no kinematic loop, *i.e.* it is a tree.

Joints We consider in this thesis the use of joints which are holonomic and bilateral, *i.e.* they transmit power without loss, their constraints are geometric (they are not expressed in terms of velocity parameters), and there are no inequality constraints. They are also globally parameterized rigid joints, as defined in [Duindam2006]. It means that when the twist $\boldsymbol{T}^c_{p,c}$ (Ψ_p and Ψ_c are respectively the parent and child frame belonging to two different rigid bodies) is mapped to a linear subspace of dimension k_j, then the joint j can be characterized by two parameters, a matrix Q_j which parameterizes the relative configuration [7], and the vector $\boldsymbol{\nu}_j \in \mathbb{R}^{k_j}$ which parameterizes the relative twist. Although the joint configuration can be described by a vector of dimension k_j (for instance Euler angles for ball joint), its time derivative may not represent the real relative velocity between Ψ_p and Ψ_c, implying the necessity to use Q_j and $\boldsymbol{\nu}_j$.

So for simplicity, this thesis only considers joints that evolve in linear configuration spaces, so the configuration is given by $\boldsymbol{q}_j \in \mathbb{R}^{k_j}$ and $\dot{\boldsymbol{q}}_j \in \mathbb{R}^{k_j}$, which respectively represent the generalized coordinates and the generalized velocities. The difference between these two types of joints is shown in Fig. 1.6.

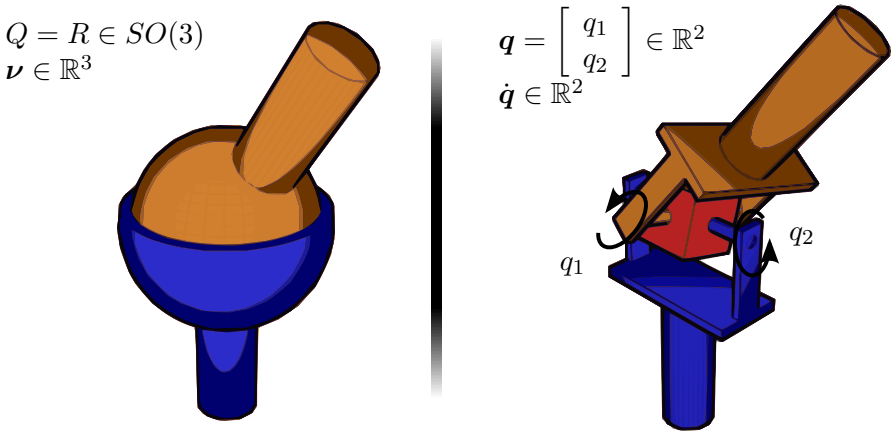


FIGURE 1.6.: Left : ball joint whose configuration is described by a matrix Q (which is a rotation matrix) and a vector $\boldsymbol{\nu}$. Right : Cardan joint whose configuration is described by two vectors $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$.

The relative position, orientation and velocity between the linked frames Ψ_p and Ψ_c are expressed as follows

$$H_{p,c} = H_{p,c}(\boldsymbol{q}_j)$$
$$\boldsymbol{T}^c_{p,c} = X(\boldsymbol{q}_j)\dot{\boldsymbol{q}}_j$$

where $X(\boldsymbol{q}_j)$ represents the domain of evolution of the joint j. The number of degrees of freedom, commonly called **DoF**, is then defined as the dimension of this domain, that is the number k_j. When $\boldsymbol{q}_j$ is null, the homogeneous transformation $H_{p,c}$ is the identity matrix, *i.e.* Ψ_p and Ψ_c are coincident.

Thus, the configuration between two rigid bodies linked with this joint is fully parameterized by the couple $(\boldsymbol{q}_j, \dot{\boldsymbol{q}}_j)$. Generally, systems are composed of several bodies connected with joints, and as their configurations lie on linear subspaces, it is possible

7. Q_j may refer to rotation or homogeneous matrices to describe configurations. However, it evolves in a k_j-dimensional subspace.

to concatenate their generalized coordinates and velocities, leading to the construction of the variables $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, the generalized coordinates and velocity of the entire system, such as

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_1 \\ \vdots \\ \boldsymbol{q}_n \end{bmatrix} \qquad \dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\boldsymbol{q}}_1 \\ \vdots \\ \dot{\boldsymbol{q}}_n \end{bmatrix}$$

where n is the number of joints. Their dimension is $n_{dof} = \sum_{j=0}^{n} k_j$ which is the total number of DoF in the system, and the joint space can be mapped on $\mathbb{R}^{n_{dof}}$.

Forward kinematics The computation of position and orientation of any frame is done using the composition rule for homogeneous matrix exposed in Eq.(1.11). Given the path of the kinematic chain between a reference frame Ψ_0 and the ending frame Ψ_e, the m joints are extracted to obtain their generalized coordinates $\boldsymbol{q}_i$, as well as the $m-1$ bodies to get the transformation matrices $H_{c_i,p_{i+1}}$ from joint i to joint $i+1$. Thus, the homogeneous matrix $H_{0,e}$ which is the transformation from Ψ_0 to Ψ_e is computed as

$$H_{0,e} = H_{0,p_1} \left(\prod_{i=1}^{m-1} H_{p_i,c_i}(\boldsymbol{q}_i) H_{c_i,p_{i+1}} \right) H_{p_m,c_m}(\boldsymbol{q}_m) H_{c_m,e} \tag{1.32}$$

This computation is called **the forward kinematics**. On the other hand, retrieving the generalized parameters $\boldsymbol{q}_i$ from the knowledge of the homogeneous matrix $H_{0,e}$ is called the inverse kinematics problem, whose solutions, if any, are more difficult to obtain.

Forward velocity model and Jacobian Twist can be computed from the time derivative of the related homogeneous matrix as described in Eq.(1.21), or one can exploit the tree structure of the system. Indeed, similarly to the forward kinematics, the twist computation of any frame can be done using the twist composition rule presented in Eq.(1.25). Given the same problem as above, the twist of Ψ_e relative to Ψ_0 expressed in any frame Ψ_a is computed as

$$\boldsymbol{T}^a_{0,e} = \boldsymbol{T}^a_{0,p_1} + \sum_{i=1}^{m-1} \left(\boldsymbol{T}^a_{p_i,c_i}(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i) + \boldsymbol{T}^a_{c_i,p_{i+1}} \right) + \boldsymbol{T}^a_{p_m,c_m}(\boldsymbol{q}_m, \dot{\boldsymbol{q}}_m) + \boldsymbol{T}^a_{c_m,e}.$$

This expression can be simplified. Indeed, some twists (those that do not depend on the generalized parameters) represent the velocity between two frames rigidly linked, and are consequently null. Thus, twists only depend on the joint velocities, which lead to the following formulation

$$\boldsymbol{T}^a_{0,e} = \sum_{i=1}^{m} \boldsymbol{T}^a_{p_i,c_i}(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i) = \sum_{i=1}^{m} Ad_{a,c_i} \boldsymbol{T}^{c_i}_{p_i,c_i}(\boldsymbol{q}_i, \dot{\boldsymbol{q}}_i) = \sum_{i=1}^{m} Ad_{a,c_i} X_i(\boldsymbol{q}_i) \dot{\boldsymbol{q}}_i \tag{1.33}$$

This computation is called the **forward velocity model**, and the mapping from the Cartesian space to the joint space is named the inverse velocity model. This is a very challenging issue especially for redundant robots like humanoids, and the literature is very abundant in this domain.

The twist $\boldsymbol{T}^a_{0,e}$ in Eq.(1.33) depends linearly on the generalized velocities of the joints composing the kinematic chain. However, it does not take into account the whole set of joints. In order to relate $\boldsymbol{T}^a_{0,e}$ with $\dot{\boldsymbol{q}}$, generalized velocities of the remaining joints are multiplied by 0 and added in the equation above. It converts instantaneous velocities from the joint space (in $\mathbb{R}^{n_{dof}}$) into the Cartesian space (in $SE(3)$) where the translational and rotational velocities are projected onto the reference frame Ψ_0. This leads to the definition of the natural Jacobian $J^a_{0,e}$ which relates the twist $\boldsymbol{T}^a_{0,e}$ to the generalized velocity $\dot{\boldsymbol{q}}$ such as

$$\boldsymbol{T}^a_{0,e} = J^a_{0,e}(\boldsymbol{q})\dot{\boldsymbol{q}} = \begin{bmatrix} \delta^1_{0,e} Ad_{a,c_1} X_1(\boldsymbol{q}_1) & \cdots & \delta^n_{0,e} Ad_{a,c_n} X_n(\boldsymbol{q}_n) \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{q}}_1 \\ \vdots \\ \dot{\boldsymbol{q}}_n \end{bmatrix} \tag{1.34}$$

$$\delta^i_{0,e} = \begin{cases} 1 & \text{if joint } i \text{ is in the kinematic chain between } \Psi_0 \text{ and } \Psi_e, \\ 0 & \text{otherwise.} \end{cases}$$

Twist displacement and composition operations expressed in Eqs.(1.24)–(1.25) are also true for Jacobians, which implies that $J^a_{0,e} = Ad_{a,b} J^b_{0,e}$ and $J^a_{0,e} = J^a_{0,b} + J^a_{b,e}$ for all frames $\Psi_0, \Psi_e, \Psi_a, \Psi_b$.

For clarity, the following notation J_e refers to the Jacobian $J^e_{0,e}$.

Force acting on a kinematic chain The last point about kinematic chains is the effect of wrenches applied on structures. In Section 1.1.3, the behavior of a single body is studied when it is subject to a wrench. But this is different when it acts on a kinematic chain due to the fact that the motions are constrained by the joints of the mechanical structure.

This section presents the transcription of wrench acting on a frame into a set of forces and torques applied at the joint level, *i.e.* the way to transform a wrench expressed in the Cartesian space into a vector of forces and torques expressed in the joint space. This problem is similar to the one explained above, so denoting by $\boldsymbol{\tau}_{w_i}$ the forces and torques applied at the i^{th} joint, the vector $\boldsymbol{\tau}_w$ is introduced as

$$\boldsymbol{\tau}_w = \begin{bmatrix} \boldsymbol{\tau}_{w_1} \\ \vdots \\ \boldsymbol{\tau}_{w_n} \end{bmatrix}$$

which is named the generalized force induced by this wrench and has the same dimension as $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. However, the link between the wrench $\boldsymbol{W}^a$ and its generalized force denoted by $\boldsymbol{\tau}_{w_a}$ is not yet established. To do so, one has to notice that the mechanical work created by the wrench $\boldsymbol{W}^a$ and the twist $\boldsymbol{T}^a$ is equal to the sum of the mechanical works produced by the joints, that is

$$\mathcal{W} = (\boldsymbol{W}^a)^\mathsf{T} \boldsymbol{T}^a = (\boldsymbol{W}^a)^\mathsf{T} J_a \dot{\boldsymbol{q}} \qquad \text{or} \qquad \mathcal{W} = \sum_{i=1}^{n} \boldsymbol{\tau}_i^\mathsf{T} \dot{\boldsymbol{q}}_i = \boldsymbol{\tau}_{w_a}^\mathsf{T} \dot{\boldsymbol{q}}$$

and by identification

$$\boldsymbol{\tau}_{w_a} = J_a^\mathsf{T} \boldsymbol{W}^a. \tag{1.35}$$

1.1.4.2. Dynamic of tree-structure

The last objective is to determine the equations that rule the behavior of tree-structure systems in space depending upon a minimal set of parameters, which are their generalized parameters expressed in joint space. This is done through the computation of the kinetic and potential energies.

Kinetic energy The total kinetic energy is quite simple to obtain because it is the summation of the kinetic energies of its subsystems, already computed in Eq.(1.30). Then, the computation of the kinetic energy Ek of the whole system is

$$\begin{aligned} Ek(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \sum_b Ek_b = \frac{1}{2} \sum_b (\boldsymbol{T}^b)^\mathsf{T} M^b \boldsymbol{T}^b = \frac{1}{2} \dot{\boldsymbol{q}}^\mathsf{T} M(\boldsymbol{q}) \dot{\boldsymbol{q}} \\ M(\boldsymbol{q}) &= \sum_b J_b^\mathsf{T}(\boldsymbol{q}) M^b J_b(\boldsymbol{q}) \end{aligned} \tag{1.36}$$

where b represents the bodies, and M is called the generalized mass matrix (or kinetic energy matrix) of the system, which is symmetric and positive-definite.

Potential energy The potential energy defines the energy accumulated by the system when it gains altitude. This energy depends upon a frame of reference, and for a single rigid body b, it is computed as follows

$$Ep_b(\boldsymbol{q}) = g m_b h_b(\boldsymbol{q})$$

where $\boldsymbol{q}$ is the generalize coordinates vector, g represents the gravitational constant, m_b is the mass of b, and $h_b(\boldsymbol{q})$ is its altitude.

Again, the potential energy of the system is the sum of all subsystems potential energies, so

$$Ep(\boldsymbol{q}) = g \sum_b m_b h_b(\boldsymbol{q}). \tag{1.37}$$

Equations of motion Finally, the equations of motion of tree-structures evolving in space are computed thanks to its Lagrangian, defined as the difference between kinetic and potential energies $\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = Ek(\boldsymbol{q}, \dot{\boldsymbol{q}}) - Ep(\boldsymbol{q})$. The equations of motion are derived from this expression and the Lagrange's equations [Murray1994]

$$\frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i \quad \forall i \tag{1.38}$$

where $q_i, \dot{q}_i$ and τ_i are respectively the independent (linear or angular) coordinate, velocity and force applied at the i^{th} degree of freedom.

Then, a compact representation of the equations of motion of an open kinematic chain can be written as follows

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + N(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau} + \sum_j \boldsymbol{\tau}_{w_j} \tag{1.39}$$

where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ are respectively the generalized coordinates, velocity and acceleration vectors, $M(\boldsymbol{q})$, $N(\boldsymbol{q}, \dot{\boldsymbol{q}})$, $\boldsymbol{g}(\boldsymbol{q})$ are the generalized inertia matrix, the nonlinear effects matrix

(Coriolis and centrifugal) and the generalized gravity force vector, that is the summation of all the gravitational forces expressed in generalized coordinates. Finally, the generalized force vector $\boldsymbol{\tau}$ represents the forces applied by joint actuators, and the vectors $\boldsymbol{\tau}_{w_j}$ denote the generalized forces induced by some external wrenches and undergone by the system, for instance friction, adhesion, contact, etc.

1.2. Simulating mechanical systems with Arboris-python

The aim of this section is to describe the algorithms used in our dynamic simulator Arboris-python to simulate the evolution of the mechanical systems. Indeed, thanks to equations of motion of a tree-structure Eq.(1.39), one can predict during a short period of time the robot state evolution. However, the study is bounded to open kinematic chains, *i.e.* to robots without any kinematic loop closure. This means that, for a humanoid robot, interactions with the environment are not determined, behaviors are unknown when it touches the ground or when it grabs an object with its hands, which is very restrictive. The extension of this model can be done with different methods and leads to the use of a robotic dynamic simulator.

Here, we present the core of the dynamic simulator Arboris-python [Arboris] which allows to model and simulate many rigid-body systems with kinematic loops.

Background In 2005, Alain Micaelli, a researcher from CEA LIST, wrote a first version of the simulator in the Matlab language. It was an implementation (and often an extension) of the algorithms described in [Murray1994], [Park2005] and [Liu2005].

He was later joined by Sébastien Barthélemy, from ISIR/UPMC, who reorganized the code to take advantage of the early matlab object-oriented features. It eventually became clear that the language was ill-designed, and that a full rewrite was necessary. With the help of the author of this thesis, also from ISIR/UPMC, Arboris-python was born. The resulting framework is quite similar to what is presented in [Duindam2006].

What is Arboris-python Arboris-python is a rigid multibody dynamics and contacts simulator written in python language. It includes a generic and easily extensible set of joints (singularity-free multi-DoF joints, non-honolomic joints, etc.) for modeling open rigid body mechanisms with a minimal set of state variables. It is mostly useful for robotic applications and human motion studies, and the python language makes it particularly suited for fast-paced development (prototyping) and education.

Arboris-python is very handy to quickly model and simulate a multibody system composed of rigid bodies interconnected by joints. The entire model is concatenated in a single instance of class `World`. It gives access to modeling, controllers and constraints. The equations of motion of these systems are obtained from the Boltzmann-Hamel equations [Greenwood2003, Duindam2007] in a matrix form. Arboris-python computes the first-order approximation of the model, and the integration is done with a time-stepping method and a semi-implicit Euler integration scheme. In this way, it is possible to solve the additional constraints, *i.e.* the kinematic loops which can be unilateral like contacts or bilateral like a revolute joint, with a Gauss-Seidel algorithm [Albu2002, Liu2005].

Notice that the Boltzmann-Hamel formalism is very interesting because it can integrate in the equations of motion some joints that do not evolve in linear configuration

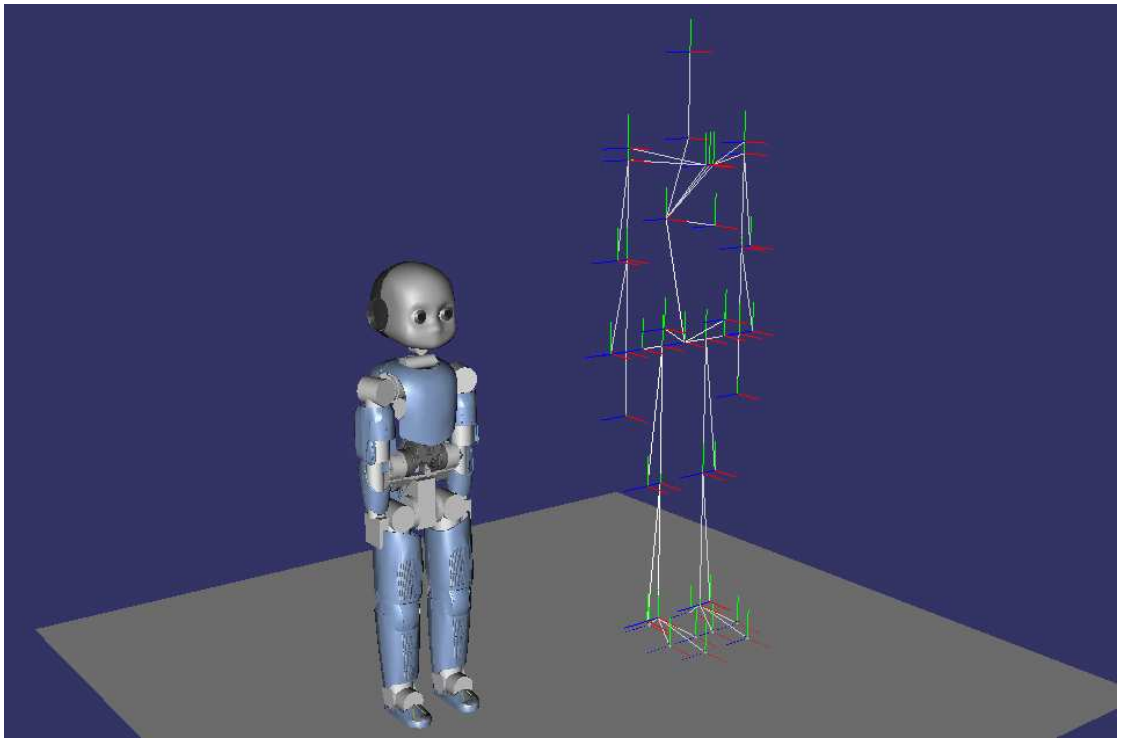


FIGURE 1.7.: View of a scene created with Arboris-python. A virtual iCub model (on the left side) stands beside a wireframe view of a human36 model [HuManS] (on the right side).

spaces, and thus avoid singular parameterizations (for instance with Euler angles to describe orientations). The configuration of such a joint is expressed with a matrix Q_j and a vector $\boldsymbol{\nu}_j$ which respectively represent the generalized coordinates and velocity of joint j, as it is illustrated in Fig. 1.6 with a ball joint. It becomes possible to write down the equations of motion in terms of Q and $\boldsymbol{\nu}$, which are loosely the concatenation of Q_j and $\boldsymbol{\nu}_j$, in a very similar manner than in Eq.(1.39), as follows

$$\overline{M}(Q)\dot{\boldsymbol{\nu}} + \overline{N}(Q,\boldsymbol{\nu})\boldsymbol{\nu} = \overline{\boldsymbol{g}}(Q) + \boldsymbol{\tau} + \sum_j \boldsymbol{\tau}_{w_j} \tag{1.40}$$

However, for simplicity reasons, the remaining of this work continues with the formalism of Eq.(1.39) to develop the algorithms used in Arboris-python. The reader should keep in mind that the Boltzmann-Hamel formalism permits to integrate more general joints without much complexity, especially to represent structure with free-floating base without singularity problem. One can refer to [Duindam2007] for more explanations.

1.2.1. Simulating the world

In Arboris-python, the simulation of the world is done through four steps which have to be called successively in the following order :

1. update the dynamic parameters of the world,
2. update the generalized forces generated by the controllers,
3. update the closure forces to ensure that the world constraints are satisfied,

4. integrate the joints positions and velocities to get the new state of the world.

These lines represent the simulation loop. The first one deals with the updating of both the world dynamic components (the matrices M, N) and the kinematic components such as the position, orientation and velocity of all frames in the system, using Eq.(1.11) and Eq.(1.33) (the frames, as well as all the other elements of world, are described in Section A.1). Notice that Arboris-python introduces a new parameter B which refers to the generalized viscosity matrix of the system. This leads to a fluid friction when multiplied by $\dot{\boldsymbol{q}}$ and permits to simulate motions in fluids, for instance in water. Denoting the bodies by b, the matrices M, N and B are computed as follows

$$M = \sum_b J_b^{\mathsf{T}} M^b J_b \qquad B = \sum_b J_b^{\mathsf{T}} B^b J_b \qquad N = \sum_b J_b^{\mathsf{T}} \left(M^b \dot{J}_b + N^b J_b \right). \tag{1.41}$$

where M^b, N^b refer to matrices in Eq.(1.28), J_b is the Jacobian matrix of Ψ_b and B^b is a constant user-defined matrix (see Section A.1.1).

The second line of the snippet of code above updates all the controllers available in `World` and computes both the input torque vector $\boldsymbol{\tau}$ applied to the system and the global admittance. This is detailed in Section 1.2.2 below. This method has an input argument `dt` which represents the time interval between two simulation loops, denoted by dt in the remaining of this section.

The third line handles the updating of the world constraints, meaning the resolution of the kinematic loop closures, an issue not addressed in the previous section. Indeed the modeling of kinematic chain composed of some closures is possible by deleting and replacing them with external wrenches whose values ensure the closures, in order to retrieve an open kinematic chain modeling. This is more detailed in Section 1.2.3 below.

Finally, the fourth line integrates the joints positions and velocities to get the new state of the world, as explained in Section 1.2.4.

1.2.2. Updating controllers

In a general manner, dynamic controllers return some generalized forces to perform the desired goals. In Arboris-python, this notion is extended with the introduction of some impedance terms, in order to give another leverage for the robotic control. The step which updates the controllers collects the generalized forces and impedances computed by all the `Controller` instances registered in the world, and it updates a so called **global mechanical admittance**.

Indeed, in order to obtain this admittance, each controller returns two arguments, the first one is the **reactive part** of control denoted by $\boldsymbol{\tau}_{c0}$, and the second one is the impedance of control (inverse of admittance) denoted by Z_c. When the latter is multiplied by the future generalized velocity $\dot{\boldsymbol{q}}(t+dt)$, one obtains the **proactive part** of control. The `Controller` class is described in Section A.1.6. Hence, considering the following integration scheme $\ddot{\boldsymbol{q}}(t+dt) = (\dot{\boldsymbol{q}}(t+dt) - \dot{\boldsymbol{q}}(t))/dt$, the first order dynamic equation of the actuated model resulting from Eq.(1.39) is written as

$$M(t)\ddot{\boldsymbol{q}}(t+dt) + \big(N(t) + B(t)\big)\dot{\boldsymbol{q}}(t+dt) = \sum_c \boldsymbol{\tau}_c(t).$$

$$\left(\frac{M(t)}{dt} + N(t) + B(t) - \sum_c Z_c(t) \right) \dot{\boldsymbol{q}}(t+dt) = \frac{M(t)}{dt}\dot{\boldsymbol{q}}(t) + \sum_c \boldsymbol{\tau}_{c0}(t) \tag{1.42}$$

where $\boldsymbol{\tau}_c$ is the generalized force returned by controller c such as $\boldsymbol{\tau}_c(t) = \boldsymbol{\tau}_{c0}(t) + Z_c(t)\dot{\boldsymbol{q}}(t+dt)$, and M, N, B are the dynamic parameters computed in Eq.(1.41). From this equation, one can introduce the world impedance Z, and admittance Y. This last matrix leads to a more compact expression of the first order model equation above, such as

$$Y(t) = Z^{-1}(t) = \left(\frac{M(t)}{dt} + N(t) + B(t) - \sum_c Z_c(t)\right)^{-1}$$
$$\dot{\boldsymbol{q}}(t+dt) = Y(t)\left(\frac{M(t)}{dt}\dot{\boldsymbol{q}}(t) + \sum_c \boldsymbol{\tau}_{c0}(t)\right). \tag{1.43}$$

1.2.2.1. Explanation about the impedance argument

This argument refers to the mechanical impedance explained for example in [Hogan1985]. It relates forces with velocities, for instance $\boldsymbol{f}(\omega) = Z(\omega)\boldsymbol{v}(\omega)$ where ω is the system frequency.

In Arboris-python, the purpose is to decompose the generalized force computed by the controller into 2 components. The first one (denoted by `gforce` in the software) represents the reactive part of the results $\boldsymbol{\tau}_{c0}$ because it only depends upon the actual state of the robot, and the second one (denoted by `impedance`) refers to the proactive part of the control which depends upon the future generalized velocity $\dot{\boldsymbol{q}}(t+dt)$. Thus, the generalized force returned by the controller is written as $\boldsymbol{\tau}_c(t) = \boldsymbol{\tau}_{c0}(t) + Z_c\dot{\boldsymbol{q}}(t+dt)$ where Z_c is the `impedance` argument.

The description of this operation is illustrated below with a proportional derivative controller with K_p and K_d the respective stiffness and damping matrices. Without any anticipation term, one should write at time t

$$\boldsymbol{\tau}_c(t) = K_p(\boldsymbol{q}_{des} - \boldsymbol{q}(t)) + K_d(\dot{\boldsymbol{q}}_{des} - \dot{\boldsymbol{q}}(t)). \tag{1.44}$$

To integrate the anticipation term into this equation, one has to consider the state of the system at the instant $t+dt$, and with the simple equation $\boldsymbol{q}(t+dt) = \boldsymbol{q}(t) + \dot{\boldsymbol{q}}(t+dt)dt$, it becomes

$$\begin{aligned}\boldsymbol{\tau}_c(t) &= K_p(\boldsymbol{q}_{des} - \boldsymbol{q}(t+dt)) + K_d(\dot{\boldsymbol{q}}_{des} - \dot{\boldsymbol{q}}(t+dt))\\ \boldsymbol{\tau}_c(t) &= \boldsymbol{\tau}_{c0}(t) + Z_c\dot{\boldsymbol{q}}(t+dt)\end{aligned} \tag{1.45}$$

with

$$\boldsymbol{\tau}_{c0}(t) = K_p(\boldsymbol{q}_{des} - \boldsymbol{q}(t)) + K_d\dot{\boldsymbol{q}}_{des} \qquad Z_c = -(dtK_p + K_d). \tag{1.46}$$

1.2.3. Updating constraints

The first order actuated model of tree-structure systems is computed in Eq.(1.43), but it does not handle kinematic loop closure constraints which may occur during simulations. The step which updates the constraints takes them into account through the `Constraint` instances. The description of this class is done in Section A.1.4. Each of them represents an external wrench which acts on the constraint frames, and the purpose of this step is to find the proper wrenches that ensures the closures. The solution is solved iteratively using the Gauss-Seidel algorithm [Albu2002, Liu2005].

For each constraint k the algorithm acts on two variables, $\mathbf{t}_k$ and $\mathbf{w}_k$ which are the images of the twist and wrench constraint $\boldsymbol{T}_k$ and $\boldsymbol{W}_k$, *i.e.* the constrained part of these vectors. For instance in the case of a ball and socket constraint, we want to prevent translational motions, so rotational motions are free, and we have

$$S = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{t}_k = S\boldsymbol{T}_k \qquad \mathbf{w}_k = S\boldsymbol{W}_k \tag{1.47}$$

The parameters $\mathbf{w}_k$ are considered as **Lagrange multipliers**. The Jacobian J_k is then defined such as $\mathbf{t}_k = J_k\dot{\boldsymbol{q}}$ and $\boldsymbol{\tau}_k = J_k^{\mathsf{T}}\mathbf{w}_k$ where $\boldsymbol{\tau}_k$ is the generalized force induced by constraint k. All these generalized forces are added to the actuation of the world, and they lead to the final model

$$\dot{\boldsymbol{q}}(t+dt) = Y(t)\left(\frac{M(t)}{dt}\dot{\boldsymbol{q}}(t) + \sum_c \boldsymbol{\tau}_{c0}(t) + \sum_k \boldsymbol{\tau}_k(t)\right). \tag{1.48}$$

To find the Lagrange multipliers values, Arboris-python focuses on the constrained subset of the world. The concatenation of the Jacobians J_k and the vectors $\mathbf{t}_k$ and $\mathbf{w}_k$ are denoted by $\overline{J}, \overline{\mathbf{t}}$ and $\overline{\mathbf{w}}$, the constraint admittance is denoted by $\overline{Y} = \overline{J}Y\overline{J}^{\mathsf{T}}$, and one can write down the following equation

$$\begin{aligned} \overline{\mathbf{t}}(t+dt) &= \overline{J}(t)\dot{\boldsymbol{q}}(t+dt) \\ \overline{\mathbf{t}}(t+dt) &= \overline{J}(t)Y(t)\left(\frac{M(t)}{dt}\dot{\boldsymbol{q}}(t) + \sum_c \boldsymbol{\tau}_{c0}(t)\right) + \overline{Y}(t)\overline{\mathbf{w}}(t) \end{aligned} \tag{1.49}$$

The first part of the equation above does not depend on the constraint forces, so the algorithm focuses on the second part. The solution is found iteratively, as presented in Algorithm (1) (we introduce the notation $\overline{\mathbf{t}}[k] = \mathbf{t}_k$ and $\overline{\mathbf{w}}[k] = \mathbf{w}_k$)[8].

Algorithm 1: Constraints resolution in Arboris-python.

Initialize $\overline{\mathbf{w}} \longleftarrow \overline{\mathbf{w}}_0$, choose ε (the tolerance)
Compute $\overline{J}$, $\overline{Y}$ and $\overline{\mathbf{t}}$ with Eq.(1.49)
repeat
  $\overline{\mathbf{t}}_{-1} \longleftarrow \overline{\mathbf{t}}$
  for *each constraint* k **do**
    $\Delta\mathbf{w}_k \longleftarrow$ solve constraint k according to $(\overline{\mathbf{t}}[k],\ \overline{Y}[k,k],\ dt)$
    $\overline{\mathbf{w}}[k] \longleftarrow \overline{\mathbf{w}}[k] + \Delta\mathbf{w}_k$ (update force of constraint k)
    $\overline{\mathbf{t}} \longleftarrow \overline{\mathbf{t}} + \overline{Y}[:,k]\Delta\mathbf{w}_k$ (update whole constrained velocities)
until $\left\|\overline{\mathbf{t}} - \overline{\mathbf{t}}_{-1}\right\| < \varepsilon$

At the end of this procedure, all the Lagrangian multipliers $\mathbf{w}_k$ are properly computed and the generalized constrained forces of Eq.(1.48) are obtained such as $\sum_k \boldsymbol{\tau}_k =$

8. In this algorithm, the notation $\boldsymbol{u}[k]$ ($M[k,k]$) corresponds to the sub-vector (sub-matrix) defined by the $k = [k_1, k_2, ..., k_n]$ rows of vector $\boldsymbol{u}$ (the k rows and k columns of matrix M). The notation $\boldsymbol{u}[:]$ ($M[:,k]$) indicates that all the vector rows (all the matrix rows and k columns) are considered. This is similar to the python language notation.

$\sum_k J_k^{\mathsf{T}} \mathbf{w}_k = \overline{J}^{\mathsf{T}} \overline{\mathbf{w}}$. However, the variable $\Delta \mathbf{w}_k$ is obtained by the procedure "solving constraint k" which depends upon the type of constraint that has to be satisfied. This kind of procedure is described in the following sections through some examples.

1.2.3.1. Joint limit

The purpose of joint limits is to maintain some generalized coordinates inside some bounded intervals. Given joint k with generalized coordinates and velocity $\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k$, and given the bounds $\boldsymbol{m}, \boldsymbol{M}$, the constraint k is $\boldsymbol{m} \preceq \boldsymbol{q}_k \preceq \boldsymbol{M}$. But this formulation does not consider the constraint force and velocity vectors.

To do so, one has to compute the future generalized coordinates $\boldsymbol{q}_k(t+dt)$ after a short period of time dt. At iteration 0, the constraint force is null, that is $\mathbf{w}_k^{(0)} = \mathbf{0}$, and noticing that the joint generalized velocity is also the constraint velocity $\dot{\boldsymbol{q}}_k(t+dt) = \mathbf{t}_k$, a simple integration gives $\boldsymbol{q}_k(t+dt) = \boldsymbol{q}_k^{(0)} = \boldsymbol{q}_k(t) + \mathbf{t}_k^{(0)} dt$. At iteration i, the velocity is modified according to the force difference, which is expressed as follows

$$\mathbf{t}_k^{(i+1)} = \mathbf{t}_k^{(i)} + \overline{Y}[k,k] \underbrace{\left(\mathbf{w}_k^{(i+1)} - \mathbf{w}_k^{(i)} \right)}_{\Delta \mathbf{w}_k} \tag{1.50}$$

To estimate $\Delta \mathbf{w}_k$, we first consider the case when no constraint forces occur in the next iteration, meaning that $\Delta \mathbf{w}_k = -\mathbf{w}_k^{(i)}$. The future generalized coordinates is then deduced

$$\boldsymbol{q}_k^{(i+1)} = \boldsymbol{q}_k(t) + \left(\mathbf{t}_k^{(i)} - \overline{Y}[k,k] \mathbf{w}_k^{(i)} \right) dt \tag{1.51}$$

which leads to three possible cases

- $\boldsymbol{m} \preceq \boldsymbol{q}_k^{(i+1)} \preceq \boldsymbol{M}$, which implies that the constraint is satisfied without any forces, so the procedure returns $\Delta \mathbf{w}_k = -\mathbf{w}_k^{(i)}$,
- $\boldsymbol{q}_k^{(i+1)} \prec \boldsymbol{m}$, meaning that the constraint is violated, so the distance must be reduced by $\boldsymbol{m} - \boldsymbol{q}_k^{(i+1)}$, the difference of velocities must be $\mathbf{t}_k^{(i+1)} - \mathbf{t}_k^{(i)} = (\boldsymbol{m} - \boldsymbol{q}_k^{(i+1)})/dt$, and the procedure returns $\Delta \mathbf{w}_k = \overline{Y}[k,k]^{-1} \left(\boldsymbol{m} - \boldsymbol{q}_k^{(i+1)} \right) / dt - \mathbf{w}_k^{(i)}$,
- $\boldsymbol{M} \prec \boldsymbol{q}_k^{(i+1)}$, meaning that the constraint is also violated. Proceeding in the same manner as above, the procedure returns $\Delta \mathbf{w}_k = \overline{Y}[k,k]^{-1} \left(\boldsymbol{M} - \boldsymbol{q}_k^{(i+1)} \right) / dt - \mathbf{w}_k^{(i)}$.

1.2.3.2. Ball and socket constraint

The ball and socket constraint allows to create kinematic loops in the mechanical structure by constraining the relative position between two frames. Given Ψ_a and Ψ_b connected together by constraint k, the homogeneous matrix $H_{a,b}$ and the constraint Jacobian J_k are written as follows

$$H_{a,b} = \begin{bmatrix} R_{a,b} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \qquad J_k = Ad_{a,b} J_{0,b}^b - J_{0,a}^a \tag{1.52}$$

where Ψ_0 is the inertial frame, $J_{0,b}^b$, $J_{0,a}^a$ are respectively the Jacobian matrix of Ψ_a and Ψ_b. The relative velocity between the two frames is given by the twist $\boldsymbol{T}_{a,b}^a = \boldsymbol{T}_k = J_k \dot{\boldsymbol{q}}$. For our problem, we only consider the translational part, so we use the variables $\mathbf{t}_k$ and

$\mathbf{w}_k$ for the constraint velocity and force, as defined in Eq.(1.47), the Jacobian J_k being reduced accordingly.

At iteration 0, the relative displacement $\boldsymbol{p}_{a,b}$ is extracted from $H_{a,b} = H_{a,0}H_{0,b}$. The force $\mathbf{w}_k$ start with a null value and must be updated to enforce that $\boldsymbol{p}_{a,b} = \mathbf{0}$. At iteration i, the new position is computed such as $\boldsymbol{p}_{a,b}^{(i+1)} = \boldsymbol{p}_{a,b}(t) + \mathbf{t}_k^{(i+1)}dt$, and given Eq.(1.50) it becomes

$$\boldsymbol{p}_{a,b}^{(i+1)} = \boldsymbol{p}_{a,b}(t) + \left(\mathbf{t}_k^{(i)} + \overline{Y}[k,k]\Delta\mathbf{w}_k\right)dt,$$

Finally, by imposing that $\boldsymbol{p}_{a,b}^{(i+1)} = \mathbf{0}$, the solution is given as follows

$$\Delta\mathbf{w}_k = -\overline{Y}[k,k]^{-1}\left(\mathbf{t}_k^{(i)} + \frac{\boldsymbol{p}_{a,b}(t)}{dt}\right). \tag{1.53}$$

1.2.4. Integrating

The integration method of Arboris-python is based on time-stepping and a semi-implicit Euler integration scheme. Even if explicit methods exist [9], as for instance Runge-Kutta which may be more precise, semi-implicit (and implicit) methods are generally more stable when simulating rigid multibody systems with contacts.

Thus, the updated generalized velocity $\dot{\boldsymbol{q}}$ is computed thanks to Eq.(1.48) after an interval of time dt. This new value is transmitted to the joints which update their inner generalized position and velocity Q_j, $\boldsymbol{\nu}_j$ (see Section 1.1.4.1) in accordance with their own integration method. It means that when the configuration of joint j lies in a linear vector space, its inner generalized position and velocity Q_j, $\boldsymbol{\nu}_j$ can be represented by vector $\boldsymbol{q}_j$, $\dot{\boldsymbol{q}}_j$, and the integration scheme is simply

$$\boldsymbol{q}_j(t+dt) = \boldsymbol{q}_j(t) + dt\dot{\boldsymbol{q}}_j(t+dt). \tag{1.54}$$

However, the integration of some joints is not straightforward, and the generalized coordinates computation is left to dedicated methods. For instance, the class `FreeJoint` included in the `arboris.joints` module describes a free motion between two frames. Thus, Q_j is equal to homogeneous matrix H_{p_j,c_j}, and its integration is done as follows

$$\begin{aligned} Q_j(t) &= H_{p_j,c_j} \\ Q_j(t+dt) &= Q_j(t)e^{(\widehat{\boldsymbol{\nu}}_j dt)} \end{aligned}$$

where $e^{(\widehat{\boldsymbol{\nu}}_j dt)}$ is the exponential representation of a rigid motion computed in Eq.(1.20).

Finally, when the integration of all joints is performed, the matrices of the world are no more up to date, and the simulator gets ready to start a new simulation step.

1.2.5. Example : 3R robot with joint limits

To illustrate the simulation process used in Arboris-python, an example is developed here for a robot composed of rigid bodies connected together with 3 revolute joints with bounded ranges (3R robot). It falls under the influence of gravity and a simple proportional derivative controller maintains the system about a reference configuration.

9. Roughly, explicit methods compute the current motion based on previous configurations, whereas implicit methods give the current motion according to the final configuration.

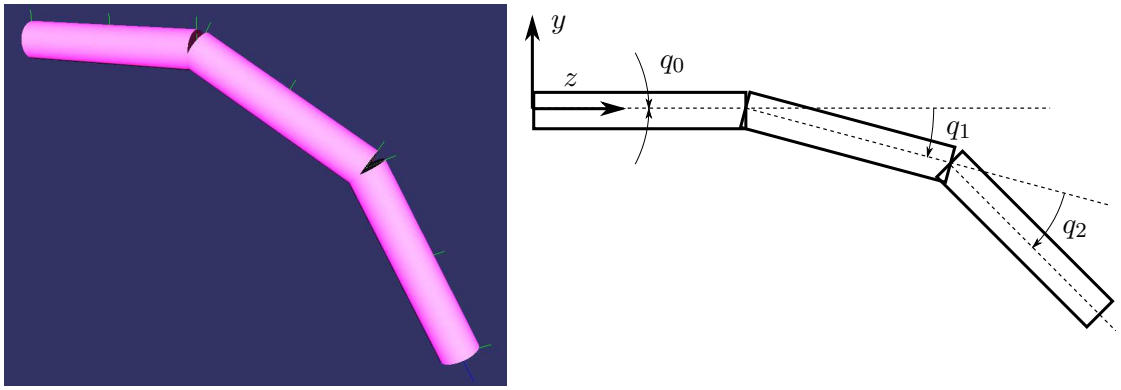


FIGURE 1.8.: 3R robot in Arboris-python (left) and its schematic representation (right) with generalized coordinates q_0, q_1, q_2. All joints rotates about the x-axis.

1.2.5.1. Modeling the system

All the rigid bodies are cylindrical bars of length 0.5 m, radius 0.1 m and mass 10.0 kg. Each body frame (denoted by Ψ_{b0}, Ψ_{b1}, Ψ_{b2}) is located at the bar center of mass. They are all connected at their extremities, and the first one is connected with the ground, as shown in Fig. 1.8.

The robot configuration at time t and the joint limits are summarized in Table (1.1). Joints 0 and 1 are on their maximal limits with positive velocities, so one can expect that some constraints are enabled and must be solved to correctly simulate the behavior of the system.

joint i	$\boldsymbol{q}_i(t)$ (rad)	$\dot{\boldsymbol{q}}_i(t)$ $(rad.s^{-1})$	$\boldsymbol{q}_i$ min (rad)	$\boldsymbol{q}_i$ max (rad)
0	0	0.196	-1.57	0
1	0.524	0.196	-1.57	0.524
2	0.314	0.196	$None$	$None$

TABLE 1.1.: Robot configuration at time t and its joint limits. Note that the initial velocities are not null. The robot has reached some bounds (red).

This configuration leads to the following matrices

$$M_{b(0,1,2)} = \begin{bmatrix} 0.215 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.215 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \quad N_{b0} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.00245 & 0 & 0 & 0 \\ 0 & 0.0421 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.96 \\ 0 & 0 & 0 & 0 & 1.96 & 0 \end{bmatrix}$$

$$N_{b1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.00491 & 0 & 0 & 0 \\ 0 & 0.0843 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -3.93 \\ 0 & 0 & 0 & 0 & 3.93 & 0 \end{bmatrix} \quad N_{b2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.00736 & 0 & 0 & 0 \\ 0 & 0.126 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -5.89 \\ 0 & 0 & 0 & 0 & 5.89 & 0 \end{bmatrix}$$

$$J_{b0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -0.25 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad J_{b1} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -0.683 & -0.25 & 0 \\ 0.25 & 0 & 0 \end{bmatrix} \quad J_{b2} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1.06 & -0.726 & -0.25 \\ 0.526 & 0.155 & 0 \end{bmatrix}$$

$$\dot{J}_{b0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \dot{J}_{b1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.0491 & 0 & 0 \\ 0.085 & 0 & 0 \end{bmatrix} \quad \dot{J}_{b2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.176 & 0.0303 & 0 \\ 0.225 & 0.0934 & 0 \end{bmatrix}$$

Thanks to these matrices, one can achieve the first step of the simulation loop consisting in updating the whole system dynamics, that is the matrices in Eq.(1.39)

$$M = \begin{bmatrix} 20.6 & 10.6 & 2.86 \\ 10.6 & 6.56 & 2.03 \\ 2.86 & 2.03 & 0.84 \end{bmatrix} \quad N = \begin{bmatrix} -0.809 & -1.36 & -0.775 \\ 0.475 & -0.0758 & -0.228 \\ 0.334 & 0.152 & 0 \end{bmatrix} \quad \boldsymbol{g} = \begin{bmatrix} 203 \\ 80.1 \\ 16.4 \end{bmatrix}$$

1.2.5.2. Updating the controllers

The second step retrieves the information from the controllers according to a short period of time $dt = 0.01\ s$.

Here, a proportional derivative controller keeps the system about a desired configuration ($\boldsymbol{q}_{des}$, $\dot{\boldsymbol{q}}_{des}$) thanks to the proportional and derivative gains K_p, K_d defined as follows

$$\boldsymbol{q}_{des} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \dot{\boldsymbol{q}}_{des} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad K_p = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix} \quad K_d = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Generally, this controller generates a generalized force $\boldsymbol{\tau}_c$ based on the current configuration, and the results of this method is shown in Eq.(1.55) (left). In Arboris-python, the possibility is given to change the impedance of the controller to separate the active and proactive parts of the control (cf. Section 1.2.2). It gives the couple ($\boldsymbol{\tau}_{c0}$, Z_c) used in the simulation loop and detailed in Eq.(1.55) (right), and leads to the admittance matrix Y in Eq.(1.56).

$$\boldsymbol{\tau}_c = \begin{bmatrix} -0.393 \\ -5.63 \\ -3.53 \end{bmatrix} \quad \Big\} \quad \Big\{ \quad \boldsymbol{\tau}_{c0} = \begin{bmatrix} 0 \\ -5.24 \\ -3.14 \end{bmatrix} \quad Z_c = \begin{bmatrix} -2.1 & -0 & -0 \\ -0 & -2.1 & -0 \\ -0 & -0 & -2.1 \end{bmatrix} \tag{1.55}$$

$$Y = \begin{bmatrix} 0.00379 & -0.0082 & 0.00672 \\ -0.00821 & 0.0233 & -0.0276 \\ 0.00673 & -0.0277 & 0.0544 \end{bmatrix} \tag{1.56}$$

1.2.5.3. Updating the constraints

In the previous sections, we have described how to compute the equations of motion of the actuated system, but nothing ensures that the joint limits are satisfied. To do so,

we use the Gauss-Seidel algorithm described in Section 1.2.3. The constraint Jacobian and admittance $\overline{J}$ and $\overline{Y}$, as well as the initial values of the reduced vectors $\overline{\mathbf{t}}$ and $\overline{\mathbf{w}}$, are shown in Eq.(1.57).

$$\overline{J} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \overline{Y} = \begin{bmatrix} 0.00379 & -0.0082 \\ -0.00821 & 0.0233 \end{bmatrix} \quad \overline{\mathbf{t}} = \begin{bmatrix} 0.441 \\ -0.0866 \end{bmatrix} \quad \overline{\mathbf{w}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{1.57}$$

Note that the initial value of $\overline{\mathbf{w}}$ may be different from $\mathbf{0}$, for instance the constraint force at the previous simulation loop, to speed up the process. The algorithm modifies the constraint force and velocity until it converges to a stable configuration. This convergence is shown in Fig. 1.9 where the tolerance value is set to $\varepsilon = 1e^{-6}$.

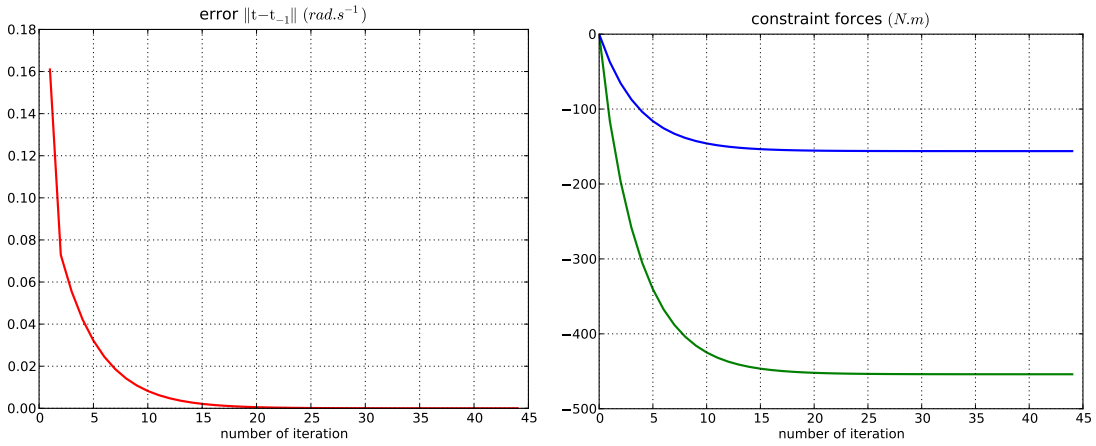


FIGURE 1.9.: Convergence of the Gauss-Seidel algorithm. The left figure shows the evolution of the velocity error which tends asymptotically to 0, and the right figure shows the evolution of the constraint forces which tend to values that ensure the respect of the constraints. The algorithm stops when the error is less than $1e^{-6}$. Note that in a general manner, these graphs should have no units because they may mix linear and rotational components.

Finally, when the process is finished, the constraint force $\overline{\mathbf{w}}$ also converges and the generalized force generated by the latter is obtained such as $\boldsymbol{\tau}_{const} = \sum_k \boldsymbol{\tau}_k = \overline{J}^{\mathsf{T}} \overline{\mathbf{w}}$. The result becomes

$$\overline{\mathbf{w}} = \begin{bmatrix} -454 \\ -156 \end{bmatrix} \qquad \boldsymbol{\tau}_{const} = \begin{bmatrix} -454 \\ -156 \\ 0 \end{bmatrix} \tag{1.58}$$

1.2.5.4. Integrating

The last step of the simulation loop is the computation of the new system configuration. Equation (1.48) sets the new generalized velocity vector, and each joint is updated according to its integration scheme. In this simulation, all joints configurations lie in linear vector spaces, so they follow the integration scheme described in Eq.(1.54). The new configuration is finally obtained in Eq.(1.59), and the simulator becomes ready to

start a new simulation loop.

$$\boldsymbol{q}(t+dt) = \begin{bmatrix} 0 \\ 0.524 \\ 0.329 \end{bmatrix} \qquad \dot{\boldsymbol{q}}(t+dt) = \begin{bmatrix} 0 \\ 0 \\ 1.46 \end{bmatrix} \tag{1.59}$$

1.3. Conclusion

This first chapter presents the mechanical equations to model the global configuration, as well as the kinematic and dynamic equations of constrained rigid-body systems, by taking into account interaction forces with the environment. It results in the equations of motion of mechanical tree-structure. Thus, it follows with the description of the dynamic simulator Arboris-python through the way to model rigid-body systems, and its algorithms to obtain physically consistent motion.

Although the simulator models the kinematic loops as redundant joints and solves their actions on the system iteratively with a Gauss-Seidel algorithm, we prefer to use these kinematic constraints directly in the control loop, as explained in the next chapter.

2. Controlling a mechanical system in interaction with its environment

Contents

This chapter addresses control methods based on optimization programs in the case where robots physically interact with their environment. A particular attention is paid to transitions and control continuity.

Generally, robotic manipulators are controlled with analytic techniques for real-time applications, and a common one is to use inverse kinematics. Jacobians, which give velocities in Cartesian space from velocities in joint space, are inverted so the control is achieved by tracking a Cartesian reference trajectory. Although this control seems quite simple to implement, some difficulties may occur, depending on the robot structure and the task to accomplish. For instance, redundancy has been one of the first problems extensively studied. In this case, robots have more DoF than needed to achieve a task. Jacobian matrices are rectangular, their inverses are not unique, and solutions can be to use the Moore-Penrose pseudo-inverse [Siciliano1990, Tevatia2000]. This inversion technique may lead to inappropriate behaviors when facing singularity problems, typically when manipulators are fully extended, and the desired joint velocities can be very high in the neighborhood of these configurations. Some algorithms are proposed in the literature to cope with this issue, for example the Damped Least Squared (DLS) exposed in [Caccavale1995, Chiaverini1997]. More recently numeric optimization programs have been used, as presented in [Escande2010]. This class of numerical methods allows to select the inversion through a cost function and some constraints, whereas pseudo-inverse is generally a simple norm minimization [1].

However, inverse velocity kinematics is not sufficient when robots have to interact with their environment because the contact forces are not deduced from the computation. To do so, one has to use inverse dynamics where the robot inertial properties are taken into account, by using Eq.(1.39), as exposed in [Kreutz1989, Murray1994]. It offers more possibilities when performing manipulation, grasping, weight compensation, etc. For example, one can perform force control [Park2006] to ensure proper contact forces, one can perform impedance control [Hogan1987, Morel1994] to manage behaviors during the disturbances rejection, or one can perform operational space control to unify motion and force control [Khatib1987].

Independently from these types of control, having redundant systems also means that many tasks can be achieve simultaneously, depending on the number of remaining DoF. The tasks are conflicting in a general manner so control is achieved in two ways, the first one is hierarchical strategy where low-level tasks are projected onto high-level tasks null spaces [Siciliano1991], the second one is weighting strategy where compromise between the tasks is found based on their respective importances [Shen2007]. For instance, Sentis performs several dynamic tasks with humanoid robots using null spaces hierarchy [Sentis2007]. Furthermore, a critical point is the control continuity when actuating mechanical systems. This is done with smooth transitions to ensure that actuators can track the desired control values. For instance, [Padois2005] deals with secondary tasks switching for wheeled mobile manipulators, and [Mansard2007] gives means to achieve smooth transitions in a hierarchical set of tasks while performing kinematic control, a

1. There is also weighted pseudo-inverse to lead the computation toward a particular solution, as presented in [Park2001]

technique reused in [Keith2009] where both the behavior and the overlapping scheduling of sequences of tasks are optimized.

However, most of the works presented above give analytic results to control the robot, and although equality constraints can be integrated in computation, it is not the case for inequality constraints. Some techniques approximate these constraints and create tasks to avoid them, for example with potential field [Khatib1985], but it may disturb the main tasks and may reduce the control range. To properly take into account these inequalities, a solution is to rely on optimization programs. It permits to keep the methods above and to separate what the robot can do, the tasks, and what it undergoes, the equality and inequality constraints. Among all existing optimization programs, we focus on Linear Quadratic Program (LQP), an algorithm widely used in robotics. It can solve some local obstacle avoidance problems [Kanehiro2008], it can achieve humanoid kinematic control to perform several tasks hierarchically using cascading LQPs [Kanoun2009b], and it can optimize the posture of the robot to improve robot balance [Collette2009]. The work of Saab [Saab2011] is very interesting because it presents humanoid dynamic control using cascading LQPs. Note that it is also dedicated to generate physically consistent motions in the computer graphic field [Abe2007, DaSilva2008, Muico2009].

Most of the works using optimization programs for dynamic control do not focus on transitions when there are some changes, both in the set of tasks and the set of constraints. The last ones happen when kinematic loop closures are broken. In this chapter, we focus on the design of a generic whole-body dynamic control based on optimization programs which can deal with these transitions. The multi task management and their sequencing is done with a weighting strategy. First, the robotic constraints are exposed to give the means to choose the most suitable algorithm, then the resolution of the problem is explained, and finally the building of the robotic problem is presented.

2.1. Definition of the problem

This section presents equations that describe robots and their interactions with their environment, which are both the internal and external physical constraints of the system.

2.1.1. Equations of motion with interactions

Humanoid robots can be modeled as underactuated tree-structures composed of rigid bodies linked together with revolute joints. Thus the evolution of such a system is described by the equations of motion Eq.(1.39) defined in the previous chapter. It derives from the Euler-Lagrange formalism and relates the generalized acceleration with the generalized forces generated by the actuators and the external wrenches. The motion depends on variables belonging on either the acceleration space or the force space, and this can be viewed as a function with the forces as input and the accelerations as output. Hence, the equation is rewritten as follows

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{g}(\boldsymbol{q}) + J_{\chi}(\boldsymbol{q})^{\mathsf{T}}\boldsymbol{\chi}. \tag{2.1}$$

where we recall that $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ are respectively the generalized coordinates, velocity and acceleration vectors, $M(\boldsymbol{q}), \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}), \boldsymbol{g}(\boldsymbol{q})$ are the generalized inertia matrix, the nonlinear effects vector (Coriolis and centrifugal, such as $\boldsymbol{n} = N\dot{\boldsymbol{q}}$ with N from Eq.(1.39)), and the gravity force vector. The last term of the equation regroups the generalized

forces applied to the system. The matrix denoted by $J_\chi(\boldsymbol{q})^\mathsf{T} = \begin{bmatrix} J_c(\boldsymbol{q})^\mathsf{T} & S \end{bmatrix}$ is called the generalized wrench Jacobian and $\boldsymbol{\chi} = \begin{bmatrix} \mathbf{w}_c^\mathsf{T} & \boldsymbol{\tau}^\mathsf{T} \end{bmatrix}^\mathsf{T}$ is called the **action variable** of the system. $J_c(\boldsymbol{q})$ is the concatenation of the n_c Jacobians of contacts denoted by $J_{c_i}(\boldsymbol{q})$ $(i \in [1..n_c])$, $\mathbf{w}_c$ is the concatenation of the associated external wrenches $\mathbf{w}_{c_i}$ $(i \in [1..n_c])$ and $J_c(\boldsymbol{q})^\mathsf{T}\mathbf{w}_c = \sum_{i=1}^{n_c} J_{c_i}(\boldsymbol{q})^\mathsf{T}\mathbf{w}_{c_i}$ is the torque induced by the wrenches applied to the system at the n_c contact points [2]. Finally, S and $\boldsymbol{\tau}$ are respectively the actuation selection matrix, which accounts for the fact that some DoF are not actuated (typically the free-floating base), and the input torque vector applied at the actuators level.

Although Eq.(2.1) is nonlinear, it is generally linearized around the state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ for small variations, and then the matrices $M, \boldsymbol{n}, \boldsymbol{g}, J_\chi$ are supposed to be known. It defines the surjective mapping from the joint torque and contact forces to the joint acceleration spaces, so $\boldsymbol{\chi}$ is sufficient to describe the dynamics of the system and $\ddot{\boldsymbol{q}}$ is considered as an overabundant variable.

However, the following expressions which describe the dynamic constraints can only depend on $\ddot{\boldsymbol{q}}$, and their expressions in the wrench space may make them less readable. Hence, for the sake of clarity, $\mathbb{X} = \begin{bmatrix} \ddot{\boldsymbol{q}}^\mathsf{T} & \boldsymbol{\chi}^\mathsf{T} \end{bmatrix}^\mathsf{T}$, called the **dynamic variable** of the system for the remaining of this thesis, is introduced to define the dynamic equations in the generic form $A(\boldsymbol{q})\mathbb{X} \leqq \boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}})$. The transformation from the overabundant formulation to the independent one exists and is discussed later.

2.1.2. Actuation limits

This section presents the actuation limits of the robot. Plenty of actuators are available in the robotic field with different types, sizes, capacities, etc. It is obvious that real mechanical systems have not unbounded capabilities, and for the sakes of safety, durability and efficiency, constructors may restrain actuators capacities. The following of this section addresses the expression and integration of these constraints.

2.1.2.1. Torque limit

Here, we consider a particular range of actuators that can achieve torque tracking [3]. As they are real mechanical systems, their output is bounded, so given $\boldsymbol{\tau}_{max}$ the maximum torque vector of the robot actuators, the torque limit is expressed as follows

$$-\boldsymbol{\tau}_{max} \preceq \boldsymbol{\tau} \preceq \boldsymbol{\tau}_{max} \quad \Leftrightarrow \quad \begin{bmatrix} 0 & S_\tau \\ 0 & -S_\tau \end{bmatrix} \mathbb{X} \preceq \begin{bmatrix} \boldsymbol{\tau}_{max} \\ \boldsymbol{\tau}_{max} \end{bmatrix} \tag{2.2}$$

where the selection matrix S_τ is defined such as $\boldsymbol{\tau} = S_\tau \boldsymbol{\chi}$.

2.1.2.2. Acceleration limit

As the actuation is bounded, the equations of motion imply that the generalized acceleration is also bounded, and Eq.(2.2) should be sufficient to limit the evolution of

2. $\mathbf{w}_{c_i}$ represents the image set of the wrench $\boldsymbol{W}^{c_i}$. For example, when a contact occurs, some torques are not transmitted, and $\mathbf{w}_{c_i}$ is the remaining contact efforts. The Jacobian J_{c_i} is then chosen accordingly.

3. We only consider torques because humanoid robots are mainly composed of revolute joints. Of course, one has to refer to forces if linear actuators are used, and this study remains valid.

the system in space. However, for some technical reasons, one may want to set his own limits in acceleration. Hence, given the boundaries $\ddot{\boldsymbol{q}}_{min} \preceq \mathbf{0},\ \ddot{\boldsymbol{q}}_{max} \succeq \mathbf{0}$, the acceleration limit is expressed as

$$\ddot{\boldsymbol{q}}_{min} \preceq \ddot{\boldsymbol{q}} \preceq \ddot{\boldsymbol{q}}_{max} \quad \Leftrightarrow \quad \begin{bmatrix} I & 0 \\ -I & 0 \end{bmatrix} \mathbb{X} \preceq \begin{bmatrix} \ddot{\boldsymbol{q}}_{max} \\ -\ddot{\boldsymbol{q}}_{min} \end{bmatrix} \tag{2.3}$$

where I is the identity matrix.

2.1.2.3. Velocity limit

Similarly to acceleration, one may want to limit the generalized velocity of the system. Theoretically, given the boundaries $\dot{\boldsymbol{q}}_{min} \preceq \mathbf{0},\ \dot{\boldsymbol{q}}_{max} \succeq \mathbf{0}$, the joint velocity limit is expressed as $\dot{\boldsymbol{q}}_{min} \preceq \dot{\boldsymbol{q}} \preceq \dot{\boldsymbol{q}}_{max}$. However this formulation does not depend on the dynamic variable of the system. This issue is very difficult to handle because there is no exact linear relation between $\dot{\boldsymbol{q}}$ and $\mathbb{X}$. Here, we propose to constrain the future generalized velocity assuming a uniformly accelerated motion of the joints, which leads to a linear relation between accelerations and velocities. Thus, the velocity limit constraint depending on the dynamic variable is written as follows

$$\dot{\boldsymbol{q}}_{min} \preceq \dot{\boldsymbol{q}} + h_1\ddot{\boldsymbol{q}} \preceq \dot{\boldsymbol{q}}_{max} \quad \Leftrightarrow \quad \begin{bmatrix} I & 0 \\ -I & 0 \end{bmatrix} \mathbb{X} \preceq \frac{1}{h_1} \begin{bmatrix} \dot{\boldsymbol{q}}_{max} - \dot{\boldsymbol{q}} \\ -(\dot{\boldsymbol{q}}_{min} - \dot{\boldsymbol{q}}) \end{bmatrix} \tag{2.4}$$

where h_1 is an anticipation coefficient set to compute the estimated values of the future state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, assuming a constant generalized acceleration $\ddot{\boldsymbol{q}}$.

2.1.2.4. Joint limit

Finally, the generalized coordinates of the robot can also be bounded. The constraint can be written $\boldsymbol{q}_{min} \preceq \boldsymbol{q} \preceq \boldsymbol{q}_{max}$, but for the same reasons as above we constrain the future coordinates, so given the boundaries $\boldsymbol{q}_{min}, \boldsymbol{q}_{max}$, the joint position limit is expressed as follows

$$\boldsymbol{q}_{min} \preceq \boldsymbol{q} + h_2\dot{\boldsymbol{q}} + \frac{(h_2)^2}{2}\ddot{\boldsymbol{q}} \preceq \boldsymbol{q}_{max} \Leftrightarrow \quad \begin{bmatrix} I & 0 \\ -I & 0 \end{bmatrix} \mathbb{X} \preceq \frac{2}{(h_2)^2} \begin{bmatrix} \boldsymbol{q}_{max} - (\boldsymbol{q} + h_2\dot{\boldsymbol{q}}) \\ -(\boldsymbol{q}_{min} - (\boldsymbol{q} + h_2\dot{\boldsymbol{q}})) \end{bmatrix} \tag{2.5}$$

where h_2 plays a role similar to h_1.

In order to avoid overabundant constraints, Eqs.(2.3)–(2.5) can be concatenated into a unique inequality as written below

$$\begin{bmatrix} I & 0 \\ -I & 0 \end{bmatrix} \mathbb{X} \preceq \begin{bmatrix} \min\left(\ddot{\boldsymbol{q}}_{max}, \dfrac{\dot{\boldsymbol{q}}_{max} - \dot{\boldsymbol{q}}}{h_1}, \dfrac{2(\boldsymbol{q}_{max} - (\boldsymbol{q} + h_2\dot{\boldsymbol{q}}))}{(h_2)^2}\right) \\ -\max\left(\ddot{\boldsymbol{q}}_{min}, \dfrac{\dot{\boldsymbol{q}}_{min} - \dot{\boldsymbol{q}}}{h_1}, \dfrac{2(\boldsymbol{q}_{min} - (\boldsymbol{q} + h_2\dot{\boldsymbol{q}}))}{(h_2)^2}\right) \end{bmatrix}. \tag{2.6}$$

where the min and max functions select the best values term-by-term, meaning that the more restrictive constraints are selected for each DoF.

2.1.3. Obstacle avoidance

Collisions are also physical robotic limitations in any environment. Indeed, working with a human in the same workspace implies limited powers, less rigid structures, but if collisions occur they may lead to great damages, so sudden impact should be prevented. Furthermore, if some unexpected obstacles block the achievement of some tasks, it would be interesting to reactively take into account these spatial constraints to avoid collision, or at least to control the contact. Finally, for the same reasons, a very challenging problem is about the self-collision. This can be easily illustrated, for example if a humanoid robot falls due to some disturbances and has to reactively recover balance by displacing its feet, then the moving leg must have a path which does not go through the supporting leg.

Many techniques have been developed in robotics to avoid obstacles. For instance, a fast and reliable technique is to use an artificial potential field approach [Khatib1985], to create virtual repulsive forces when manipulators get too close to the obstacles. These potential fields are built from convex shapes surrounding obstacles. They allow real-time applications and reactive avoidance control. But as a result, this technique reduces the working space independently of the robot velocity and may adversely affect mission achievement, or it may lead to oscillating behaviors in very constrained environments. An interesting data called the *collidability* measure is introduced in the following paper [Choi2000] which gives some knowledge about possible future collisions by taking into account the linear and rotational object velocities in the computation of this measure. However, it is done with ellipsoid and spherical bounding volumes, which again reduces the working space.

The paper of [Maciejewski1985] proposes a two-levels control where the principal goal is the main task (for example path tracking) and the secondary goal is the obstacle avoidance using null space projector. This gives more space to the manipulator when environments are complex and are composed of several obstacles. The same idea is proposed in [Sentis2005], where obstacle avoidance is set as the principal goal in a control scheme based on recursive null space projections. Besides, [Stasse2008] deals with collision and auto-collision for humanoid robots by using avoidance tasks on convex bounding volumes, and a particular attention is paid to control continuity. However, avoidances are considered as tasks in these methods and not as inequality constraints.

In [Faverjon1987], collision avoidances are considered as constraints in a LQP. This proposition allows to extend the available workspace to the maximum (generally with some safety margins), and it takes into account the velocity of the system. In addition, this work presents an *assembly tree* that approximates the robot shape with unions of convex bounds, whose number and precision increase in the subtrees. For efficient distance computation between very complex meshes (even non-convex), the reader can refer to [Merlhiot2009].

All these solutions are adapted to local procedures and cannot be applied in more complex tasks, for example when manipulators have to reach some goals in highly cluttered spaces. The global approach resolution is more complicated and is addressed in other works where off-line path planning is proposed [LaValle2006]. As the control developed in this work must be reactive, we focus on local techniques that can be used in optimization programs.

2.1.3.1. Constraint expression

To prevent any collision, obstacle avoidance constraints must be set between several couples of shapes, as proposed in [Faverjon1987], but here in a dynamic framework. One needs to find the vector $\boldsymbol{\Delta}_{i,j}$ which supports the minimal distance $d_{i,j}$ between the shapes i and j, and to maintain it positive $d_{i,j} > 0$. Again, this is a position constraint, and the link with the dynamic parameters of the system is not straightforward without any assumptions or approximations. Similarly to the position and velocity limits, we consider a uniformly accelerated motion of the points which determine the minimal distance and keep it positive.

This statement is very strong, firstly because the acceleration is not constant, and secondly the points which define the minimal distance run on the shapes, and are consequently not constant. However using smooth shapes and low speeds, the points which define the future minimal distance will be located in the neighborhoods of the current points, and this assumption can be considered valid.

Hence, the constraint can be expressed in terms of the dynamic variable. The Jacobian of collision avoidance between the shapes i and j is defined as the mapping from the generalized velocity to their relative velocities projected onto axis $\boldsymbol{\Delta}_{i,j}$ such as $J_{i,j}(\boldsymbol{q}) = J_j(\boldsymbol{q}) - J_i(\boldsymbol{q})$, and its time-derivative is defined such as $\dot{J}_{i,j}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{J}_j(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \dot{J}_i(\boldsymbol{q}, \dot{\boldsymbol{q}})$. Usually, one wants to perform many obstacle avoidances in the same time, and given $\boldsymbol{d}, \dot{\boldsymbol{d}}, J_a, \dot{J}_a$ the concatenation of respectively the relative distances, velocities, and Jacobians, the obstacle avoidance constraint becomes

$$\begin{aligned} \mathbf{0} &\preceq \boldsymbol{d} + h_3 \dot{\boldsymbol{d}} + \frac{(h_3)^2}{2} \left(J_a(\boldsymbol{q}) \ddot{\boldsymbol{q}} + \dot{J}_a(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} \right) \\ \Leftrightarrow -\frac{(h_3)^2}{2} &\begin{bmatrix} J_a(\boldsymbol{q}) & 0 \end{bmatrix} \mathbb{X} \preceq \boldsymbol{d} + h_3 \dot{\boldsymbol{d}} + \frac{(h_3)^2}{2} \dot{J}_a(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} \end{aligned} \tag{2.7}$$

where h_3 plays a role similar to h_1 and h_2. Note that this equation can also be used for self-collision avoidance.

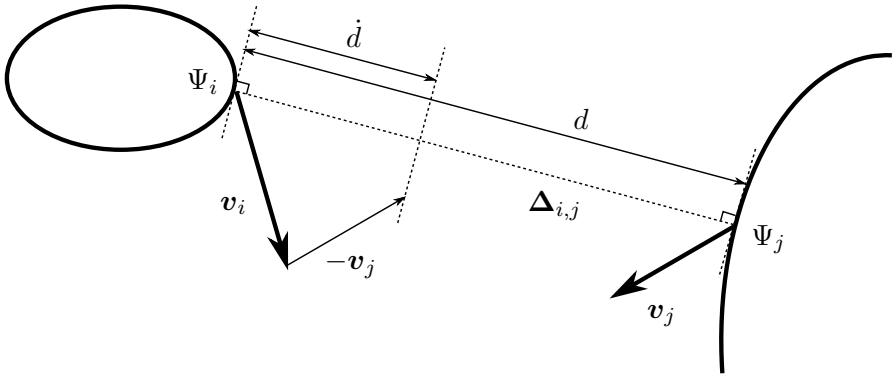


FIGURE 2.1.: Description of the obstacle avoidance problem : the objective is to obtain $\boldsymbol{d}$ and $\dot{\boldsymbol{d}}$, which are the relative distance and velocity between Ψ_i and Ψ_j.

2.1.4. Interaction with the environment

Whereas the section above focuses on the modeling of a constraint which prevents any collision, this one deals with the interaction between the robot and the environment

to achieve the objectives defined by the user. As humanoid robots are under-actuated systems, meaning that they cannot actuate all their DoF (their location and orientation in space), it must interact with the surrounding world for its activities, starting with its locomotion. To do so, the system creates some kinematic loops and loses its tree-structure property. Like the joints, there is many different type of interactions, which can be unilateral or bilateral, time-dependent, holonomic or not, etc. In this section, we focus on two type of interactions, the bilateral closure and the unilateral frictional contact.

2.1.4.1. Bilateral Closure

Bilateral closures constrain the relative motion between two frames of the system with bilateral actions. It means that interaction forces do not evolve in half-spaces, as it is the case with contact points where there is no attractive force. In fact, it is a joint which cannot be integrated in the equations of motion Eq.(1.39) (because it would be considered as redundant) and must be considered as a constraint. But unlike the joint, this closure is expressed through the fixed part of the link, that is the subspace with no relative motion.

For example, if a ball and socket joint is added to create a bilateral closure between frames Ψ_i and Ψ_j, it means that there is no relative translational motion which leads to $\boldsymbol{v}_i - \boldsymbol{v}_j = \mathbf{0}$. Denoting by J_i, J_j and $\dot{J}_i, \dot{J}_j$ the frames Jacobians and their derivatives composed of their translational parts, the equation can be expressed as follows

$$\begin{aligned} &\big(J_i(\boldsymbol{q}) - J_j(\boldsymbol{q})\big)\ddot{\boldsymbol{q}} + \big(\dot{J}_i(\boldsymbol{q},\dot{\boldsymbol{q}}) - \dot{J}_j(\boldsymbol{q},\dot{\boldsymbol{q}})\big)\dot{\boldsymbol{q}} = \mathbf{0} \\ \Leftrightarrow &\begin{bmatrix} J_i(\boldsymbol{q}) - J_j(\boldsymbol{q}) & 0 \end{bmatrix} \mathbb{X} = -\big(\dot{J}_i(\boldsymbol{q},\dot{\boldsymbol{q}}) - \dot{J}_j(\boldsymbol{q},\dot{\boldsymbol{q}})\big)\dot{\boldsymbol{q}}. \end{aligned} \tag{2.8}$$

Now, if this bilateral closure is created with a hinge joint, it is sufficient to extend the Jacobians with their blocked rotational parts.

2.1.4.2. Unilateral frictional contact

Similarly to bilateral closures, unilateral frictional contacts create kinematic-loops and relate two frames by constraining their relative velocity. But the difference lies in the fact that interaction forces evolve in half-spaces, *i.e.* there are only repulsive forces and no attractive force. Besides, when the contact appears, the velocity goes from a non-negative value to 0 very quickly, and a reaction force is created to avoid interpenetration. The contact point can remain at the same location, but it can also slide on the contact surfaces with some frictional forces. One refers to open and closed contacts whether some forces are transmitted or not, and due to these binary states, some problems about nonlinearity may occur.

The control of robot with unilateral contacts is not easy to handle [Park2006], it depends on the stiffness of the materials, their roughness, the dissipation of the mechanical energy during impacts, etc. However, a simple approximation of the contact comes from the Amontons' law and the use of the Coulomb cone of friction. This modeling is sufficient for the rest of this thesis, but one may refer to more complicated ones, for example a "soft finger contact" which integrates a frictional torque about the normal axis as used in [Trinkle2001, Liu2005].

Note that the resolution of several frictional contacts in multibody systems simulators is usually performed through Linear Complementary Problem (LCP) [Stewart1996,

Anitescu1997]. Here, we only consider the modeling of unilateral frictional contact, the resolution being left to another optimization program.

Constraint expression The expression of the contact is based on the modeling of dry friction. Considering the i^{th} contact, it depends on two variables, one describing its velocity $\boldsymbol{v}_{c_i}$ and the other one describing its force $\mathbf{w}_{c_i}$, the same as that defined in Section 2.1.1. Each one has a related Coulomb friction cone which relates the normal component of the force with the tangential component through the frictional coefficient μ_i. Then given the normal vector of contact $\boldsymbol{n}$, the equation becomes

$$\|\mathbf{w}_{c_i} - (\mathbf{w}_{c_i} \cdot \boldsymbol{n})\boldsymbol{n}\| \leq \mu_i(\mathbf{w}_{c_i} \cdot \boldsymbol{n}). \tag{2.9}$$

This friction cone model can be formulated as a Linear Matrix Inequality (LMI), and [Han2000] gives very good explanations on this formulation and the related optimization programs to find the contact forces.

When a simpler model is sufficient, one can use a less precise model reduced to a linear approximation of the Coulomb cone, and the equation above can be written in a matrix form such as $C_{c_i}\mathbf{w}_{c_i} \preceq \mathbf{0}$ where C_{c_i} represents the linearized cone, as shown in Fig. 2.2.

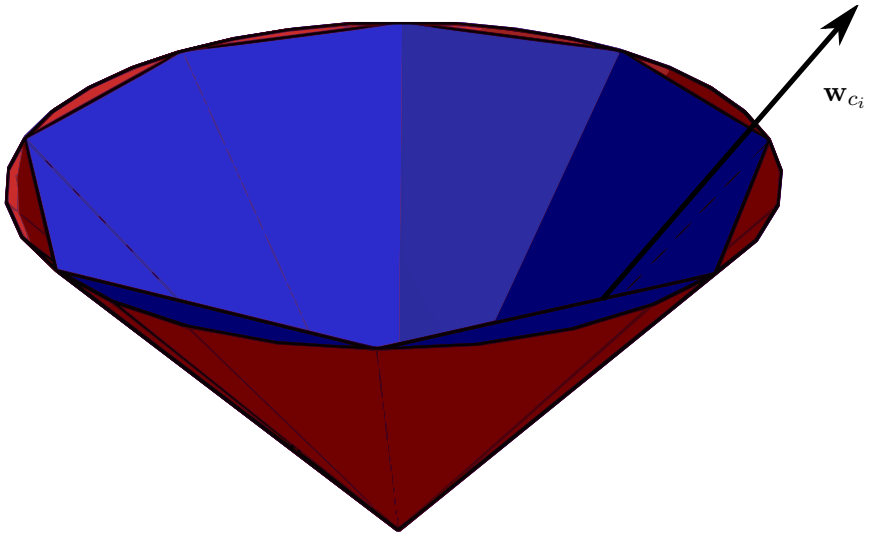


FIGURE 2.2.: Coulomb cone of friction (red) and its linear approximation (blue). The force $\mathbf{w}_{c_i}$ must stay inside to avoid sliding.

As previously stated, a frictional contact has different discrete states[4]. First, the contact is established, leading to a null velocity and a force lying in the friction cone, so Eq.(2.9) is valid and $\boldsymbol{v} = \mathbf{0}$. Second, there is no contact, meaning that the motion of the points is not constrained and the force is null, so $\mathbf{w}_{c_i} = \mathbf{0}$. These cases have to be expressed in terms of the dynamic variable. Recalling the Jacobian $J_{c_i}(\boldsymbol{q})$, composed of the translational part only, relates $\dot{\boldsymbol{q}}$ to the linear velocity $\boldsymbol{v}_{c_i}$ such as $\boldsymbol{v}_{c_i} = J_{c_i}(\boldsymbol{q})\dot{\boldsymbol{q}}$, and

4. Here, we present only two cases. There is also a transitional case when the contact is lifting requiring a positive velocity along the normal, and a sliding case when the contact is closed and the force lies outside the cone, requiring a null velocity along the normal. However, these two cases are not discussed in this thesis because they are useful only in complex situations that are not relevant in this work.

defining the selection matrix $S_{\mathrm{w}_{c_i}}$ such as $\mathbf{w}_{c_i} = S_{\mathrm{w}_{c_i}} \boldsymbol{\chi}$, each closed contact constraint can be expressed with the linearized friction cone as follows

$$\left\{ \begin{array}{lcl} J_{c_i}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{J}_{c_i}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} & = & \mathbf{0} \\ C_{c_i}\mathbf{w}_{c_i} & \preceq & \mathbf{0} \end{array} \right. \Leftrightarrow \left\{ \begin{array}{lcl} \left[\begin{array}{cc} J_{c_i}(\boldsymbol{q}) & 0 \end{array} \right] \mathbb{X} & = & -\dot{J}_{c_i}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \\ \left[\begin{array}{cc} 0 & C_{c_i} S_{\mathrm{w}_{c_i}} \end{array} \right] \mathbb{X} & \preceq & \mathbf{0} \end{array} \right. . \tag{2.10}$$

Finally, the trivial case of the open contact (no interaction force) is expressed according to $\mathbb{X}$ as $\left[\begin{array}{cc} 0 & S_{\mathrm{w}_{c_i}} \end{array} \right] \mathbb{X} = \mathbf{0}$.

2.2. Solving convex problems

Although many optimization programs exist, all are not suitable for a particular problem. Of course, the choice of the program depends upon the type of cost function and constraints, but one can modify the level of detail of the modeling to use faster algorithms. This is useful for real-time applications. Generally, the input parameters of robotic low-level controllers belong to the continuous domain, so it is possible to build a continuous cost function to control the robot, and the resolution of such a problem, constrained or not, relies mainly on its convexity. If it is not convex, the most common approach is to use a branch-and-bound method. Otherwise, when the problem is convex, it refers to the convex optimization field, which contains lots of interesting subproblems as the Linear Problem (LP), the Linear Quadratic Problem (LQP), the Second-Order Cone Program (SOCP), etc. To go deeper into this field, the reader can refer to the following books [Bertsekas2003, Boyd2004].

Convex optimization is a tool largely used in robotics. for instance [Barthelemy2008] employs a LP to specify a residual radius which formalizes the robustness of virtual human dynamic equilibrium by taking into account the forces lying into the Coulomb friction cone. The set is defined by a polyhedron and the residual radius is computed as the solution of the LP which solves the following problem :

$$\begin{array}{ll} \text{minimize} & \boldsymbol{f}^{\mathsf{T}}\boldsymbol{x} \\ \text{subject to} & G\boldsymbol{x} \preceq \boldsymbol{h} \\ & A\boldsymbol{x} = \boldsymbol{b} \end{array}$$

where $\boldsymbol{f}^{\mathsf{T}}\boldsymbol{x}$ is the linear cost function which determines the balance state, the couples $(A, \boldsymbol{b})$ & $(G, \boldsymbol{h})$ are respectively the linear equality and inequality constraints.

Furthermore, [Park2007] considers the problem of active balancing for legged robots through a SOCP, which is written as follows

$$\begin{array}{ll} \text{minimize} & \boldsymbol{f}^{\mathsf{T}}\boldsymbol{x} \\ \text{subject to} & \|G_i\boldsymbol{x} + \boldsymbol{h}_i\|_2 \leq \boldsymbol{c}_i^{\mathsf{T}}\boldsymbol{x} + d_i \quad \forall i \\ & A\boldsymbol{x} = \boldsymbol{b} \end{array}$$

where $\boldsymbol{f}^{\mathsf{T}}\boldsymbol{x}$ defines the balance state, the couple $(A, \boldsymbol{b})$ is the linear equality constraint, and the tuples $(G_i, \boldsymbol{h}_i, \boldsymbol{c}_i, d_i)\forall i$ represent the conic inequality constraints.

Finally, the whole-body kinematic control of humanoid robots has been explored in [Kanoun2009a, Kanoun2009b] (where the achievement of a hierarchical set of possibly conflicting tasks is solved with cascading LQPs) and in [Saab2011]. The latter propose

humanoid dynamic control using cascading LQPs to perform model-based motions (from algorithms using equations of motion) or data-driven motions (from human motion capture).

Although computer graphics and animation are not part of the robotics field, they share the same algorithms in order to obtain physically consistent motions when the elements of the scene have to interact with each other, are subject to gravity and are subject to collisions. This is done with optimization as in [Muico2009], and especially with LQP [Abe2007, DaSilva2008].

Notice that one may face huge constraints sets while performing robotic control with these methods, especially many inequality constraints which may be larger than the number of DoF of the system. That may lead to incompatibilities in the constraint set and return no available solution, but it is inconceivable to let the robot uncontrolled in these situations. This issue is not addressed in this thesis, but the reader may refer to [Rubrecht2010a, Rubrecht2010b] which ensure the management of inequality problems for highly redundant robots in over-constrained spaces.

This work focuses on the resolution of convex problems, unconstrained, constrained with equalities, and constrained with inequalities, with a particular focus on LQP.

2.2.1. Resolution of unconstrained convex problem

A function $f : \mathbb{R}^n \to \mathbb{R}$ is called convex if for all vectors $\boldsymbol{a}, \boldsymbol{b}$ and all real numbers $t \in [0, 1]$, the relation $f(t\boldsymbol{a}+(1-t)\boldsymbol{b}) \leq tf(\boldsymbol{a})+(1-t)f(\boldsymbol{b})$ is satisfied. These functions are particularly interesting because all their local minima are also global, which simplifies the resolution of the minimization problem. An unconstrained convex problem is written as follows

$$\text{minimize} \quad f(\boldsymbol{x}) \tag{2.11}$$

where $\boldsymbol{x}$ is the unknown vector and f is a convex twice continuously differentiable function [5]. Thanks to convexity, the research of local minima (with fast and efficient methods) leads to a global solution. The most common algorithm for this purpose is the gradient descent which is described hereafter.

2.2.1.1. Gradient descent

Given a starting point, this algorithm seeks for a local minimum according to the path with the steepest descent. This can be illustrated by a ball subject to the gravity which falls into a bowl representing the convex function, and finishes its path at the lowest point, as shown in Fig. 2.3. The algorithm is described as follows

This procedure is very simple and when the gradient $\nabla(f(\boldsymbol{x}))$ of the function f at point $\boldsymbol{x}$ is inferior to the tolerance ε, the actual position $\boldsymbol{x}$ is returned, *i.e.* one considers that it is very close to the lowest point of the convex function. Unfortunately, this method implies that the solution is close to the optimum **according to an error**. So, in the presence of convex function with very flat "bottom", it may lead to solutions far from real optimum.

Besides, the computation of the step t is quite difficult. A low value may slow down the search of the optimum and increase the time, and a high value may involve oscillating behaviors near the solution, as shown in Fig. 2.4.

5. The minimization of f is also the maximization of its opposite $-f$.

Algorithm 2: Gradient Descent.

Initialize $\boldsymbol{x} \longleftarrow \boldsymbol{x}_0$, choose ε (the tolerance)
repeat
  compute the gradient descent direction $-\nabla(f(\boldsymbol{x}))$
  compute the step $t > 0$
  update : $\boldsymbol{x} \longleftarrow \boldsymbol{x} - t\nabla(f(\boldsymbol{x}))$
until $\nabla(f(\boldsymbol{x})) \leq \varepsilon$
return $\boldsymbol{x}$.

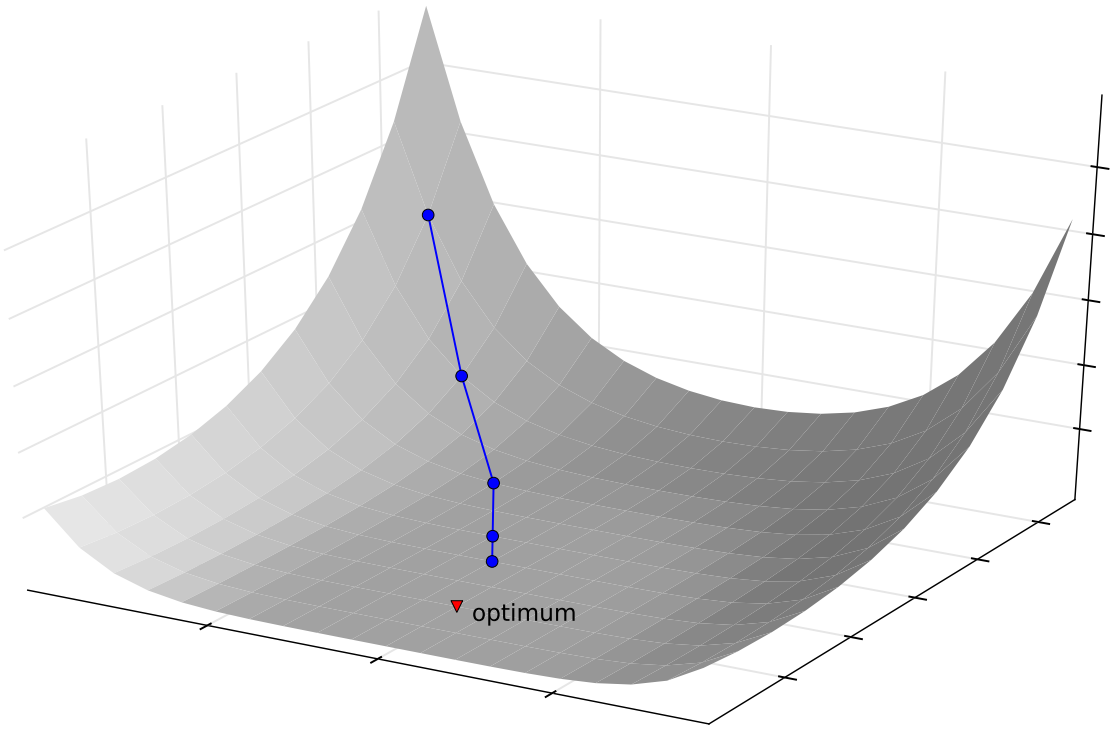


FIGURE 2.3.: Illustration of the gradient descent algorithm. The first iterations go down quickly toward the optimal value, and slow down when they get closer to the solution.

The selection of t can be addressed by some dedicated algorithms, for example the backtracking linesearch, which initializes it to a high value and decreases it as the method gets closer to the solution, *i.e.* the descent becomes less steep.

2.2.1.2. Special case : the quadratic problem

A quadratic function, meaning that it can be written as a homogeneous polynomial of degree two, is a particular convex function whose minimum can be computed analytically. In this case, the function to optimize is written in a general form as follows

$$\text{minimize} \quad f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathsf{T}} P \boldsymbol{x} + \boldsymbol{q}^{\mathsf{T}} \boldsymbol{x} + r. \tag{2.12}$$

and the optimality condition is obtained through the computation of its derivative which must be null, so $P\boldsymbol{x}^* + \boldsymbol{q} = \boldsymbol{0}$ with $\boldsymbol{x}^*$ is the optimal solution. It leads to two cases

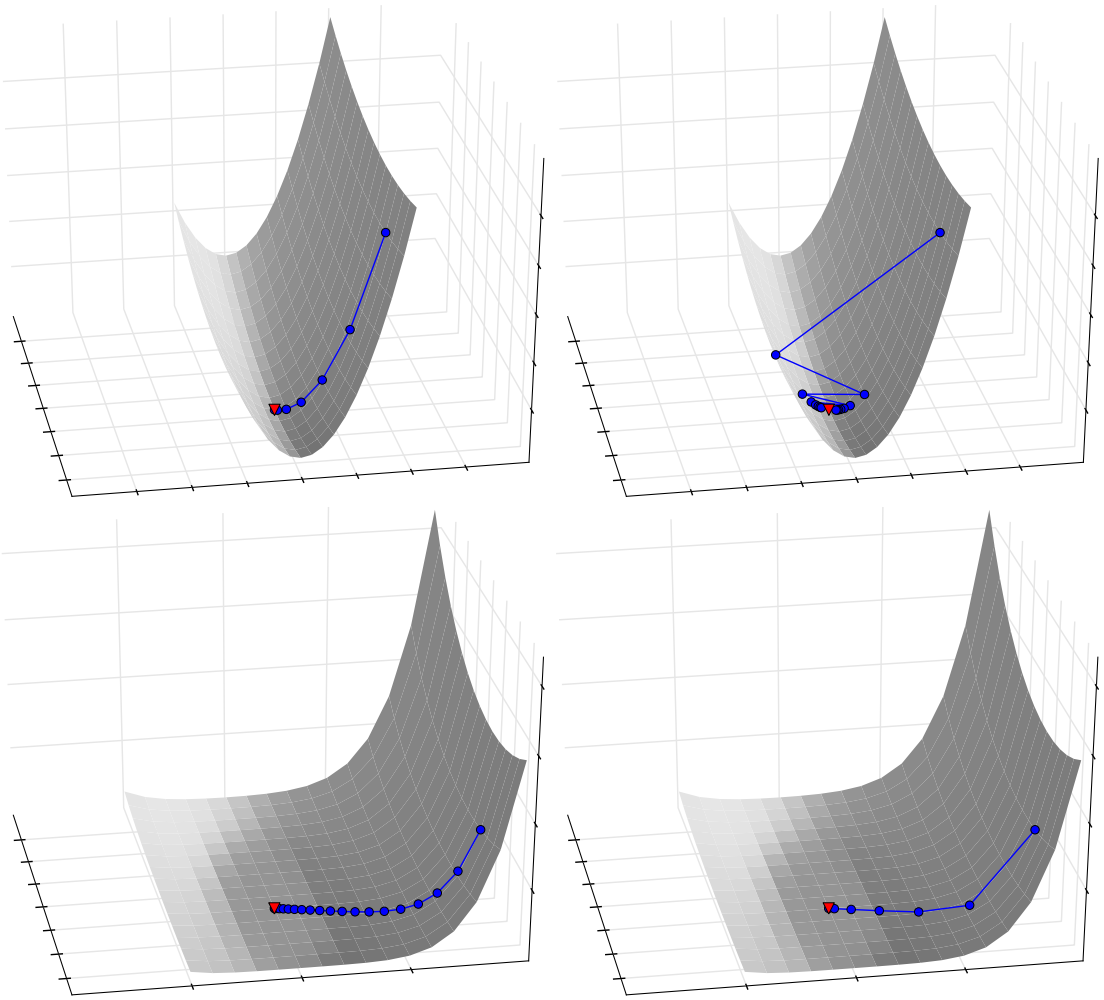


FIGURE 2.4.: Comparison between different values of parameter t. low values are very effective on steep descents (upper left) but take more iterations on "flat area" (lower left). In contrast, the search may oscillate on steep descents with high values (upper right) but optimal value is recovered with less iterations on flat area (lower right).

- the matrix P is invertible and the optimal solution is computed as $\boldsymbol{x}^* = -P^{-1}\boldsymbol{q}$,
- the matrix P is not invertible, so all $\boldsymbol{x}^*$ which satisfy $P\boldsymbol{x}^* + \boldsymbol{q} = \mathbf{0}$ are solutions, if any.

Alternatively, the quadratic program can be written in a different manner as follows

$$\text{minimize} \quad \|E\boldsymbol{x} - \mathbf{f}\|_2^2 = \boldsymbol{x}^\mathsf{T}(E^\mathsf{T}E)\boldsymbol{x} - 2(E^\mathsf{T}\mathbf{f})^\mathsf{T}\boldsymbol{x} + \mathbf{f}^\mathsf{T}\mathbf{f}. \tag{2.13}$$

This formulation is quite similar to the previous one and the transformation from one to the other one is straightforward. The optimal condition of this problem becomes $E^\mathsf{T}E\boldsymbol{x}^* = E^\mathsf{T}\mathbf{f}$, and the solution relies on the invertibility of the matrix $E^\mathsf{T}E$. One possibility is to express it in terms of $E^\dagger$, the pseudo-inverse of Moore-Penrose of E [Albert1972], such as $\boldsymbol{x}^* = E^\dagger\mathbf{f}$, and given a reference vector $\boldsymbol{z}_0$, the solution becomes

$$\boldsymbol{x}^* = E^\dagger\mathbf{f} + (I - E^\dagger E)\boldsymbol{z}_0 \tag{2.14}$$

where $\boldsymbol{z}_0$ is any vector projected onto $(I - E^\dagger E)$ the null space of E, and consequently does not interfere with the optimality condition.

2.2.2. Resolution of convex problem with equality constraints

This section considers the convex minimization problem subject to linear equality constraints, expressed as follows

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to} && A\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{2.15}$$

The dimension of the problem and the constraints are respectively denoted by n and p such as $\boldsymbol{x} \in \mathbb{R}^n$, $A \in \mathbb{R}^{p \times n}$, $\boldsymbol{b} \in \mathbb{R}^p$ with the assumptions that the matrix A is full rank and that $p < n$. If the first statement is false, it means that some constraints are expressed many times in the set of equalities, therefore it is mandatory to extract the full rank subsystem [6]. Furthermore, if the second statement is false, the problem is then over-constrained and cannot be optimized.

Subject to these assumptions, the solution derives from the Karush-Khun-Tucker conditions (KKT) [Boyd2004] which are always necessary for optimality, but also sufficient for convex problems due to the fact that all minima are global. Thus, the computation of the solution $\boldsymbol{x}^*$ is related to a vector $\boldsymbol{y}^*$, the solution of the dual problem, such as

$$\begin{cases} A\boldsymbol{x}^* = \boldsymbol{b} \\ \nabla f(\boldsymbol{x}^*) + A^\mathsf{T}\boldsymbol{y}^* = \boldsymbol{0} \end{cases}$$

where ∇ is the vector differential operator. The first line, called the **primal feasibility equations**, is a linear expression with respect to the primal solution vector $\boldsymbol{x}^*$, and the second line, called the **dual feasibility equations**, is generally nonlinear with respect to $\boldsymbol{x}$ and depends on the dual solution vector $\boldsymbol{y}^*$.

The resolution of this problem can be tackled in two ways. The first one is to build the unconstrained dual problem and to recover the solution $\boldsymbol{x}^*$ from the optimal dual $\boldsymbol{y}^*$, a more complex method not developed in this thesis. The second one is to directly reduce the problem to a non-constrained one with a variable substitution. To do so, one has to find a couple $(F, \boldsymbol{x}_0) \in \mathbb{R}^{n \times (n-p)} \times \mathbb{R}^n$, where F is a null space projector of A and $\boldsymbol{x}_0$ is a solution of the equality constraint [7]. Hence, the following sets are equivalent

$$\{\boldsymbol{x} \mid A\boldsymbol{x} = \boldsymbol{b}\} = \left\{F\boldsymbol{z} + \boldsymbol{x}_0 \mid \boldsymbol{z} \in \mathbb{R}^{n-p},\ \boldsymbol{x}_0 \in \mathbb{R}^n,\ AF\boldsymbol{z} = \boldsymbol{0},\ A\boldsymbol{x}_0 = \boldsymbol{b}\right\} \tag{2.16}$$

and the problem 2.15 can now be transformed into a non-constrained one

$$\text{minimize} \quad f'(\boldsymbol{z}) = f(F\boldsymbol{z} + \boldsymbol{x}_0) \tag{2.17}$$

whose solution is found using the gradient descent (see Algorithm (2)). The optimal solution $\boldsymbol{x}^*$ of Problem (2.15) is finally obtained from the optimal solution $\boldsymbol{z}^*$ of Problem (2.17) with $\boldsymbol{x}^* = F\boldsymbol{z}^* + \boldsymbol{x}_0$.

6. This is the full-rank decomposition, done from the Singular Value Decomposition (SVD). Decomposing A such as $A = USV^\mathsf{T}$ (U, V are orthogonal, S is diagonal), the rank r of S determines the rank of A. The reduced matrices U_r, S_r, V_r, whose obsolete rows and columns are deleted, leads to the equivalent full-rank equality system $S_r V_r^\mathsf{T}\boldsymbol{x} = U_r^\mathsf{T}\boldsymbol{b}$.

7. A simple method to find one couple $(F, \boldsymbol{x}_0)$ is to use $A^\dagger$, the Moore-Penrose pseudo-inverse of A described above, and to notice that $A\boldsymbol{x} = \boldsymbol{b} \Leftrightarrow \boldsymbol{x} = \underbrace{A^\dagger\boldsymbol{b}}_{x_0} + \underbrace{(I - A^\dagger A)}_{F}\boldsymbol{z}$.

2.2.2.1. Special case : the quadratic problem

Again, a particular attention is paid to the minimization of a quadratic function, now subject to linear constraints, which transforms the problem as follows

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\boldsymbol{x}^\mathsf{T} P\boldsymbol{x} + \boldsymbol{q}^\mathsf{T}\boldsymbol{x} + r \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{2.18}$$

Indeed, it holds some good properties which lead to an analytic solution very efficiently by expressing the KKT conditions into a matrix form

$$\left\{ \begin{array}{l} A\boldsymbol{x}^* = \boldsymbol{b} \\ P\boldsymbol{x}^* + \boldsymbol{q} + A^\mathsf{T}\boldsymbol{y}^* = \boldsymbol{0} \end{array} \right. \quad \Leftrightarrow \quad \left[\begin{array}{cc} A & 0 \\ P & A^\mathsf{T} \end{array} \right] \left[\begin{array}{c} \boldsymbol{x}^* \\ \boldsymbol{y}^* \end{array} \right] = \left[\begin{array}{c} \boldsymbol{b} \\ -\boldsymbol{q} \end{array} \right] \tag{2.19}$$

where $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ are the primal and dual solutions if they exist. This equation is called the **KKT system**, and the coefficient matrix on the left-side is called the **KKT matrix**. The resolution is reduced to the cases of the unconstrained quadratic minimization expressed above, depending this time on the rank of the KKT matrix. Thus, the resolution of problem (2.18) becomes straightforward, and the solutions of the primal and dual problem are returned simultaneously.

2.2.3. Resolution of convex problem with inequality constraints

The previous section gives methods to deal with convex optimization subject to equality constraints, this section addresses the core of the issue in optimization, that is the management of inequality constraints. Given a set of m convex functions f_i that represent in a generic manner the inequality constraints, the convex problem becomes

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{x}) \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b} \\ & f_i(\boldsymbol{x}) \leq 0 \quad i \in [1..m] \end{aligned} \tag{2.20}$$

Notice that no assumption is made about the linearity of these constraints. There is no analytical solution, but this system can be solved with iterative methods whose purpose is to reduce the problem into convex subproblems subject to equality constraints only.

According to the encountered problem, many methods are presented in the literature, which go from the simplex algorithm for linear programming [Bartels1969], to interior-point methods (for example the polynomial-time Karmarkar algorithm [Karmarkar1984], again for linear programming), the ellipsoid methods [Luthi1985], and finally the active constraint-set methods [Dauer1991, Dimitrov2009, Escande2010]. The following sections focus on two of these methods.

2.2.3.1. Interior point algorithm : barrier method

The main idea is to approximate Problem (2.20) with another convex problem with equality constraints only [Hindi2006]. To do so, the function $\mathbf{I}_- : \mathbb{R} \longrightarrow \mathbb{R}$, which is the indicator function to the non-positive reals, is introduced such as

$$\mathbf{I}_-(u) = \left\{ \begin{array}{ll} 0 & u \leq 0 \\ \infty & u > 0 \end{array} \right. . \tag{2.21}$$

so it becomes possible to rewrite the problem without any inequality constraints with the following formulation

$$\begin{aligned} &\text{minimize} \quad f_0(\boldsymbol{x}) + \sum_{i=1}^{m} \mathbf{I}_-\left(f_i(\boldsymbol{x})\right) \\ &\text{subject to} \quad A\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{2.22}$$

However, although $\mathbf{I}_-$ is convex, it is not twice continuously differentiable, which makes the resolution difficult with the previous methods.

The idea is then to approximate the indicator function with a logarithmic function such as $\mathbf{I}_- \approx -(1/t)\log(-u) \quad \forall u \in \mathbb{R}^{+*}$ where the parameter $t > 0$ set the accuracy of the approximation as shown in Fig. 2.5 (when $t \longrightarrow \infty$, one gets the function $\mathbf{I}_-$).

Let $\phi(\boldsymbol{x}) = \sum_{i=1}^{m} -\log\left(-f_i(\boldsymbol{x})\right)$ be the concatenation of all the approximated inequality constraints, and multiplying the cost function with the accuracy parameter t, the approximated convex problem can be written as follows

$$\begin{aligned} &\text{minimize} \quad f_0(\boldsymbol{x})t + \phi(\boldsymbol{x}) \\ &\text{subject to} \quad A\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{2.23}$$

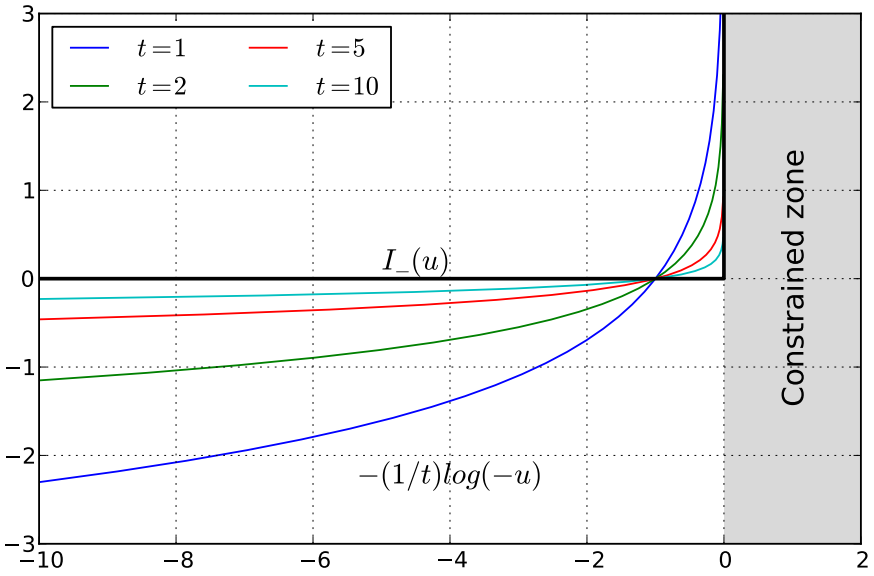


FIGURE 2.5.: Approximation of $\mathbf{I}_-$ according to parameter t.

Roughly, from the optimality conditions and the bounds defined by the primal and dual solutions, one can prove that the optimal solution $\boldsymbol{x}_1^*$ of Problem (2.23) is a m/t-suboptimal solution of the original problem, which means that given the original solution $\boldsymbol{x}_0^*$, it verifies $\|f_0(\boldsymbol{x}_1^*) - f_0(\boldsymbol{x}_0^*)\| \leq m/t$, where m and t are the number of inequalities and the accuracy parameter. Hence, the approximation of the optimal value is found with a bounded error, and given a desired tolerance $\varepsilon = m/t$, Problem (2.20) is transformed into the following one

$$\begin{aligned} &\text{minimize} \quad \frac{m}{\varepsilon} f_0(\boldsymbol{x}) + \phi(\boldsymbol{x}) \\ &\text{subject to} \quad A\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{2.24}$$

The accuracy of this approximation increases while ε decreases.

Although this procedure gives theoretically the optimal solution with a given tolerance, in practice it only works for small problems with moderate accuracy, because the research of the optimal point is very difficult with numerical methods. This is due to the fact that the variations of the Hessian matrix of the cost function (related to the second order derivative of $\phi(\boldsymbol{x})$) are too sharp near the "barriers", which adversely affects convergence.

To cope with this issue, the iterative algorithm called **the barrier method** (formerly named the **Sequential Unconstrained Minimization Technique** (SUMT)) is used. Actually, it is an extension of the previous method which searches for the optimal argument of Problem (2.24) iteratively with a decreasing value of ε, *i.e.* from very inaccurate solutions to more precise ones. Given a feasible point of the problem, a first imprecise solution designated as a central point is found. Then, a loop computes a new central point from the last one with increasing accuracy until the tolerance is fulfilled. The trajectory described by these points is called the central path and is shown in Fig. 2.6. The complete SUMT method is described in Algorithm (3).

Algorithm 3: Sequential Unconstrained Minimization Technique (SUMT).

Input: strictly feasible $\boldsymbol{x}$, $t := t^0 > 0$, $\mu > 1$, tolerance $\varepsilon > 0$

repeat

1 *Increase.* $t := \mu t$

2 *Centering step.* Computing $\boldsymbol{x}^*$ the optimal solution of Problem (2.24)

3 *Update.* $\boldsymbol{x} := \boldsymbol{x}^*$

until $m/t < \varepsilon$ *(m is the number of inequalities)*

2.2.3.2. Active constraint-set method

With the previous method, the whole set of inequality constraints is taken into account simultaneously, which is a great drawback when one has huge inequality set. It turns out that very few constraints are active, *i.e.* the optimal point lies on a very small subset of edges of the domain defined by the inequalities, or is strictly inside. Hence, Problem (2.20) is not modified if it only takes into account the active inequality constraints, which can also be considered as equality constraints in this case.

Of course, the main difficulty is to discover which constraints are active. A method would be to test all the cases (all the inequalities subsets) and to return the best solution, but the number of tests has to be limited by a dedicated algorithm to save the computation time. This is the role of the **active constraint-set method** which cleverly appends or removes some constraints at each iteration, until an optimal and feasible solution is found.

The illustration of this active constraint-set method is developed below for a quadratic problem subject to linear equality and inequality constraints. But in the first place, it is necessary to determine the optimality of a given point, which is possible by checking the KKT conditions of the quadratic problem $\min((1/2)\boldsymbol{x}^\mathsf{T} P\boldsymbol{x} + \boldsymbol{q}^\mathsf{T}\boldsymbol{x} + r$ subject to some linear inequality constraints $G\boldsymbol{x} \preceq \boldsymbol{h}$. Assuming that all these inequalities are active, and recalling that the KKT conditions are necessary and sufficient for optimality in case

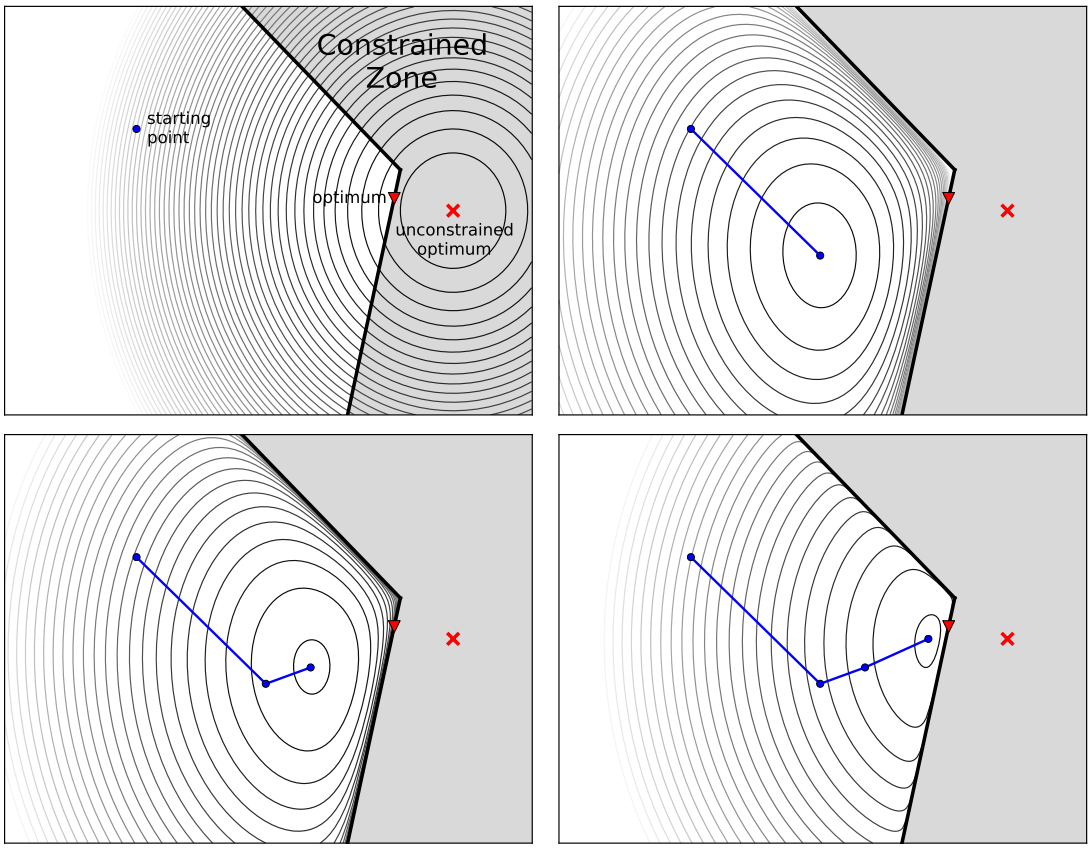


FIGURE 2.6.: Barrier method : the problem is presented in the upper left graph. Contour lines range from light to dark when approaching the optimal value which is located in the constrained zone. So the objective is to build successive unconstrained convex problems by taking "barriers" into account directly in the cost function. The first iteration starts with gentle slopes and finds a solution far from the optimum (upper right). The next ones find more accurate values and converge toward the optimum of the constrained problem, but they become harder to solve due to steeper slopes.

of convex problems, one has the following relation

$$\begin{bmatrix} G & 0 \\ P & G^{\mathsf{T}} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{h} \\ -\boldsymbol{q} \end{bmatrix}. \tag{2.25}$$

The optimality condition is obtained by observing the dual vector $\boldsymbol{y}$: if $\boldsymbol{y} \succeq \mathbf{0}$, the variable $\boldsymbol{x}$ is thus optimal, otherwise the solution can be improved [8]. Using this test, the active constraint-set method for quadratic problems is developed in Algorithm (4).

2.3. Controlling a mechanical system with optimization tools

In the previous sections of this chapters, we have discussed robot control issues for considering robot limitations and physical constraints, which are implicitly taken into

8. There is a third possibility but it only occurs for non-strictly convex problems, which is not the case here.

Algorithm 4: Active constraint-set method.

Set problem : minimize $(f(\boldsymbol{x}) = (1/2)\boldsymbol{x}^\mathsf{T} P\boldsymbol{x} + \boldsymbol{q}^\mathsf{T}\boldsymbol{x} + r)$ s.t. $(G\boldsymbol{x} \preceq \boldsymbol{h})$
Input: a feasible point $\boldsymbol{x}_0$, an active constraint set $\mathcal{S}_0 = \{\}$
repeat
  $\boldsymbol{s}_k \leftarrow$ argmin $f(\boldsymbol{x}_k + \boldsymbol{s})$ *s.t.* $G_k\boldsymbol{s} = \boldsymbol{0}$
  if *(the point* $\boldsymbol{x}_k + \boldsymbol{s}_k$ *feasible)* **then**
    $\boldsymbol{x}_{k+1} \leftarrow$ argmin $f(\boldsymbol{x})$ *s.t.* $G_k\boldsymbol{x} = \boldsymbol{h}_k$
    $\boldsymbol{y}_{k+1} \leftarrow$ the related Lagrangian coefficients of Eq.(2.25)
    if *(*$\boldsymbol{y}_{k+1} \succeq \boldsymbol{0}$*)* **then**
      break : $\boldsymbol{x}_{k+1}$ is optimal !
    else
      $\mathcal{S}_{k+1} \leftarrow \mathcal{S}_k \setminus \{i \mid i^{th}$ line where $\boldsymbol{y}_{k+1}$ negative $[\boldsymbol{y}_{k+1}]_i < 0\}$
  else
    $\mathcal{S}_{k+1} \leftarrow \mathcal{S}_k \cup \{j$, the most violated constraints $[G(\boldsymbol{x}_k + \boldsymbol{s}_k) - \boldsymbol{h}]_j > 0\}$
    $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + u_k\boldsymbol{s}_k$, such as $\boldsymbol{x}_{k+1}$ feasible and $u_k \in [0, 1]$
  $G_{k+1}, \boldsymbol{h}_{k+1}$ sub-matrices of $G, \boldsymbol{h}$ composed of numbered line in $\mathcal{S}_{k+1}$
until *optimal point is found*

account by the optimization problem as equality and inequality constraints, as well as the methods to solve convex problems. Here, this section addresses the optimization-based control of a mechanical system. It aims to determine the most suitable program for the dynamic control problem, as well as the construction of the cost function, and the resolution techniques.

The control systems lead to a rather way to correct the error between actual and desired states, for example by minimizing an error resulting from the quadratic norm of their difference. Beyond the convexity of the problem, this error can be minimized with many dedicated algorithms depending upon the modeling of the problem, such as the Linear Quadratic Program (LQP), the Second-Order Cone Program (SOCP), and the Semi-Definite Program (SDP) [9].

Mechanical systems advanced control design may take benefit of a suitable optimization program. Actually these problems depend hierarchically on each other. The LQP problem, composed of a quadratic cost function with linear constraints, can be translated into a SOCP problem, composed of a linear cost function with quadratic constraints, which can also be translated into a SDP problem, composed of a linear cost function subject to linear matrix inequality constraints. However, the reverse transformations are not possible, and programs which solve a greater variety of problems are also slower in a general manner.

When looking at the constraints expressed in Section 2.1, all of them are linear except one. Indeed, Eq.(2.9) representing the constraint of the contact forces lying in the Coulomb friction cone is nonlinear, but it is at least quadratic. The LMI formulation of the friction cones can be solved with SOCP or SDP [Han2000]. But for simplicity, a less accurate model using a linear formulation is presented in Eq.(2.10).

So, two proper solutions are available : the SOCP (or SDP) with a more accurate

9. There is also the Quadratically Constrained Quadratic Program (QCQP) with possibly non-convex problems, but this is not discussed in this thesis.

model, and the LQP with a faster resolution. In this thesis, **we will rely on the LQP formulation** [10] because the problem is simpler and it is more likely to be used in real time controllers, which is required for the dynamic control of humanoid robots.

2.3.1. Tasks definition

This section focuses on the low-level control of the robot through the achievement of tasks, meaning the control of one or more DoF of the system toward objectives which are generally the minimization of tracking errors. This is the task function approach, a formalism used in robotic control [Espiau1990, Samson1991]. Even if these objectives are independent, they are subject to many disturbances due to the internal and external constraints, the dynamic couplings, and their conflicting interests, which may affect their achievements. The tasks are composed of two main elements, the subset of the controlled DoF, and their desired goals. The following sections develop their mathematical expressions and explain the important points of their constructions.

2.3.1.1. Generic task

A generic task function is defined as an error dependent on $\mathbb{X}$ (the dynamic variable defined in Section 2.1.1) which has to be minimized in the dynamic space. Although $\mathbb{X}$ is overabundant due to the interdependence of $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\chi}$, it makes the description of the tasks simpler, and the way to transform the tasks from a non-independent problem to an independent one is addressed later on.

As written above, a task function is the norm of an error, and a generic task function can be defined in a general manner such as

$$T(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}) = \left\| K_N K_H \big(E(\boldsymbol{q})\mathbb{X} - \mathbf{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \big) \right\|^2 . \tag{2.26}$$

where $(E(\boldsymbol{q})\mathbb{X} - \mathbf{f}(\boldsymbol{q}, \dot{\boldsymbol{q}}))$ is the error to minimize. The purposes of matrices K_H and K_N are respectively to give the possibility to change the homogeneity of the task, and to weight the norm of the computation if one wants to modify the classic Euclidean norm. A focus on these matrices is given in the next sections. In order to separate the parts which compute the acceleration error from the force error, the matrix $E(\boldsymbol{q})$ can be decomposed as follows $[\ E_{\ddot{q}}(\boldsymbol{q}) \quad E_{\chi}(\boldsymbol{q})\ ]$.

2.3.1.2. Acceleration task

An acceleration task acts on the derivative of a twist attached to the system in the Cartesian or joint spaces. Thus, it only depends on the generalized acceleration $\ddot{\boldsymbol{q}}$, and its general definition is written as

$$T_i = \left\| K_{N_i} K_{H_i} \left(\dot{\mathbf{t}}_i - \dot{\mathbf{t}}_i^{des} \right) \right\|^2 \Leftrightarrow \left\| K_{N_i} K_{H_i} \left(J_i(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{J}_i(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \dot{\mathbf{t}}_i^{des} \right) \right\|^2 \tag{2.27}$$

where $J_i(\boldsymbol{q})$, $\dot{J}_i(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and $\dot{\mathbf{t}}_i^{des}$ are respectively the Jacobian of the controlled parts i of the robot, its derivative and the desired acceleration. According to the space considered (Cartesian or joint), $\dot{\mathbf{t}}_i$ may represent respectively a subset of the twist derivative $\dot{\boldsymbol{T}}^i$

10. LQP can always be transformed into SOCP/SDP with the appropriate constraints.

related to the frame Ψ_i, or a subset of the variable $\ddot{\boldsymbol{q}}$, to select the DoF one wants to control. Of course, matrices J_i and $\dot{J}_i$ are built accordingly.

As it only depends on $\ddot{\boldsymbol{q}}$, the acceleration task is defined with the following elements

$$E_i(\boldsymbol{q}) = \begin{bmatrix} J_i(\boldsymbol{q}) & 0 \end{bmatrix} \qquad \mathbf{f}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = -\left(\dot{J}_i(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \dot{\mathbf{t}}_i^{des}\right). \tag{2.28}$$

2.3.1.3. Wrench task

Similarly to the acceleration task, a wrench task acts in the Cartesian and joint spaces. This task is a little simpler because it directly acts on the action variable of the system, so it suffices to find the proper selection matrix. Thus, it only depends on $\boldsymbol{\chi}$, and a wrench task is defined in a general manner as

$$T_i = \left\| K_{N_i} K_{H_i} \left(\mathbf{w}_i - \mathbf{w}_i^{des} \right) \right\|^2 \tag{2.29}$$

where $\mathbf{w}_i$ and $\mathbf{w}_i^{des}$ are respectively the effective and desired values of the wrench i. In the Cartesian space, it represents an interaction with the world (contact, bilateral closure) and may refer to a subset of the variable $\mathbf{w}_{c_i}$, whereas in the joint space it represents a direct control of the actuators, referring to a subset of the input torque vector $\boldsymbol{\tau}$.

As it only depends on $\boldsymbol{\chi}$, it is sufficient to extract the selection matrix S_{w_i} such as $\mathbf{w}_i = S_{\mathrm{w}_i}\boldsymbol{\chi}$, and the wrench task is defined with the following elements

$$E_i(\boldsymbol{q}) = \begin{bmatrix} 0 & S_{\mathrm{w}_i} \end{bmatrix} \qquad \mathbf{f}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \mathbf{w}_i^{des} \tag{2.30}$$

2.3.1.4. Tasks homogenization

In robotics, especially in humanoid robotics, several tasks with different natures are generally performed simultaneously to complete the high-level goals, for instance in acceleration and/or in wrench space. The compatibility of all these tasks is not guaranteed without any prior treatments, this is the multi-tasks issue, *i.e.* the achievement of all goals knowing that they may not be feasible simultaneously.

From a high-level point of view, this problem is expressed as a scheduling process where a sequence has to be selected to optimize some high-level objectives over a finite time horizon. From a low-level point of view, it refers to the simultaneous achievement of the several tasks at one instant, and the problem is then to define the relative tasks importance and the related strategy (hierarchy or trade-off). But, their comparison is possible only if the tasks are all from the same nature, which advocates for a homogenization step.

To consider this problem, the parameter K_{H_i} previously defined is used as a homogenization matrix. For example, one possibility is to "convert" all the acceleration tasks into wrench tasks. To do so, the projection of the inertia matrix onto the task space is defined such as $\Lambda_i(\boldsymbol{q}) = \left(J_i(\boldsymbol{q})M(\boldsymbol{q})^{-1}J_i(\boldsymbol{q})^{\mathsf{T}}\right)^{-1}$, with J_i and M the task Jacobian and the generalized mass matrices. In Cartesian space, Λ_i, which only depends on the configuration $\boldsymbol{q}$, represents the equivalent inertia of the whole tree-structure reduced to one virtual body linked to the frame Ψ_i. The conversion is done by setting $K_{H_i} = \Lambda_i$, which leads to

$$T_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}) = \left\| K_{N_i}\Lambda_i \left(J_i(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{J}_i(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \dot{\mathbf{t}}_i^{des} \right) \right\|^2. \tag{2.31}$$

Another possibility is to "convert" all the wrench tasks into acceleration tasks. This time, the inverse of the projection of the inertia matrix onto the task space is used as the homogenization matrix K_{H_i}, which implies

$$T_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}) = \left\| K_{N_i} \Lambda_i^{-1} \left(\mathbf{w}_i - \mathbf{w}_i^{des} \right) \right\|^2 . \tag{2.32}$$

Of course, these are not the only ways to transform the nature of the tasks, indeed this formulation let the possibility to define its own comparison technique, for example to compute the cost of the tasks as power, energy, etc.

2.3.1.5. Task normalization

Modification of the norm For various reasons, one may want to use a modified norm to compute the tasks errors instead of the Euclidean norm used by default. An idea is to use the Euclidean W-weighted norm. Given a symmetric positive-definite matrix W and a vector $\boldsymbol{a}$, its W-weighted norm is defined such as $\|\boldsymbol{a}\|_W = \sqrt{\boldsymbol{a}^\mathsf{T} W \boldsymbol{a}}$. Due to its good properties, the matrix W admits a square root $\sqrt{W}$ which can be computed with the Cholesky or the Singular Value Decomposition (SVD) algorithms, leading to the following equation

$$\|\boldsymbol{a}\|_W = \left\| \sqrt{W} \boldsymbol{a} \right\| . \tag{2.33}$$

Hence, it becomes possible to change the normalization method in the task computation thanks to the normalization matrix K_N defined above, such as $K_N = \sqrt{W}$. So given task T_i, the left-multiplication of its error by K_{N_i} is equivalent to compute its Euclidean W-weighted norm, as expressed hereinafter

$$T_i = \|K_{N_i} K_{H_i} (E_i \mathbb{X} - \mathbf{f}_i)\|^2 = \|K_{H_i} (E_i \mathbb{X} - \mathbf{f}_i)\|_W^2 . \tag{2.34}$$

The matrix W may refer to any symmetric positive-definite matrix, for example the projection of the inertia matrix in the task space Λ_i, its inverse, the representation of the dynamic manipulability ellipsoid, etc.

Problem of metric in $SE(3)$ Although the normalization problem is explained in a very general context, the desire to change the Euclidean norm comes from a great problem in robotics, that is the metric in $SE(3)$. The tasks are considered as the error minimization in a subset of wrenches or time derivative of twists, but these vectors are composed of translational and rotational parts, and the computation of the classic norm in the concerned space gives an inhomogeneous result.

According to [Doty1992, Doty1993], one way is to use the inertia matrix Λ_i as the normalization matrix, according to the type of error one wants to minimize. For instance, given $\varepsilon_{\ddot{q}}$ the acceleration error of T_i, the Λ_i-weighted norm of this error gives

$$\|\varepsilon_{\ddot{q}}\|_{\Lambda_i}^2 = \varepsilon_i^\mathsf{T} \Lambda_i \varepsilon_i = \delta P_i. \tag{2.35}$$

Similarly, given ε_χ the force error of T_j, its Λ_j^{-1}-weighted norm is expressed as

$$\|\varepsilon_\chi\|_{\Lambda_j^{-1}}^2 = \varepsilon_j^\mathsf{T} \Lambda_j^{-1} \varepsilon_j = \delta P_j. \tag{2.36}$$

The two values δP_i and δP_j are physically consistent. They are both homogeneous to power time-derivative, thus it becomes possible to compare these two different tasks.

2.3.2. Problem formalism : dependent or independent variable

The description of the equations of motion Eq.(2.1) raises an interesting point about the interdependence of the dynamic variables. All the previous equations dealing with the dynamic of the system are function of $\ddot{\boldsymbol{q}}$ and $\boldsymbol{\chi}$ (respectively the generalized acceleration and the action variable) but the equations of motion Eq.(2.1) show that $\boldsymbol{\chi}$ is sufficient. So there are two available formalisms, one using an overabundant representation of the system with the dynamic variable $\mathbb{X}$ as the unknown of the problem, and the other using the action variable $\boldsymbol{\chi}$ only. The transformation of the tasks from the first formalism to the other one is performed by implicitly using the equations of motion in the tasks representation.

As the generalized mass matrix M is usually strictly positive-definite, one gets the following equation (for the sake of clarity, the dependencies with respect to $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are omitted) $\ddot{\boldsymbol{q}} = M^{-1}(\boldsymbol{g}-\boldsymbol{n}+J_\chi^\mathsf{T}\boldsymbol{\chi})$, and the generic task defined in Section 2.3.1.1 becomes

$$\begin{aligned} T &= \|K_N K_H\left(E\mathbb{X}-\mathbf{f}\right)\|^2 = \|K_N K_H\left(E_{\ddot{q}}(\boldsymbol{q})\ddot{\boldsymbol{q}}+E_\chi\boldsymbol{\chi}-\mathbf{f}\right)\|^2 \\ &= \left\|K_N K_H\left(E_{\ddot{q}}M^{-1}\left(\boldsymbol{g}-\boldsymbol{n}+J_\chi^\mathsf{T}\boldsymbol{\chi}\right)+E_\chi\boldsymbol{\chi}-\mathbf{f}\right)\right\|^2 \\ &= \|K_N K_H\left(E_2\boldsymbol{\chi}-\mathbf{f}_2\right)\|^2 \end{aligned}$$

with

$$E_2 = E_{\ddot{q}}M^{-1}J_\chi^\mathsf{T}+E_\chi \qquad\qquad \mathbf{f}_2 = \mathbf{f}-E_{\ddot{q}}M^{-1}(\boldsymbol{g}-\boldsymbol{n}). \tag{2.37}$$

The whole problem has to be expressed according to either the variable $\mathbb{X}$ or the variable $\boldsymbol{\chi}$. Then, the equality and inequality constraints defined in Section 2.1 must also be transformed, and they become

$$\begin{aligned} &A\mathbb{X} \leqq \boldsymbol{b} \\ \Leftrightarrow &A_{\ddot{q}}\ddot{\boldsymbol{q}}+A_\chi\boldsymbol{\chi} \leqq \boldsymbol{b} \\ \Leftrightarrow &A_2\boldsymbol{\chi} \leqq \boldsymbol{b}_2 \end{aligned}$$

with

$$A_2 = A_{\ddot{q}}M^{-1}J_\chi^\mathsf{T}+A_\chi \qquad\qquad \boldsymbol{b}_2 = \boldsymbol{b}-A_{\ddot{q}}M^{-1}(\boldsymbol{g}-\boldsymbol{n}). \tag{2.38}$$

Due to the surjective relation from the action space (where $\boldsymbol{\chi}$ lies) to the acceleration space (where $\ddot{\boldsymbol{q}}$ lies), the transformation from $\mathbb{X}$ to $\boldsymbol{\chi}$ can be achieved, but the transformation from $\mathbb{X}$ to $\ddot{\boldsymbol{q}}$ is generally not possible without any assumption on the vector $\boldsymbol{\chi}$, that is why this case is not exposed in this thesis.

2.3.3. Special case : the task space dynamics representation

To validate our definition of the task functions, the previous expressions are compared with existing definitions. For instance, a specific representation of the tasks denoted by **the task space dynamics representation** is described in the literature [Khatib1987] and relies on the following equation

$$\Lambda_i(\boldsymbol{q})\dot{\mathbf{t}}_i = \bar{J}_i(\boldsymbol{q})^\mathsf{T}J_\chi(\boldsymbol{q})^\mathsf{T}\boldsymbol{\chi}-\boldsymbol{\mu}_i(\boldsymbol{q},\dot{\boldsymbol{q}})+\boldsymbol{p}_i(\boldsymbol{q}) \tag{2.39}$$

with $\dot{\mathbf{t}}_i$ the task space acceleration associated to task i, $\Lambda_i(\boldsymbol{q}), \boldsymbol{\mu}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}), \boldsymbol{p}_i(\boldsymbol{q})$ the task space projections of respectively the inertia matrix (defined above), the nonlinear effects vector and the gravity force vector. By definition they are expressed as follows : $\boldsymbol{\mu}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \bar{J}_i(\boldsymbol{q})^\mathsf{T} \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \Lambda_i(\boldsymbol{q}) \dot{J}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}}$ and $\boldsymbol{p}_i(\boldsymbol{q}) = \bar{J}_i(\boldsymbol{q})^\mathsf{T} \boldsymbol{g}(\boldsymbol{q})$, where matrix $\bar{J}_i(\boldsymbol{q})$ represents the dynamically consistent weighted pseudo-inverse [11] of $J_i(\boldsymbol{q})$ which is expressed as $\bar{J}_i(\boldsymbol{q}) = M(\boldsymbol{q})^{-1} J_i(\boldsymbol{q})^\mathsf{T} \Lambda_i(\boldsymbol{q})$.

Now, due to the fact that the system is subject to dynamic couplings and constraints, if one wants to perform a desired task space acceleration such as $\dot{\mathbf{t}}_i = \dot{\mathbf{t}}_i^{des}$, the solution is to minimize the error induced in Eq.(2.39). It turns out that this expression can be written as

$$
\begin{aligned}
&\left\| \bar{J}_i(\boldsymbol{q})^\mathsf{T} J_\chi(\boldsymbol{q})^\mathsf{T} \boldsymbol{\chi} - \boldsymbol{\mu}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{p}_i(\boldsymbol{q}) - \Lambda_i(\boldsymbol{q}) \dot{\mathbf{t}}_i^{des} \right\|^2_{\Lambda_i^{-1}} \\
&= \left\| \Lambda_i(\boldsymbol{q}) \left(J_i(\boldsymbol{q}) M(\boldsymbol{q})^{-1} \left(J_\chi(\boldsymbol{q})^\mathsf{T} \boldsymbol{\chi} - \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) \right) + \dot{J}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} - \dot{\mathbf{t}}_i^{des} \right) \right\|^2_{\Lambda_i^{-1}} \\
&= \left\| \Lambda_i(\boldsymbol{q}) \left(J_i(\boldsymbol{q}) \ddot{\boldsymbol{q}} + \dot{J}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} - \dot{\mathbf{t}}_i^{des} \right) \right\|^2_{\Lambda_i^{-1}}
\end{aligned}
$$

where the Λ_i^{-1}-weighted norm is used to get a consistent value. Finally, it results in the exact definition of an acceleration error converted into a wrench task, with the matrices $K_{N_i} = \sqrt{\Lambda_i^{-1}}$ and $K_{H_i} = \Lambda_i$. Given the acceleration error $\varepsilon_{\ddot{q}} = \dot{\mathbf{t}}_i^{des} - \dot{\mathbf{t}}_i$, the computation of the task is reduced to

$$
T_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}) = \varepsilon_{\ddot{q}}^\mathsf{T} \Lambda_i \varepsilon_{\ddot{q}}. \tag{2.40}
$$

2.3.4. Predefined controllers to achieve task : task servoing

The previous section deals with the construction of the tasks, now it remains to determine the task errors, *i.e.* to compute the desired values one wants to reach. Here, the aim is to express the instantaneous objective of the task without taking into account the multiple disturbances which may make its achievement impossible. In this purpose, some controllers from the literature are presented in this section.

2.3.4.1. Proportional derivative control

In order to ensure some robustness with respect to robot model and to enforce a given task behavior (*e.g.* a virtual spring-damper system), a controller can be derived at the task level. When dealing with an element from the time-derivative of a twist during a tracking task, this type of controller is often written

$$
\dot{\mathbf{t}}^{des} = \dot{\mathbf{t}}^{goal} + K_p \boldsymbol{\epsilon}_p + K_d \dot{\boldsymbol{\epsilon}}_p \tag{2.41}
$$

where $\dot{\mathbf{t}}^{goal}$ is the trajectory reference acceleration, $\boldsymbol{\epsilon}_p, \dot{\boldsymbol{\epsilon}}_p$ are the pose and velocity errors, and K_p, K_d are the proportional and derivative gains reflecting respectively the stiffness and damping of the virtual system. Unlike the vector $\dot{\boldsymbol{\epsilon}}_p = \mathbf{t}^{goal} - \mathbf{t}$, the computation of $\boldsymbol{\epsilon}_p$ is trivially computed when dealing with its position parts, but it requires to resort to a non-minimal representation to describe its orientation parts, *e.g.* by using

11. The dynamic consistency is ensured by using the inertia matrix as a weight for pseudo-inversion.

the quaternions [Altmann1986, Yuan1988, Wang2006]. Notice that a reaching task is a particular case of the tracking task where the goal trajectory is reduced to one point and $\dot{\mathbf{t}}^{goal} = \mathbf{t}^{goal} = \mathbf{0}$.

An equivalent wrench formulation leads to

$$\mathbf{w}^{des} = \mathbf{w}^{goal} + K_{fp}\boldsymbol{\epsilon}_w + K_{fi}\int \boldsymbol{\epsilon}_w dt \tag{2.42}$$

where $\mathbf{w}^{goal}$ is the reference force along a trajectory, $\boldsymbol{\epsilon}_w$ is the wrench error, and the matrices K_{fp}, K_{fi} are proportional and integral gains.

2.3.4.2. Quadratic control for walking

When considering humanoid control, and particularly the walking task, the accomplishment of complex behaviors with simple virtual spring-damper systems is very difficult to obtain, that is why more sophisticated task controllers are necessary. The purpose of this subsection is to control the robot for the walking task, it does not intend to quantify the robot equilibrium state, as proposed in [Barthelemy2008]. Here, the objective is to get some trajectories that the robot has to track to perform locomotion.

To tackle this problem, the literature proposes the control of the Zero Moment Point (ZMP), a virtual point projected onto the ground which gives some information about the balancing of the robot. It has been introduced in [Vukobratovic1973], and further used in the other works to control a humanoid robot, for instance in [Kajita2006, Wieber2006, Kanoun2009b] . But some assumptions are required to compute the ZMP properly. First, all the contact points must be coplanar, which implies a flat ground, no interaction with the upper body during the walk and no task dedicated to the climbing of the stairs. Second, the friction between the feet and the ground is not taken into account, meaning that the control remains the same when the robot walks on concrete or on ice.

The computation of the ZMP is not linear with respect to the action variable (the work of Sardain and Bessonet [Sardain2004a] gives useful explanations), however its approximation can be controlled in order to perform a walking or a balancing task through the control of the Center of Mass (CoM), as exposed in [Kajita2003]. It is computed as follows

$$\begin{aligned} \ddot{\boldsymbol{c}} &= J_{com}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{J}_{com}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \\ \boldsymbol{z} &= \boldsymbol{c} - \frac{h}{g}\ddot{\boldsymbol{c}} \end{aligned} \tag{2.43}$$

where J_{com} is the Jacobian of the CoM, h is its height, $\boldsymbol{z}, \boldsymbol{c}$ are respectively the horizontal position vectors of the ZMP and of the CoM, and g is the gravity value. These equations directly refer to the Linear Inverted Pendulum Model (LIPM), an unstable system whose control must consider the prediction of the future states of the system. $\boldsymbol{z}$ and $\boldsymbol{c}$ would respectively represent the foot of the pendulum and the projection of the point mass onto the ground.

As described in [Wieber2006], the initial configuration of the CoM is denoted by $C_0 = [\boldsymbol{c}, \dot{\boldsymbol{c}}, \ddot{\boldsymbol{c}}]^\mathsf{T}$ and the prediction of the ZMP along a time horizon H is denoted by $Z_H = [\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_H]^\mathsf{T}$. Hence, the input matrix is composed of the future CoM jerks $\dddot{C}_H = [\dddot{\boldsymbol{c}}_0, \dddot{\boldsymbol{c}}_1, ..., \dddot{\boldsymbol{c}}_{H-1}]^\mathsf{T}$ and one can compute the state and input matrices P_x, P_u to

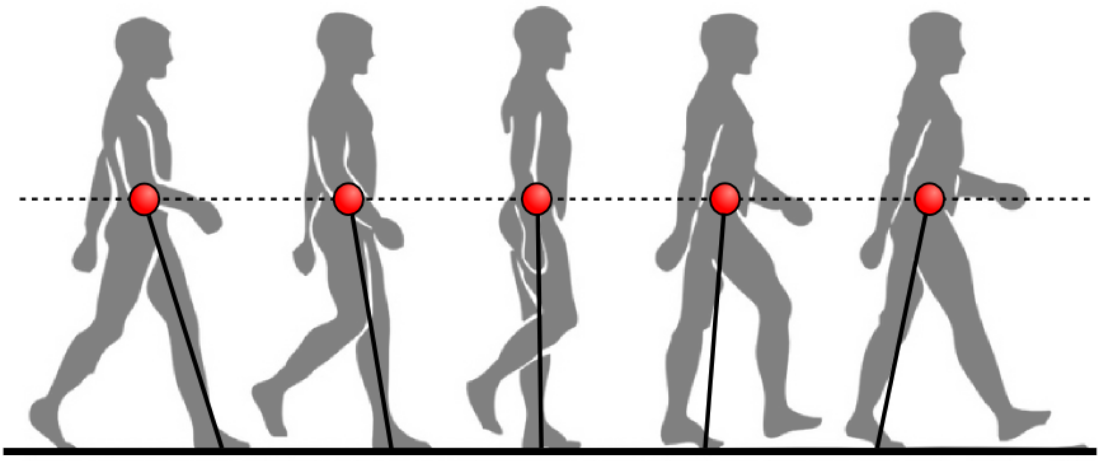


FIGURE 2.7.: The human gait can be model by a Linear Inverted Pendulum Model (LIPM) where the total mass of the human is supposed to be concentrated in one point (red) whose altitude remains constant. Then, the supporting foot stands for the pendulum foot.

obtain the equation

$$Z_H = P_x C_0 + P_u \dddot{C}_H$$

The objective is then to minimize the norm of the difference between a ZMP reference (Z_{ref} computed *a priori*) and the predicted trajectory. This minimization problem is unconstrained and can be solved analytically, which gives

$$\dddot{C}_H = -(P_u^{\mathsf{T}}.P_u + R.I)^{-1}.P_u^{\mathsf{T}}.(P_x.X_{com} - Z_{ref}) \tag{2.44}$$

where R is a ratio between the state and the input tracking error, and I is the identity matrix. Finally, the first row of this input matrix controls the ZMP trajectory, and given $\ddot{\boldsymbol{c}}_p$ the previous acceleration of the CoM and dt the time step, the desired acceleration of the CoM task becomes

$$\dot{\mathbf{t}}_{com}^{des} = \ddot{\boldsymbol{c}}_p + \dddot{\boldsymbol{c}}_0 dt.$$

2.3.4.3. Impedance control with the upper limbs

The impedance control is a way to unify the twist and wrench controls [Hogan1987, Morel1994, Morel1998, Szewczyk1998]. Instead of directly controlling forces or accelerations, impedance control aims at inducing a proper behavior to reject the disturbances with respect to the contacts with the environment. A typical approach is to set a reference position to a frame attached to the robot and to control the wrench induced by a disturbance with a unified controller for non-constrained and constrained motions.

A simple impedance controller can be modeled using a spring-damper system linking the controlled part of the system to the virtual reference. Hence, the choice of the reference position, the stiffness and the damping coefficients allows to tune the reaction behavior, for instance to speed up the completion of the task or to increase the contact wrench of the end-effector.

Note that impedance control can be coupled with the quadratic control presented in Section 2.3.4.2 for the walking task. It allows manipulation tasks while maintaining balance, by taking into account the relative perturbations which can be anticipated or not. So it gives the possibility to generate safer procedures by tuning the stiffness of manipulation tasks to change the ZMP trajectory [Ibanez2011].

2.3.5. Managing several tasks simultaneously

All the tasks described above have to be mixed to achieve the desired high-level missions. This can be treated with a stack of tasks as presented in [Mansard2009]. However, these tasks are subject to all the equality and inequality constraints developed in Section 2.1, plus some incompatibilities between them, leading to the use of an optimization program. The well-known solver used in this thesis is the LQP which solves the following problem

$$\begin{aligned} \min_{(\boldsymbol{x})} \quad & \frac{1}{2}\left(\boldsymbol{x}^\top P\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} + r\right) \\ s.t.: \quad & G\boldsymbol{x} \preceq \boldsymbol{h} \\ & A\boldsymbol{x} = \boldsymbol{b} \end{aligned} \tag{2.45}$$

where $\boldsymbol{x}$ is the vector to optimize, $P, \boldsymbol{q}, r$ represent the quadratic cost function, $G, \boldsymbol{h}$ define the inequality constraints, and $A, \boldsymbol{b}$ define the equality constraints, which are the concatenation of Eqs.(2.1)–(2.10).

The aim is to build these matrices according to the tasks and the physical constraints acting on the robot and, when the solution is found, to extract the input torque vector $\boldsymbol{\tau}$ and actuate the system. $\boldsymbol{x}$ can be replaced either by $\mathbb{X}$ or $\boldsymbol{\chi}$, depending upon the chosen formalism. In the remaining of the thesis, the variable used is $\mathbb{X}$, knowing that the transformation from one formalism to another is always possible, as described in Section 2.3.2. Two opposite strategies are described in the following sections to perform the tasks, the hierarchy and the weighting.

2.3.5.1. Hierarchical strategy

In the case of a hierarchical strategy, the higher tasks are achieved without any disturbance from the lower tasks. The first point to do is to sort the tasks by order of importance, a problem handled by the user or a high-level decision program. This strategy is close to the projection onto successive null spaces that grows as the number of tasks increases, as exposed in [Sentis2007, Khatib2008], but the equality and especially the inequality constraints must be taken into account. To do so, we follow the work of [Kanoun2009b], and the optimization problem is solved recursively as detailed in Algorithm (5). Note that in his work, Kanoun also implements a mean to treat "inequality tasks" thanks to the hierarchical strategy.

Given an ordered set of n tasks $T_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X})$, the program i is solved subject to $A_i\mathbb{X} = \boldsymbol{b}_i$ and $G\mathbb{X} \preceq \boldsymbol{h}$, and the solution $\mathbb{X}_i^*$ will be part of the next problem in the equality constraints, knowing that the initial problem is built such as $A_1 = A$, $\boldsymbol{b}_1 = \boldsymbol{b}$. The function to minimize at level i is then

$$\frac{1}{2}\left(T_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}) + \omega_0^2 . T_0(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X})\right) \tag{2.46}$$

A task T_0 is dedicated the minimization of the whole optimization variable. This is required when the control problem has many solutions, in order to ensure the uniqueness and more specifically to provide a solution that minimizes the input torque vector $\boldsymbol{\tau}$. Being in conflict with any other task, T_0 has a very small weight $0 < \omega_0 \ll 1$ to limit the induced error.

Algorithm 5: LQP-based control with hierarchical strategy.

$A_1 \leftarrow A$ and $\boldsymbol{b}_1 \leftarrow \boldsymbol{b}$
for $i = 1..n$ **do**
  $T_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}) = \left\| K_{N_i} K_{H_i} \big(E_i(\boldsymbol{q}) \mathbb{X} - \mathbf{f}_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) \big) \right\|^2$
  get $\mathbb{X}_i^*$, *the solution of the problem*

$$\begin{aligned} \min_{(\mathbb{X})} \quad & \frac{1}{2} \big(T_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}) + \omega_0^2 . T_0(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}) \big) \\ s.t. : \quad & G\mathbb{X} \preceq \boldsymbol{h} \\ & A_i \mathbb{X} = \boldsymbol{b}_i \end{aligned}$$

  $A_{i+1} \leftarrow \begin{bmatrix} A_i^\mathsf{T} & (K_{N_i} K_{H_i} E_i)^\mathsf{T} \end{bmatrix}^\mathsf{T}$
  $\boldsymbol{b}_{i+1} \leftarrow \begin{bmatrix} \boldsymbol{b}_i^\mathsf{T} & (K_{N_i} K_{H_i} E_i \mathbb{X}_i^*)^\mathsf{T} \end{bmatrix}^\mathsf{T}$

Finally, in order to have a canonical expression of the quadratic cost function, the elements $P, \boldsymbol{q}, r$ in Eq.(2.45) depend on the tasks T_0, T_i such as

$$E = \begin{bmatrix} \omega_0 K_{N_0} K_{H_0} E_0 \\ K_{N_i} K_{H_i} E_i \end{bmatrix} \qquad \mathbf{f} = \begin{bmatrix} \omega_0 K_{N_0} K_{H_0} \mathbf{f}_0 \\ K_{N_i} K_{H_i} \mathbf{f}_i \end{bmatrix}$$

$$P = E^\mathsf{T} E \qquad \boldsymbol{q} = -2\mathbf{f}^\mathsf{T} E \qquad r = \mathbf{f}^\mathsf{T} \mathbf{f}. \tag{2.47}$$

2.3.5.2. Weighting strategy

A weighting strategy associates each task with a coefficient that sets its importance with respect to the others (a task with a higher weight gets a higher priority). Again, the first point is to set these coefficients, but unlike the hierarchy which defines a discrete set of importances, these ones are real values. As a consequence, priorities are not strict and all tasks are achieved according to the trade-off defined by the weights, as described in Algorithm (6). Given a set of n tasks and their related weights $(T_i(\boldsymbol{q}, \dot{\boldsymbol{q}}, \mathbb{X}), \omega_i)$ $i \in [1..n]$, one solves their weighted sum subject to the concatenation of the constraints Eqs.(2.1)–(2.10). The task T_0 plays the same role as above, where its importance is very low with respect to all the other tasks, meaning that $\omega_0 \ll \omega_i \ \forall i$. This program is solved only one time per call. To obtain the canonical form of the quadratic cost, the computation of the elements $P, \boldsymbol{q}, r$ is done in the same way as in Eq.(2.47) considering the following components

$$E = \begin{bmatrix} \omega_0 K_{N_0} K_{H_0} E_0 \\ \vdots \\ \omega_n K_{N_n} K_{H_n} E_n \end{bmatrix} \qquad \mathbf{f} = \begin{bmatrix} \omega_0 K_{N_0} K_{H_0} \mathbf{f}_0 \\ \vdots \\ \omega_n K_{N_n} K_{H_n} \mathbf{f}_n \end{bmatrix}. \tag{2.48}$$

Algorithm 6: LQP-based control with weighting strategy.

get $\mathbb{X}_i^*$, *the solution of the problem*

$$\begin{aligned} \min_{(\mathbb{X})} \quad & \frac{1}{2}\left(\sum_{i=1}^{n}\left(\omega_i^2.T_i(\boldsymbol{q},\dot{\boldsymbol{q}},\mathbb{X})\right)+\omega_0^2.T_0(\boldsymbol{q},\dot{\boldsymbol{q}},\mathbb{X})\right) \\ s.t.: \quad & G\mathbb{X} \preceq \boldsymbol{h} \\ & A\mathbb{X} = \boldsymbol{b} \end{aligned}$$

2.3.6. Chaining up tasks in a physically constrained environment

The optimization-based control of robots is presented above, here we focus on the synthesis of complex behaviors. The previous section describes how to perform several tasks simultaneously, but the mechanical system must also chain the tasks over time in a continuous way, and the continuity of the input torque vector must be ensured, which is a critical point of the control.

2.3.6.1. Transition in the set of task

This issue can be treated according to two different approaches. On one hand, the state changes dramatically, for instance due to an impact, an objective out of range, etc. The reaction is quick, but the actuators may have some problems to respect the optimal torque values, that is why a model of the actuation capability must be added to the set of constraints. This has to be expressed in terms of the dynamic variable, so given the maximum time derivative of torque $\dot{\boldsymbol{\tau}}_{max}$, this constraint becomes

$$-\dot{\boldsymbol{\tau}}_{max} \leq \frac{\boldsymbol{\tau}-\boldsymbol{\tau}_p}{dt} \leq \dot{\boldsymbol{\tau}}_{max} \Leftrightarrow \quad \begin{bmatrix} 0 & S_\tau \\ 0 & -S_\tau \end{bmatrix} \mathbb{X} \quad \preceq \begin{bmatrix} \dot{\boldsymbol{\tau}}_{max}dt+\boldsymbol{\tau}_p \\ \dot{\boldsymbol{\tau}}_{max}dt-\boldsymbol{\tau}_p \end{bmatrix} \tag{2.49}$$

where $\boldsymbol{\tau}_p$ is the previous torque vector applied to the robot and dt is the sampling time between two successive calls of the LQP. Hence the continuity of the torque vector is ensured by this constraint.

On the other hand, the user decides to change the set of tasks, their objectives, their activities, etc. and a more versatile approach is to choose the weighting strategy and to change the weights of the tasks in a continuous manner until the transition is complete. This can be done manually or through a program which scripts their evolutions based on perceptual information, and gives a great basis to interface the low-level control with a high-level decision making engine. This is more detailed in the experiments of Chapter 4. If the transitions are done manually, they must follow continuous functions during a certain amount of time, again to preserve the input torque continuity. For instance, one can deal with linear functions or piecewise polynomial functions, and as the importances are a matter of order, one may prefer a logarithmic scale to represent the transitions.

Of course, the choice of the tasks weights and their evolutions should be automated to generate an efficient generic whole-body motion controller. With the use of a relatively sequential approach, each task has triggers for its start and its end. A supervisor may evaluate its completeness and modifies its weight according to the mission context and

with respect to the criticality of the other tasks. As a matter of fact, the tasks weights can be modified automatically without requiring any complex tuning

$$\omega_i = \begin{cases} \omega_{prev}, \ t < t_e \\ \exp\left((\log(\omega_{next}) - \log(\omega_{prev})) \dfrac{t - t_e}{t_s - t_e} + \log(\omega_{prev}) \right), \ t \in [t_e, t_s] \\ \omega_{next}, \ t > t_s \end{cases} \tag{2.50}$$

where ω_{prev} and ω_{next} are the weights of task i respectively for the previous and the next mission contexts, and t_e, t_s are the time triggers respectively for the end of the previous mission context and the start of the next one. The time between t_e and t_s is thus naturally defined as the transition time. Weights evolution based on Eq.(2.50) is presented in Fig. 2.8.

Of course, the constraint about the input torque derivative in Eq.(2.49) and the smooth transitions of the tasks weights are complementary and can be used simultaneously.

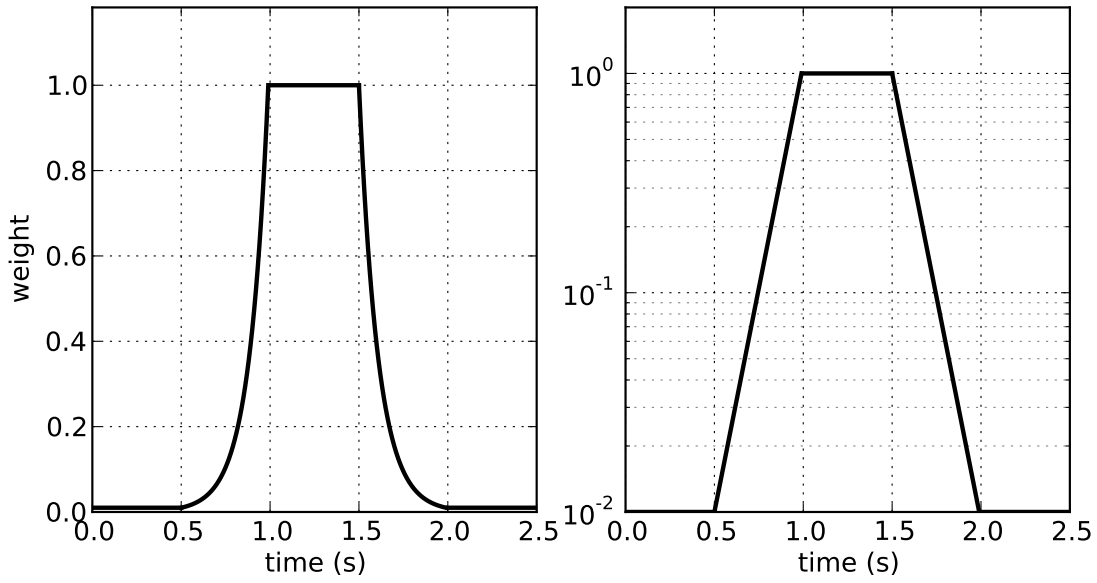


FIGURE 2.8.: Weights evolution, from $\omega = 1e^{-2}$ to 1 and vice versa – Left : plotting on linear scale. Right : plotting on logarithmic scale.

2.3.6.2. Transition in the set of constraints

Similarly, the set of constraints evolves over time, mainly due to the contacts evolution of the robot with its environment. However, the management of these modifications is difficult because the state of the constraints is binary, *i.e.* the contacts are open or closed, which implies that their evolutions are discontinuous.

One can wonder what would be the reaction of the robot subject to these modifications. It may have few consequences, for example if the robot grabs or releases a door knob, because the input vectors of the two states are close enough and do not induce very different dynamic behaviors. But in the opposite, the reaction may become erratic, and even dangerous, if the behaviors with and without the contact vary greatly and imply very different input vectors. Furthermore, it is totally different if the robot breaks the

contacts willingly or not, for instance if the robot gets up from a chair, or if the support vanishes. Although Eq.(2.49) ensures that the torque vector does not undergo any sharp evolution, it may lead to very problematic states, so this constraint is not sufficient.

When the contact breaking is not wanted, the robot undergoes the modifications and must have the required capabilities to recover a stable behavior. But when it is wanted, the idea is to limit the variation between the constrained and the non-constrained states, and to transform the interaction constraints into tasks with a weighting strategy. Considering one kinematic loop in the graph of the system and its related constraint, it represents a joint in the system which forces the relative twist between two frames thanks to a wrench. If the loop is open (*i.e.* no contact), the wrench is null and the twist is free, but if it is closed, the twist is constrained and the wrench is generally not null. It recalls the Signorini conditions [Jean1999], the objective is then to respect them in a continuous manner.

For example, if a frictional contact has to be broken, the transition of the constraint expressed in Eq.(2.10) is performed as follows. First, a wrench task T_{w} is set with a desired wrench $\mathbf{w}^{des} = \mathbf{0}$ and a null weight $\omega_{\mathrm{w}} = 0$. By increasing ω_{w}, the error of the task decreases accordingly until it becomes inferior to a very low tolerance. Then, the interaction becomes practically null and the wrench constraint can be removed, but the acceleration constraint must be transformed into an acceleration task [12] T_a, with a desired value $\dot{\mathbf{t}}^{des} = \mathbf{0}$ and a heavy weight ω_a which decreases to 0 during a short period of time. Hence, it avoids any sudden motion and ensures a continuous transition.

2.4. Conclusion

This chapter presents the methods to perform robotic optimization-based controls in dynamics. The first part enumerates the main constraints acting on robots, both internal as actuation limits or external as contacts, which are equalities and inequalities. Because of these constraints, analytical solutions for control are difficult to obtain, and this advocates for the use of optimization program. So the second part focuses on the resolution of convex problems whose variables are continuous, especially on LQP which is dedicated to the resolution of quadratic problems. Then, the equalities and inequalities are taken into account as the LQP constraints, and the definition of the quadratic cost function is developed in the third part of this chapter. It describes the tasks and their possible modifications, as well as a formalism using an independent variable. Besides, it analyses the changes in the tasks and constraints sets, and proposes a way to perform good transitions (to obtain continuous evolutions of the input torque vector) with a weighting strategy and soft modifications of the weights.

However, the choice of these weights and their evolution is not straightforward. For simple sequences, it can be user-defined, but in the presence of undetermined events this should be automated. The next chapter presents some experiments about the use of this LQP-based controller, its capabilities and performances, and develops an association with a high-level controller to automatically choose the tasks goals and weights.

12. Equality constraint $A\mathbb{X} = b$ can become task $T = \|A\mathbb{X} - b\|$. Then, there should be an error.

3. Experiments on humanoid robots : low-level control

Contents

This chapter aims at illustrating the effectiveness of the proposed control framework on a fairly complex humanoid robot : iCub [Sandini2007]. iCub was designed by the RobotCub consortium [RobotCub] and is shown on Fig. 3.1. The aim of this consortium was to develop an open-source infant-like robotic platform dedicated to *developmental robotics*. Especially, its size, motor and cognitive abilities were inspired from the cognitive and physical abilities of a three years old child. iCub is a 53 DoF full body humanoid robot designed to crawl on all fours members, and sit up with free arms. Most of its DoF are located in the upper-body, especially in the highly anthropomorphic hands, which allow dexterous and fine manipulation. It has proprioceptive, visual, vestibular, auditory and haptic sensory capabilities [Tsagarakis2007, Metta2010]. It is a completely open system platform : both hardware and software are licensed under the GNU General Public License (GPL), and the middleware used for intra-process communication, YARP, is also an open-system released under the GNU Lesser General Public License (LGPL) [Metta2006].

Despite the presence of an early version of iCub at ISIR, the proposed validation is done on a simulated version of the robot. Several technical aspects justify this choice :

- iCub legs and feet were not designed with walking in mind. As a result the feet are very small and rigid, the power of the legs is not very high and the force sensors required for walking are not available ;
- iCub comes with an umbilical which is mandatory for energy reasons (the robot has no batteries) and which strongly restricts its free locomotion ;

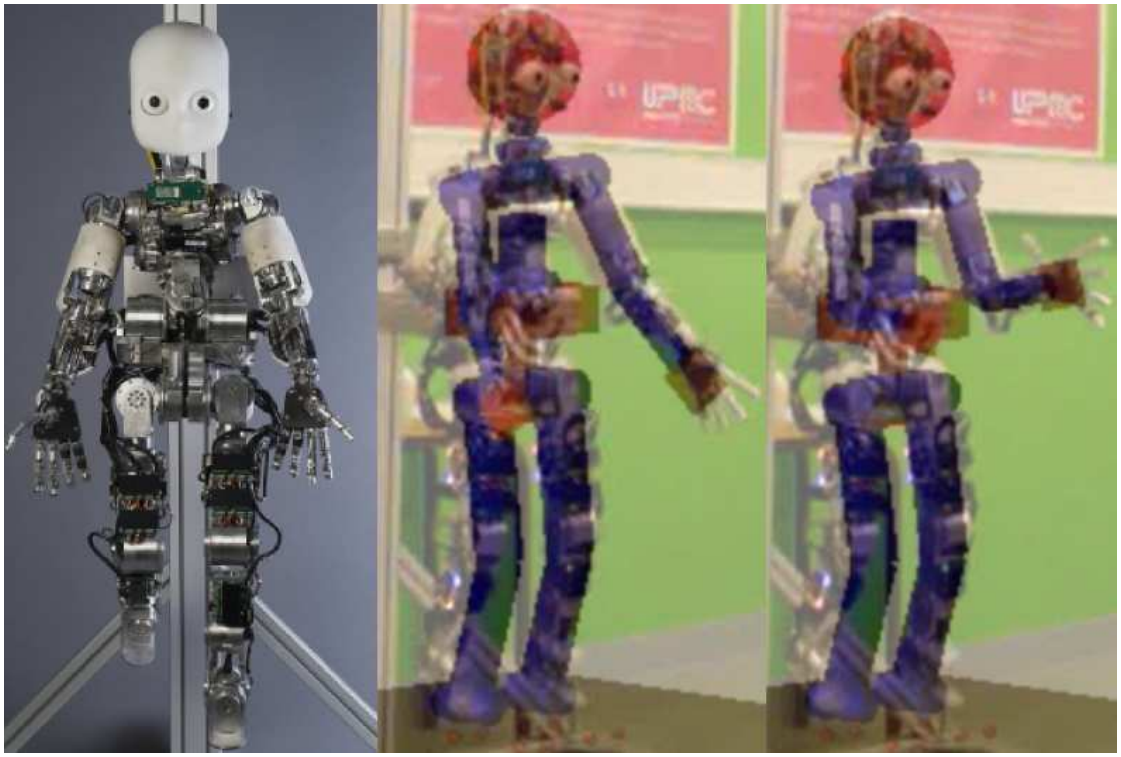


FIGURE 3.1.: Left : iCub robot designed by the RobotCub consortium. Middle & right : superimposition between virtual and real sequences performing the same trajectories.

– low-level torque control cannot be achieved, which is a major issue when dealing with robots whose physical interactions with the environment are mandatory to ensure their controllability.

Given these constraints and the fact that the major contribution of this work is on control algorithms, the choice is made to keep the work in simulation and to propose prototyping software tools that can be used prior to any test on a real robot. Even though this choice alleviates the implementation from several experimental factors (mostly related to perception), constraints related to the overall computational load of the control algorithm are considered.

The simulation framework is the open-source dynamic simulator Arboris-python developed at the ISIR and whose internal mechanisms are detailed in Section 1.2. Dexterous manipulation and active visual servoing are not topics covered by the work in this thesis, and as a consequence only 32 actutated DoFs are modeled in addition to six virtual floating joints to locate the root of the robot with respect to the inertial frame. Also, four contact points are added at the extremities of each foot to simulate contact with the ground (using the Arboris-python class `SoftFingerContact`, cf. Section A.1.4) [1].

On the programming side, the controller is coded in the Python language which is highly suitable for quick and efficient software prototyping and naturally integrates with the simulator. More specifically, the implementation of this controller relies on CVXOPT [Cvxopt], a free Python package dedicated to convex optimization. Among many other

1. Arboris-python cannot simulate contact between two boxes, or a box and a plane

problems, CVXOPT can solve LQPs and SOCPs using the interior-point method (see Section 2.2.3.1).

To define the problem constraints (see Section 2.3.2), the maximum input torque vector $\boldsymbol{\tau}_{max}$ and the joint position limits $\boldsymbol{q}_{min}, \boldsymbol{q}_{max}$ are extracted from the description of the iCub robot [RobotCub], the boundaries of the generalized velocity are $-\dot{\boldsymbol{q}}_{min} = \dot{\boldsymbol{q}}_{max} = 5\ rad.s^{-1}$, and the anticipation coefficients are set to $h_1 = 0.2\ s$, $h_2 = 0.1\ s$ (we consider no direct constraint on the generalized acceleration vector). Task T_0 presented in Algorithms (5)–(6) is defined such as $E_0 = I$, $\mathbf{f}_0 = \mathbf{0}$ and its weight $\omega_0 = 1e^{-7}$. Finally, the simulations sampling time is set to $dt = 0.01\ s$. For simplicity, there is no tasks consistency transformation or normalization except in the last experiments, which implies that the matrices K_H and K_N defined in Sections 2.3.1.4–2.3.1.5 are equal to the identity matrix.

The three first sections of this chapter describe dynamic tasks and the interactions, illustrated through some examples. The fourth section deals with transitions while performing relatively complex sequences of tasks. The last section consists in the comparison of the different problem formalisms.

3.1. Performing multi-tasks

The following experiments describe in three separate sequences the virtual iCub standing up from a chair, walking, and grabbing a box using a weighing strategy. They are used as standard sequences for more complicated experiments in the rest of the thesis.

3.1.1. Stand up

Here, the robot stands up from a seat. A contact is added between the seat and the pelvis (the buttocks), and four tasks are used to achieve the sequence :

– two for the postures, sitting T_{sit} and standing T_{stand},
– one for the spine T_{spine},
– one for the CoM T_{CoM}.

The sequence [2] is illustrated in Fig. 3.2. At the beginning the robot is sitting, so the contact with the seat is established. ω_{sit} (the weight related to the sitting posture task T_{sit}) is equal to $1e^{-3}$, and ω_{stand} (related to T_{stand}) is equal to 0. As the robot is idle, the CoM is not controlled, and the weight of its task ω_{CoM} is null. Notice that the spine task which maintains the torso joints in the same positions with a constant weight $\omega_{spine} = 1e^{-2}$ is added to stiffen the upper body and to displace the trade-off effects (due to the weighting strategy) in the other joints. Finally, all these tasks use proportional derivative controllers (denoted by PD, see Section 2.3.4.1) whose gains are $K_p = 10\ s^{-2}$ and $K_d = 2\sqrt{K_p}\ s^{-1}$ (in the remaining of this thesis, if the value of K_d is not given, one must refer to this relation).

Then the order is given to stand up. In this purpose, the position of the CoM is controlled in the horizontal plane to reach the center of the base of support (*i.e.* the convex hull defined by the projection of the feet onto the ground). As it is the most important task of the sequence, its weight becomes $\omega_{CoM} = 1$. However, this sudden change in the control of the system may lead to discontinuities in the evolution of the

2. It corresponds to the movie `01_getting_up.avi` provided as an attachment to this thesis.

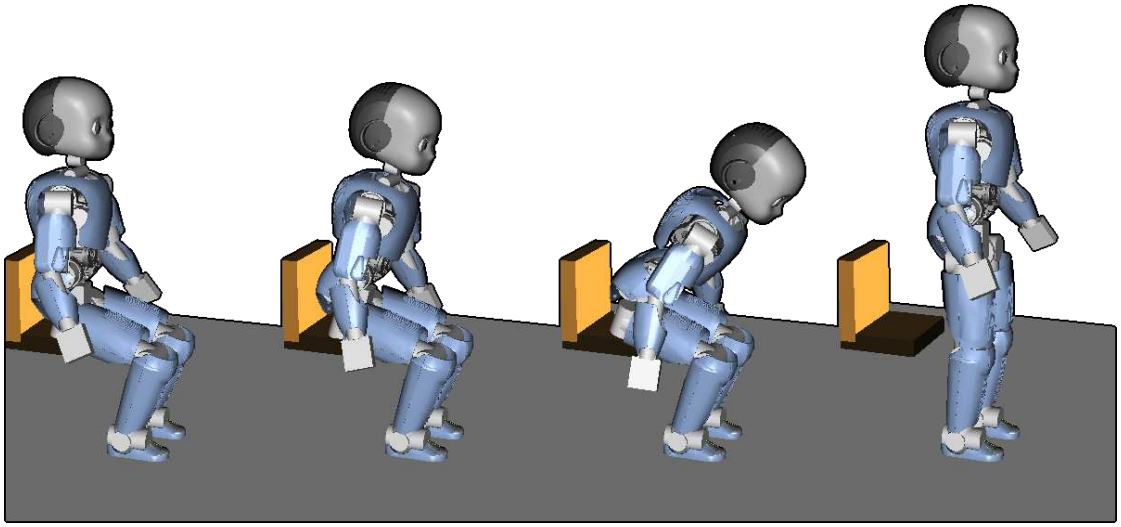


FIGURE 3.2.: Sequence "stand up" : 1-the robot is idle on the seat ; 2-it moves its CoM ; 3-the CoM is in the base of support, the contacts are broken and it starts to stand up ; 4-it stands upright and becomes idle again.

input torque vector, a bad behavior for real systems. This issue can be overcome in two ways. The costs of the CoM task at the previous and the next times have to be equal to 0, so either the error or the weight ω_{CoM} must be null. The first case means that the target of the task starts at the actual position of the CoM, and goes progressively toward the middle of the feet. In the second case, ω_{CoM} starts at 0 and increases progressively to 1. Here, we choose the second solution, and the evolution of ω_{CoM} follows Eq.(2.50) with a transitional time $t_s - t_e = 0.25\ s$.

Hence, the robot moves its upper body forward to reach a state which does not lead to the fall when it rises. From a dynamic point of view, this assertion is difficult to qualify, and the ZMP cannot be defined due to the fact that the contact points are not coplanar. So we just look for the static equilibrium. The contact is deleted when the projection of the CoM gets into the convex hull, the acceleration constraints at the pelvis level is released, and then, the robot can go from a sitting posture to a standing posture. It results in a continuous change of their respective importances where ω_{stand} and ω_{sit} go respectively to $1e^{-3}$ and 0 with a transitional time of 0.25 s as described above. Finally, after a short period of time, the robot reaches all the objectives, finishes the sequence and stands idle in front of the seat.

Figure 3.3 shows the evolution of the input torque vector with respect to time. It is continuous for the whole experiment except at time $t = .7\ s$ of the simulation, when the contact is broken. At this instant, a big gap between the previous and next states occurs. This issue is discussed in Section 3.4.2.

The weights of the tasks performed in this experiment are all less or equal to 1. Actually, the choice of these values is theoretically arbitrary as long as their relative importances are respected. Nevertheless, for the sake of normalization, we impose the task with the highest importance to have a unitary weight (the other tasks weights being set accordingly).

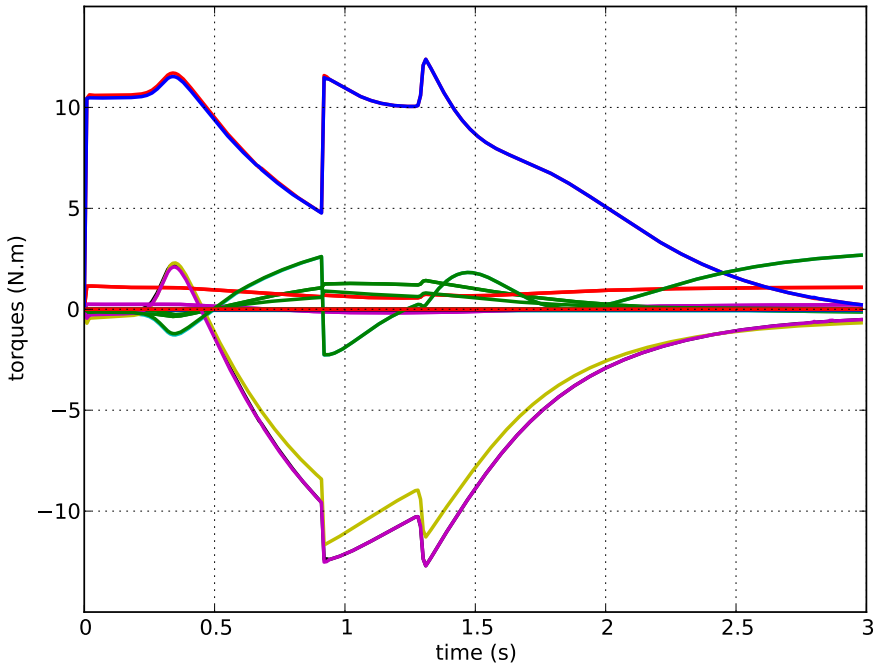


FIGURE 3.3.: Evolution of the input torque vector over time. At time $t \approx 0.9\ s$, the contacts are broken, leading to discontinuities.

3.1.2. Walk

Here, the robot has to walk for 0.5 meter and stop. The achievement of this mission relies on two important points, the CoM control with a quadratic controller defined in Section 2.3.4.2 which generates the oscillating movement and the forward motion, and the feet control along predefined trajectories. This sequence is composed of two tasks, a posture task T_{pos} ($\omega_{pos} = 1e^{-2}$) which controls the joints to stay around some reference positions, and a walk task T_{walk} ($\omega_{walk} = 1$) which controls the CoM and the feet.

As the goal of the latter is the ZMP control, its objective computation is not straightforward so we proceed as follows. First, one has to give the path from the starting to the final position in the horizontal plane, as well as the angular path during the displacement (for lateral or even back motions). So, given the fact that the longitudinal distance between two steps should be equal 0.08 m, the lateral distance of the feet should be 0.05 m and the maximal height of the steps should be 0.01 m. Then, a set of points and angles is extracted from these paths to specify the positions and orientations of the feet during the walk. This returns both the feet trajectories, whose control is done with a PD controller with a high stiffness ($K_p = 150\ s^{-2}$), and the reference ZMP trajectory. This last one is used to compute the desired CoM jerks at each instant (the parameters of Eq.(2.44) are set to $R = 1e^{-6}$ and $H = 1.7/dt$), and the desired CoM acceleration is obtained by integration.

The result of this simulation [3] is illustrated in Fig. 3.4, and the CoM and ZMP trajectories are presented in Fig. 3.5. But although the mission is properly achieved, the upper body of the robot oscillates too much, which is not close to the human gait [4].

3. Corresponds to Attachment `02_01_walking_only.avi`.
4. This comparison is not really possible because their controls and their actuations are completely

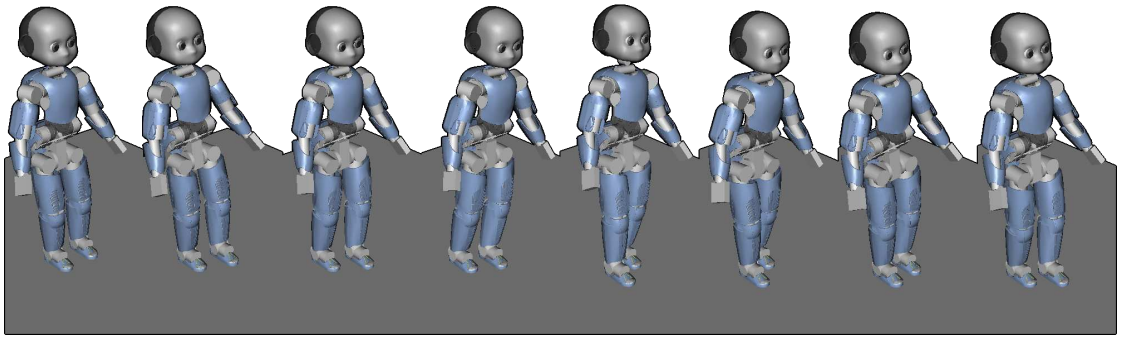


FIGURE 3.4.: Strip of the first sequence "walk" : the robot moves forward but the knees are not bent, the pelvis is not controlled and the upper body oscillates significantly.

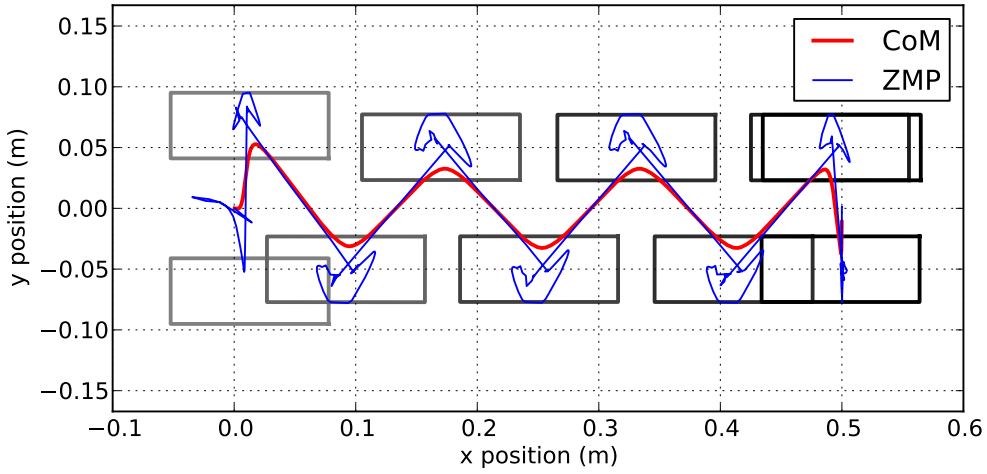


FIGURE 3.5.: CoM and ZMP trajectories during first sequence "walk".

In order to reduce the upper body oscillations, two new tasks are added, the first one to stiffen the spine T_{spine} ($\omega_{spine} = 1e^{-1}$) as described in the previous experiment, and the second one to control the altitude and orientation of the pelvis T_{pelvis} with $\omega_{pelvis} = 1$. The gait of the robot is slightly improved [5], as illustrated in Fig. 3.6, and the new CoM and ZMP trajectories are plotted in Fig. 3.7.

3.1.3. Grab

In this experiment, the objective is to grab a box located on a table ahead, whose mass is set to 0.1 kg. However, by simplifying the robot, the hands are modeled roughly without any finger, so the action of grasping is not possible. This issue is overcome by creating four bilateral closures between the hands and the box through the Arboris-python class `BallAndSocketConstraint`, whose activities (their closure states) are

different. For instance, humans use the energy stored in the tendons to maintain the gait cycle, whereas the robot motors do not possess this capacity. Therefore, one cannot reasonably expect the same behaviors.

5. Corresponds to Attachment `02_02_walking_with_root_and_back.avi`.

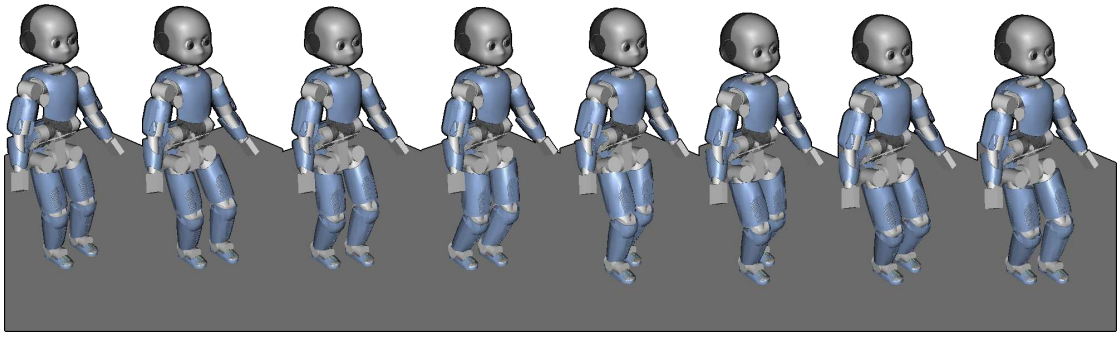


FIGURE 3.6.: Strip of the second sequence "walk" : here the knees are bent, the pelvis is controlled and the upper body remains straight.

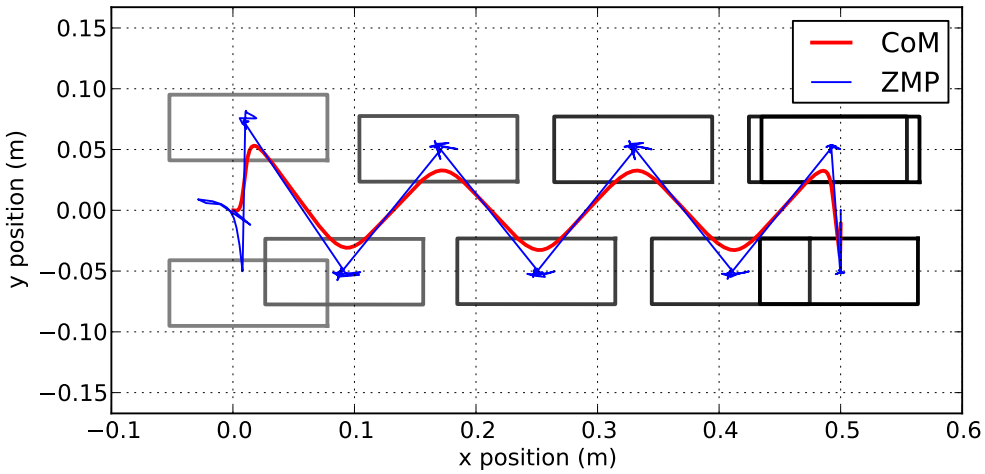


FIGURE 3.7.: CoM and ZMP trajectories during second sequence "walk".

set by the user. Here, the main purpose is to show the management of kinematic loops induced by grasping.

This sequence is composed of five tasks :

- one for the standing posture T_{stand} ($\omega_{stand} = 1e^{-4}$),
- one for the spine T_{spine} ($\omega_{spine} = 1e^{-2}$),
- one for the altitude and orientation of the pelvis T_{pelvis} ($\omega_{pelvis} = 1$),
- one for the CoM T_{CoM} ($\omega_{CoM} = 1$),
- and one for the hands T_{hand}.

The four first tasks are the same as defined in the previous experiments, and the last one controls the positions and orientations of the hands.

At the beginning, the robot is inactive, meaning that it just stands upright with $\omega_{hand} = 0$. Then the order is given to get the object, so this importance increases to $1e^{-1}$ in 0.25 s. The objectives of the related task are straight paths from the hands to the box, tracked with PD controllers with high stiffness $K_p = 200\ s^{-2}$. When this task is finished, the hands are located on the virtual handles of the box, so the bilateral closures can be activated, creating four virtual ball and socket joints thanks to the interaction forces. Finally, the robot lifts the box from the table and the mission is complete.

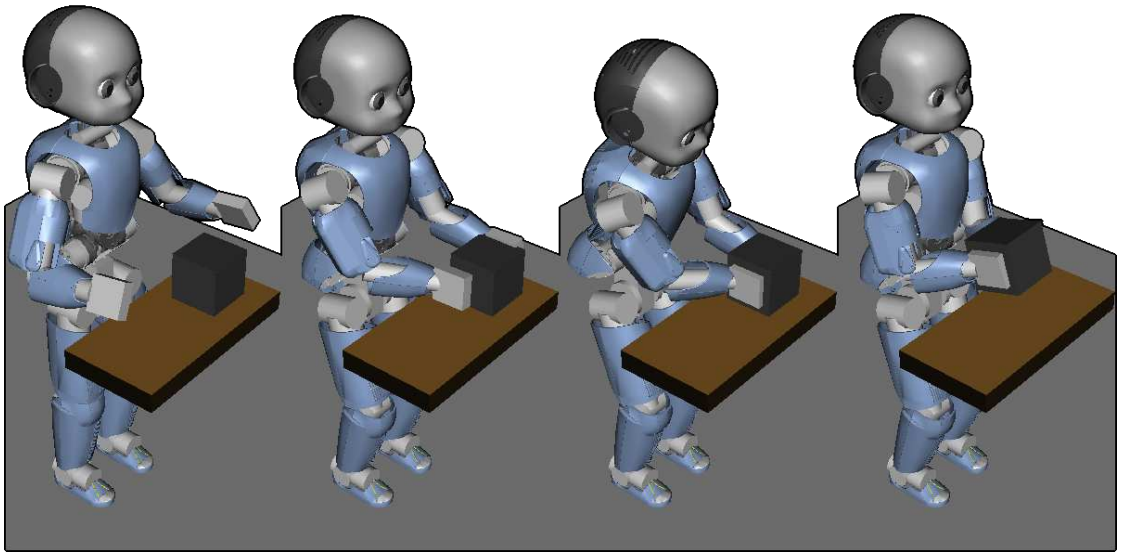


FIGURE 3.8.: Sequence "grab" : 1-the robot is idle in front of the box ; 2-the order is given, so it moves to grab the box ; 3-the kinematic loop closure is achieved ; 4-the robot grabs the box.

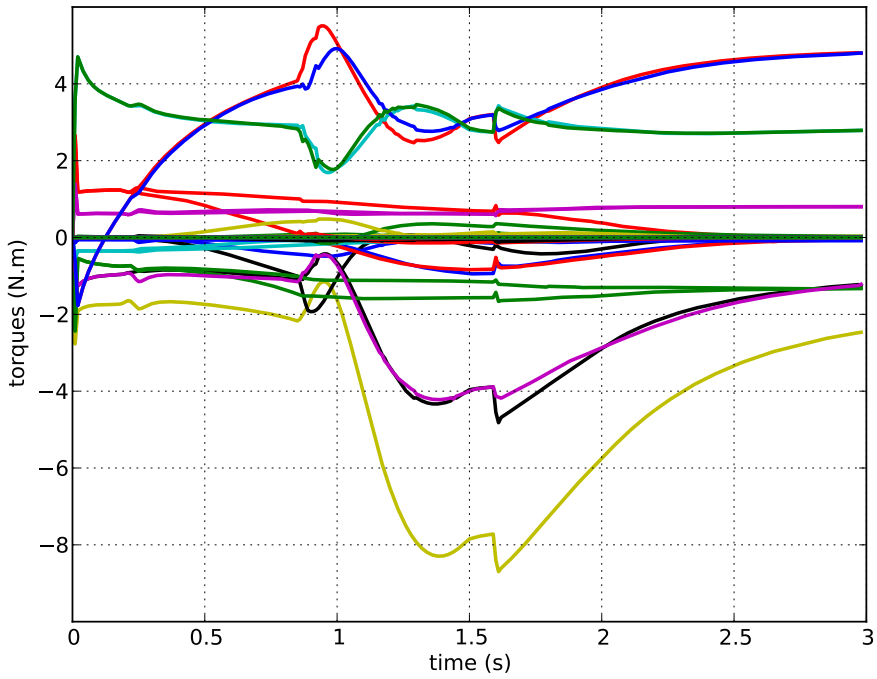


FIGURE 3.9.: Evolution of the input torque vector over time. Notice a discontinuity at time $t \approx 1.6\ s$ due to no management of the tasks transition.

The sequence[6] is described in Fig. 3.8, and Fig. 3.9 shows the evolution of the input torque vector applied to the robot. A discontinuity occurs when the robot grabs the box due to the fact that there is no management of the tasks transition, a problem discussed

6. Corresponds to Attachment `03_grabbing_a_box.avi`.

in Section 3.4.1.

3.2. Impedance controller for interaction

As mentioned in Section 2.3.4, the impedance control gives a method to unify the kinematic and force controls. It provides a way to deal with interactions which may occur in a changing environment where the robot behavior may be passive if it undergoes some disturbances, or active if it initiates or anticipates the actions (for instance if it has to pull, to push, etc.).

In this experiment, iCub stands in front of a moving table which prints a vertical oscillating motion, and eight contact points are active between the hands and the board. The mission is to reach some positions located in the space swept by the moving object, leading to some conflicting actions between the table and the robot when the contact is established.

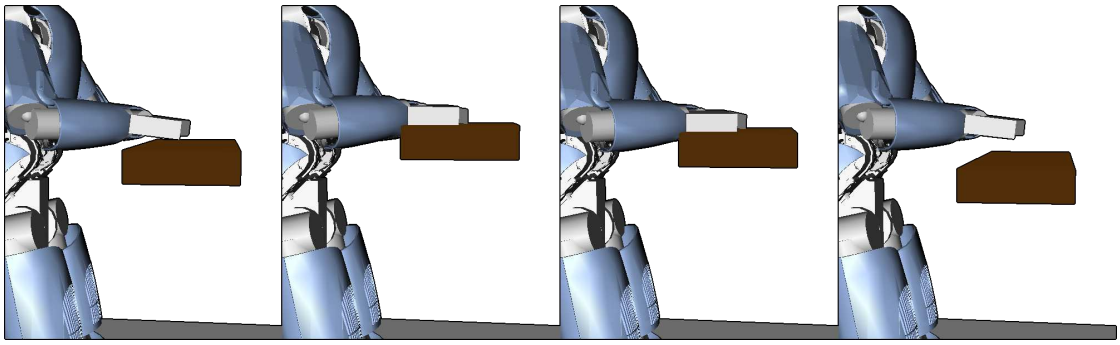


FIGURE 3.10.: Impedance control simulation : the board comes into contact with the hands which offer resistance, depending on the control stiffness. Then it goes down to track the sinusoidal reference trajectory, and the hands return to their objectives.

In the purpose of having some real bilateral efforts, the board is controlled dynamically. It weighs 1 kg and follows a sinusoidal reference trajectory with a period of 3 s and amplitude of 0.04 m. The tracking is done with a spring-damper system whose gains are given such as $K_p = 10\ s^{-2}$.

Besides, the robot has to perform the following tasks :

- one for the standing posture T_{stand} ($\omega_{stand} = 1e^{-4}$),
- one for the spine T_{spine} ($\omega_{spine} = 1$),
- one for the position and orientation of the pelvis T_{pelvis} ($\omega_{pelvis} = 1$),
- and one for the hands T_{hand} ($\omega_{hand} = 1e^{-1}$).

With this new set of weights, the robot stands upright while the effects of the disturbances due to the board principally act on the arms.

This experiment [7] is shown in Fig. 3.10. When the contact occurs, a force of interaction depending on the error of the task T_{hand} appears. Thus, the table tries to go further and slows due to a higher contact force, until it has to go down, so the hands return to their objectives with no disturbances. Unlike the previous experiments, the contacts are not considered as constraints any more, meaning that Eq.(2.10) is not taken into account in the LQP-based controller which "assumes" that the hands are free to move. Thus, some

7. Corresponds to Attachment `04_impedance_control.avi`.

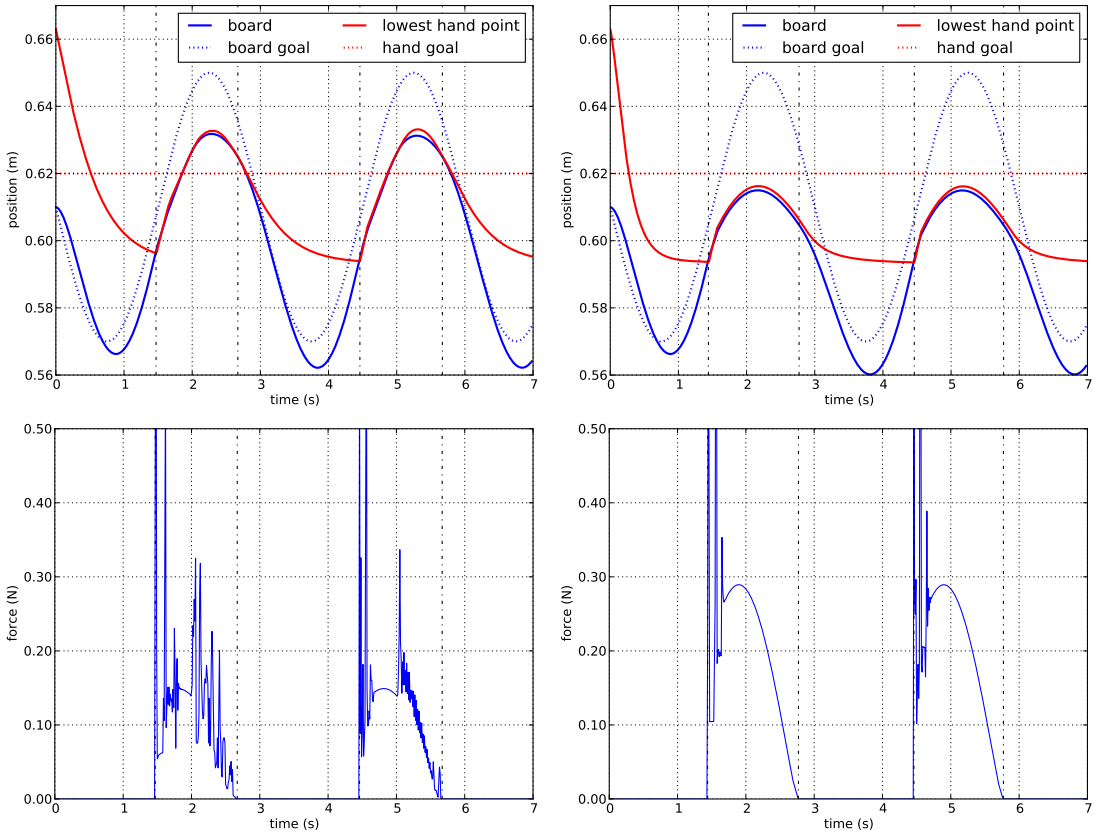


FIGURE 3.11.: Evolution of board and hands positions, as well as their objectives (top) & evolution of interaction forces at the contact level (down) – Left : The control stiffness is low, the tracking of the board trajectory has minor errors, and the hands take some time before returning to their goals. Right : The control stiffness is high, the tracking is worse due to more resistance, and the hands return faster to their objectives.

forces are produced by the hands during the interactions with the surrounding world, depending upon their kinematic objectives in a non-constrained environment.

The task T_{hand} uses a PD controller, and the behavior of the robot is analyzed for two different cases in Fig. 3.11. In the first one (left), the associated gains are set with $K_p = 10\ s^{-2}$, and in the second one (right), they are set with $K_p = 50\ s^{-2}$.

Notice that in the second case, the hands oppose greater resistance to the table and they take less time to return toward their goals. This illustrates the behavior of the robot subject to a disturbance depending on the impedance controller stiffness.

3.3. Obstacle avoidance

The previous section deals with the control of the robot while interacting with its environment, but sometimes the contacts should be avoided. Here, the experiments focus on the obstacle avoidance when the robot is operating in a constrained space. Two experiments are performed to illustrate the different cases, the first one where the objects are fixed and the robot has to move without touching them, and the second one where

they are moving and it has to react accordingly.

3.3.1. Goal reaching in presence of fixed obstacles

The robot has to reach a target located just in front of it with its left hand, but the straight path is blocked with some obstacles. For this experiment, the robot performs the following tasks :

- one for the standing posture T_{stand} ($\omega_{stand} = 1e^{-1}$),
- one for the spine T_{spine} ($\omega_{spine} = 1$),
- one for the position and orientation of the pelvis T_{pelvis} ($\omega_{pelvis} = 1$),
- one for the position and orientation of the left hand T_{hand} ($\omega_{hand} = 1$).

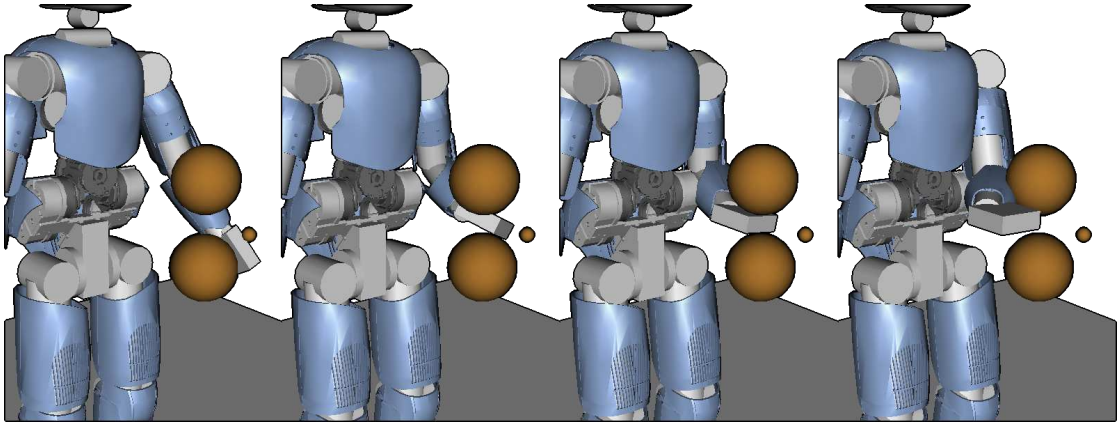


FIGURE 3.12.: Fixed obstacle avoidance simulation : 1-the left hand must reach a objective located in front of the chest ; 2-it gets closer to the obstacles and slows down due to the associated constraints ; 3-the available space is large enough, the hand can pass ; 4-the hand reaches its objective.

The hand is attracted toward the goal with the PD controller whose stiffness is $K_p = 10\ s^{-2}$, and when it gets closer to the obstacles the motion is constrained due to Eq.(2.7), and the path is reactively modified. The sequence [8] is shown in Fig. 3.12.

However, Fig. 3.14 shows that this method does not guarantee to find a feasible path to reach the goal [9], the hand is blocked by too many obstacles and cannot reactively get out of this situation. This problem is addressed by more global approaches, for example path planning.

3.3.2. Avoiding moving obstacles

Here, the scenario is based on the impedance control experiment with the same tasks (see Section 3.2), but this time the mission of the robot is to avoid any contact with the board while this one moves near the hands objectives. The path of the moving obstacle is not known *a priori*, but the instantaneous position and velocity are supposed to be observable, hence it is possible to compute the equation of obstacle avoidance Eq.(2.7) and prevent any collision with the moving object. The resulting sequence [10] is shown in

8. Corresponds to Attachment `05_01_obstacle_avoidance.avi`.
9. Corresponds to Attachments `05_02_hand_blocked.avi` & `05_02bis_hand_blocked.avi`.
10. Corresponds to Attachment `05_03_moving_obstacle.avi`.

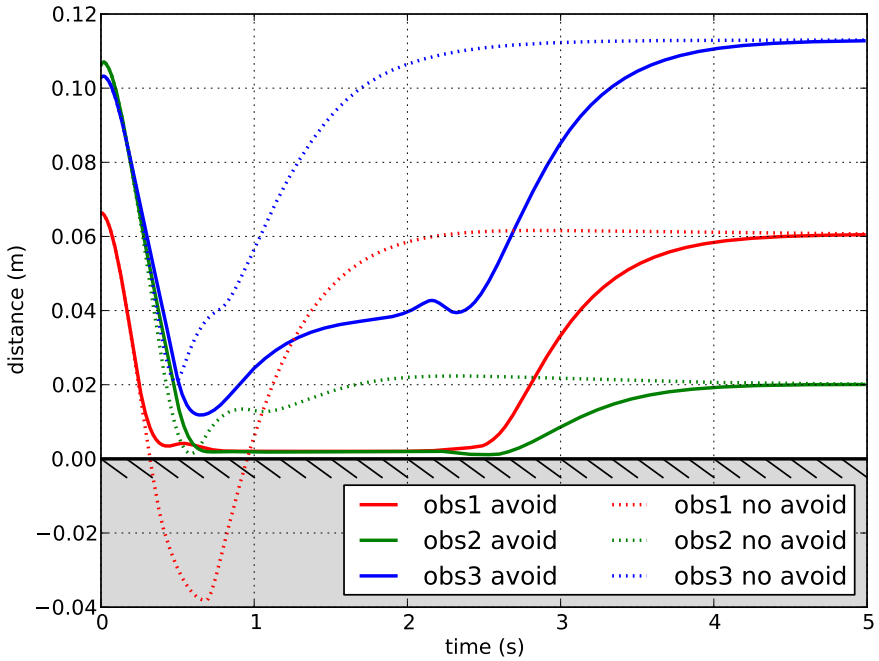


FIGURE 3.13.: Evolution of the minimal distance between the hand and the obstacles. Without avoidance, interpenetration occurs at time $t \approx 0.2\ s$, otherwise the minimal distance remains positive (with a margin of $0.002\ m$).

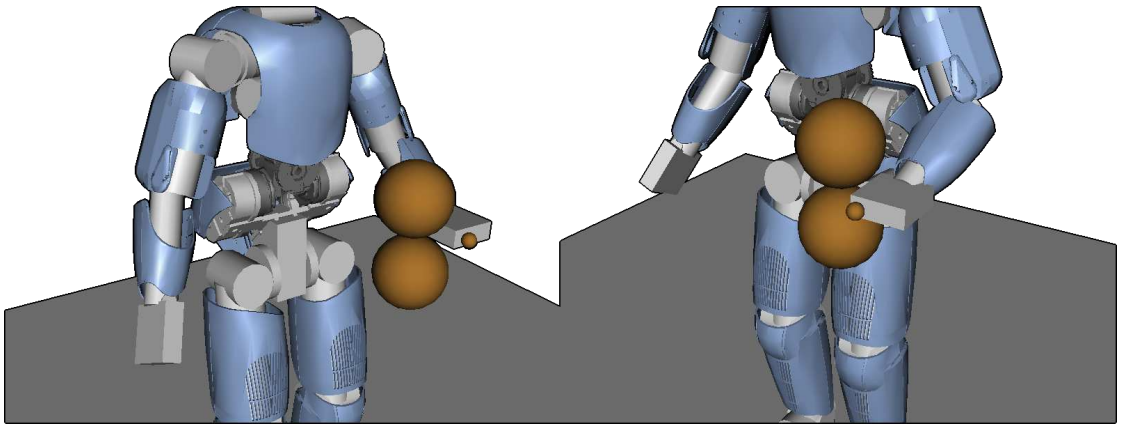


FIGURE 3.14.: Fixed obstacle avoidance simulation : because of a local unfeasible path, the hand is stuck and cannot get out reactively.

Fig. 3.15, and the behavior of the robot is tuned thanks to anticipation coefficient h_3 as illustrated in Fig. 3.16.

3.4. Transitions in the tasks and constraints sets

The previous section shows that the dynamic control of humanoid robots interacting or not with the environment is possible thanks to the use of a LQP-based controller and a weighting strategy. But even if the sequences previously described provide some

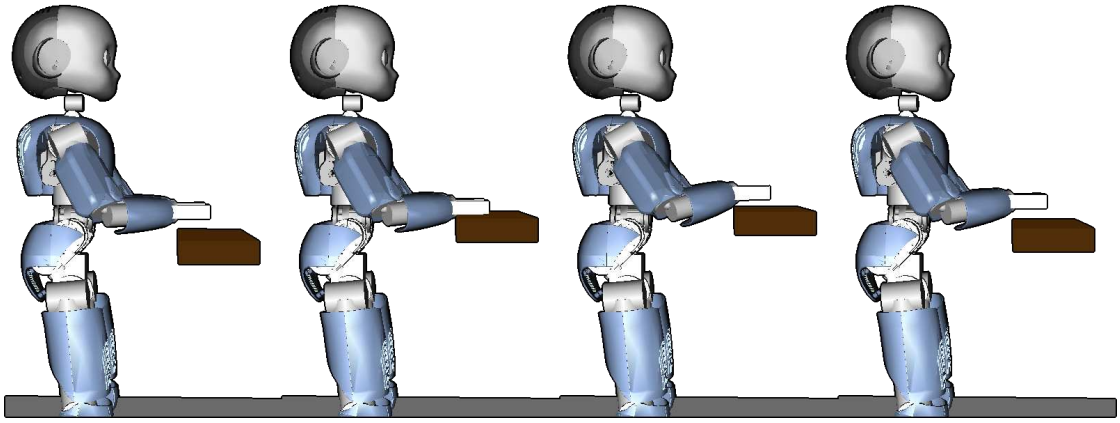


FIGURE 3.15.: Collision avoidance simulation : like impedance control simulation, the board follows a sinusoidal trajectory. Unlike the latter, the robot avoids contact and gets closer to its goal when the table moves away.

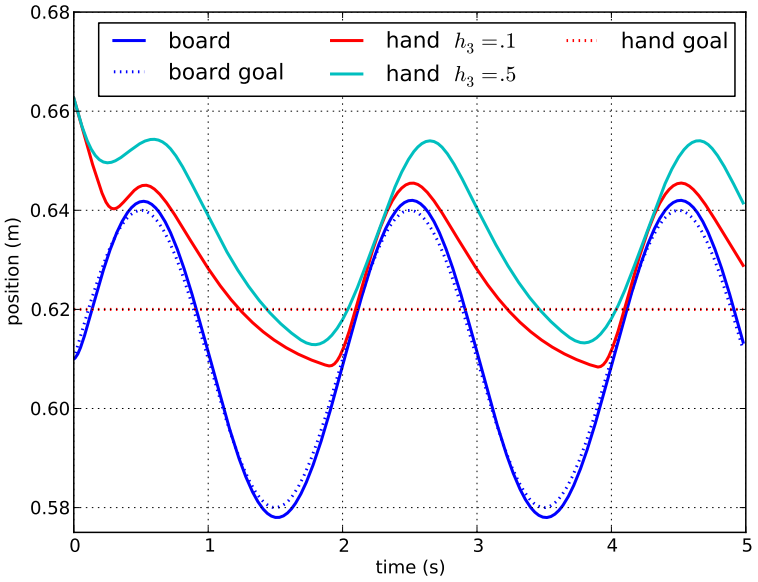


FIGURE 3.16.: Evolution of board and hands positions, as well as their objectives. The hands behavior changes according to anticipation coefficient h_3.

interesting behaviors, the input torque vector is not always suitable for real mechanical systems, mainly due to bad management of transitions in the control scheme. This section addresses the problem of transitions in the set of tasks and constraints described in Section 2.3.6.

3.4.1. Transition in the tasks set

The context is the same as the one proposed in the grabbing experiment detailed above, extended with some other tasks to perform more significant transitions. Here, the scenario is described in Fig. 3.17 where iCub catches a box and moves it to another

position [11].

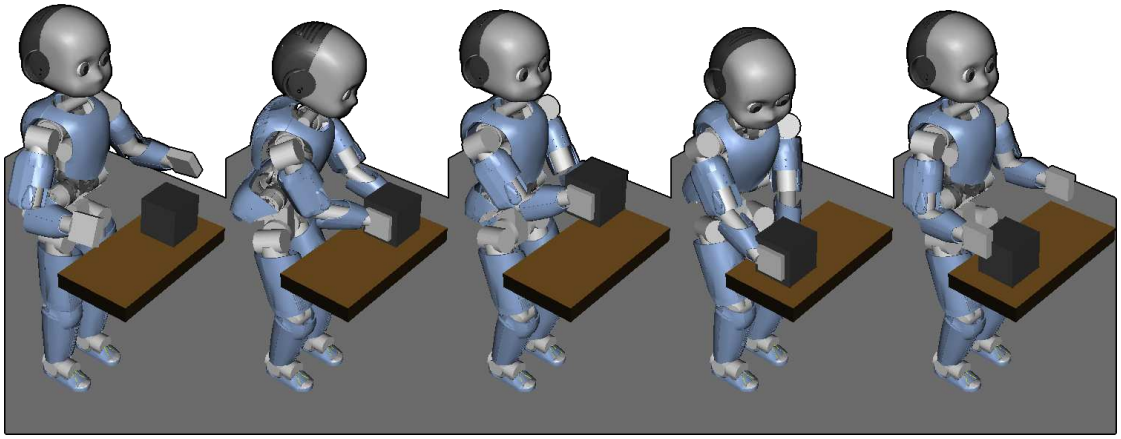


FIGURE 3.17.: Sequence "grab" extended with action "drop" : the robot bends over to grab the box on the left side. It straightens up and brings it at the chest level, then drops it on the right side, and becomes idle again.

A comparison is made between the hierarchical strategy (where importances are integer values) and the weighing strategy (where importances are real values), to show the benefits of the tasks weights during transitions in the set of tasks. Here, each task importance is denoted by a couple representing first the weight of the weighting strategy, and second the priority value of the hierarchy [12]. In both cases, six tasks are used. Three of them are constant, which are :

- the standing posture T_{stand} ($\omega_{stand} = (1e^{-4} \mid 4)$),
- the altitude and orientation of the pelvis T_{pelvis} ($\omega_{pelvis} = (1. \mid 0)$),
- the position of the CoM T_{CoM} ($\omega_{CoM} = (1. \mid 0)$),

and the three other ones are :

- the spine T_{spine},
- the hands T_{hand},
- the position of the box T_{box}.

The importances of T_{spine}, T_{hand} and T_{box} must evolve over time, otherwise they cannot reach their objectives and the displacement of the box cannot be completed. The following of this section focuses on these three tasks.

The experiment is conducted as follows. At first the robot is idle, it has to stand upright without looking for interaction with the box, so the relative importance of the spine task is $\omega_{spine} = (1e^{-1} \mid 1)$, and the others are not active. When the sequence begins, the robot needs to bend over to achieve the box grabbing which requires a modification of the relative tasks importances, so ω_{spine} becomes equal to $(1e^{-3} \mid 3)$ and the hands task is activated with $\omega_{hand} = (1e^{-1} \mid 1)$ (in the case of the weighting strategy, these values are modified in a continuous manner over a time horizon of 0.5 s). The hands have to reach the box, and the achievement of the related task takes about 1.5 s. When they are close enough to their targets, the bilateral closures which simulate the grasping are activated. Then, the robot straightens up while lifting the box, and thanks to the new kinematic loops, the motion of the box can be controlled through a task. The hands task

11. Corresponds to Attachment `06_transition_tasks.avi`.
12. The tasks of the hierarchical strategy are solved in ascending order.

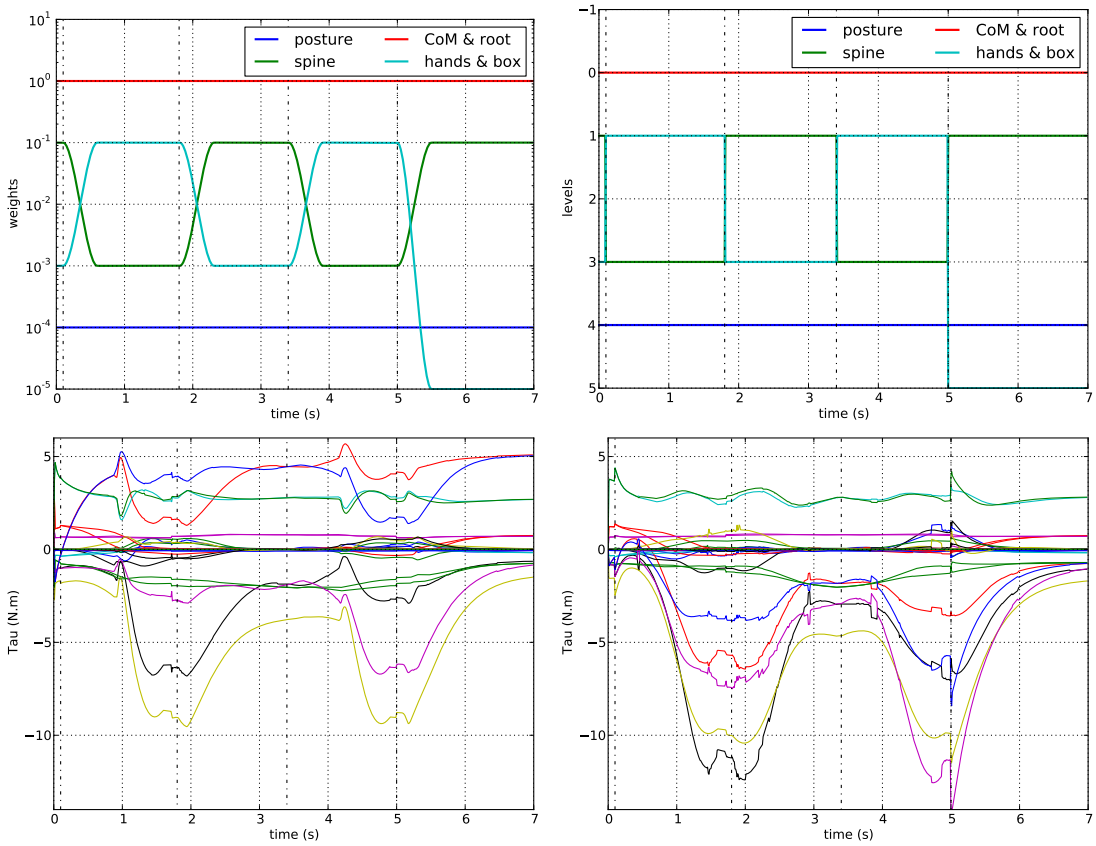


FIGURE 3.18.: Evolution of tasks importances (top) & evolution input torque vector over time (down) – Left : The weighing strategy generates smooth transitions in the set of importances, leading to a continuous torque vector and a proper control. Right : The hierarchy leads to discontinuous transitions, which affects the evolution of the input torque vector.

is deleted and the new importances become $\omega_{spine} = (1e^{-1} \mid 1)$, and $\omega_{box} = (1e^{-3} \mid 3)$. Again, the transition is smooth for the weighting strategy over a time horizon of 0.5 s. When the robot brings the box at the chest, it waits a moment and drops it at the other side of the table, so it has to bend over again and the importances of the tasks change in the same way. Finally, at the end of the mission, the robot releases the box, the bilateral closures are deactivated and it becomes idle again.

The change in the set of importances is illustrated on Fig. 3.18 (on top) in the case of weighting strategy (left), and hierarchical strategy (right). Their resulting input torque vectors are respectively shown on Fig. 3.18 (at the bottom). Some sharp evolutions occur with the hierarchy, this illustrates the advantages of using smooth transitions in the set of tasks to obtain good control continuity.

3.4.2. Transition in the constraints set

The previous section focuses on the transitions in the set of tasks, this one focuses on the sequencing of tasks inducing a transition between constraint and non-constraint states. Here, the experiment is based on the scenario where the robot stands up from a

seat described in Section 3.1.1, with the same set of tasks. Actually, iCub has to break the contact with the chair to stand up, so it leads to some modifications in the constraints set when the contact vanishes and some control discontinuity. Thus, the purpose of the following experiment is to develop a strategy to avoid any sharp evolution of the input torque vector during a change in the set of constraints. Two ways are compared, the first one uses weights to perform the transitions, the second one uses a constraint on the time derivative of the torque.

In the first case, two new tasks associated to the contact with the seat are added to the set of tasks, which are :

– one for the contact acceleration $T_{c_{acc}}$,
– one for the contact force $T_{c_{force}}$.

The transition has to be performed with the method explained in Section 2.3.6.2. Initially, the task $T_{c_{acc}}$ has a high weight $\omega_{c_{acc}} = 1$ with null desired acceleration at the contact point [13], and the task $T_{c_{force}}$ has no weight $\omega_{c_{force}} = 0$ with a desired null force. The beginning of the simulation is similar to the previous one, the difference occurs when the projection of the CoM onto the ground enters in the base of support. At this instant, the importance $\omega_{c_{force}}$ increases up to 1 over a time horizon of 0.5 s, which preserves the sitting position but decreases the contact force down to 0 N. Then, the contact can be broken without big consequences on the input torque vector. The importance of the task related to the acceleration at the contact point level goes down to 0 in 0.5 s, and finally the robot becomes completely free to stand up from the chair. This method is possible only with the weighting strategy. Figure 3.19 shows the evolution of the input torque vector during this simulation. The sharp transition occurring in Fig. 3.3 is deleted, which allows a better control of the robot.

In the second case, the management of the transition is done by bounding the time derivative of the torque vector. Here, when the CoM enters into the base of support, The contact constraint is deleted the same way as the first experiment (see Section 3.1.1), but the transition is ensured with Eq.(2.49). The result is illustrated in Fig. 3.20.

Of course, these methods also work when a contact appears, it suffices to run the processes backward. Furthermore, the two methods described above are relatively complementary and can be used simultaneously without any problem, the first one allowing more active monitoring of the transition, and the second one more passive management.

3.5. Comparison between the different task types

The method described in this thesis is used to perform many missions, but the way to achieve them is not unique. An interesting point is the control performance in terms of computation time, because the computation of homogeneity, normalization and especially the formalism used in the control scheme may change the performance of the control.

3.5.1. Comparison between the two formalisms

Here, the two formalisms exposed in Section 2.3.2 are compared. The first one uses the overabundant dynamic variable $\mathbb{X}$ and the second one relies on the independent

13. For now, this acceleration task has no effect on the control, since the contact constraint is still active.

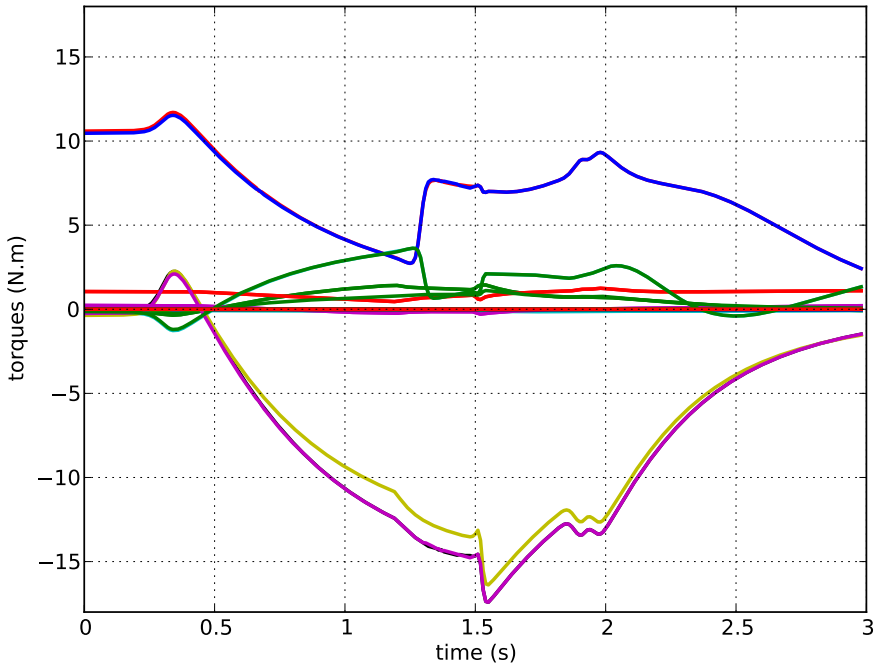


FIGURE 3.19.: Evolution of the input torque vector. The continuity is ensured with soft transitions, even when the contacts are lost.

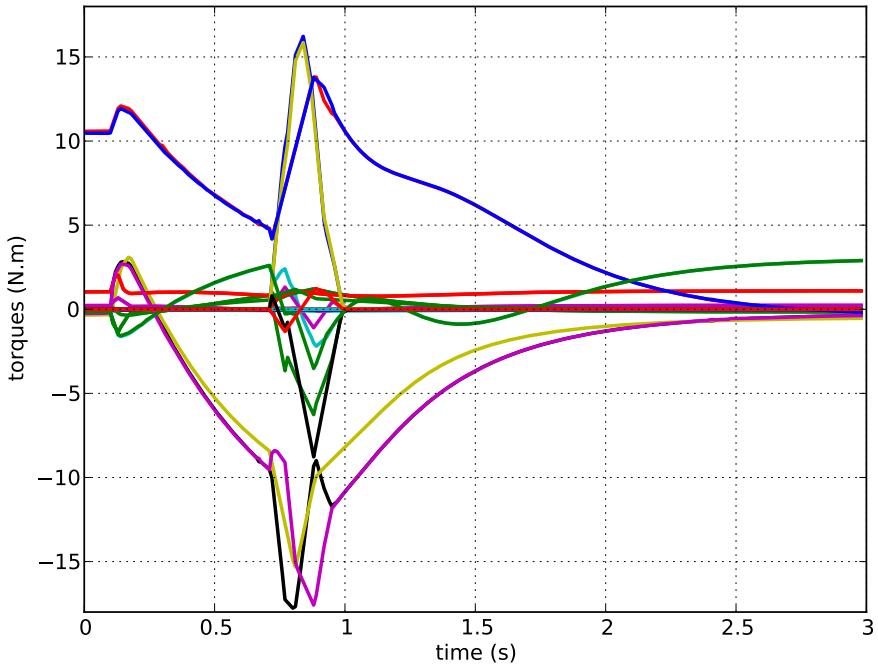


FIGURE 3.20.: Evolution of the input torque vector. Even if the continuity is ensured by constraints, the variations are relatively sharp.

action variable $\boldsymbol{\chi}$. However, although the dimension of the problem is bigger using $\mathbb{X}$, the writing of the cost function and the constraints is more complicated with $\boldsymbol{\chi}$ and

may spend more computing resources. Besides, due to the fact that the solver uses the barrier method (explained in Section 2.2.3.1), some errors may occur between the two problems. Actually, the tuning of the tolerance associated to this method is made through the solver parameters `abstol`, `reltol` and `feastol`, which are respectively the absolute, relative and feasible tolerance of the solution. So, in order to highlight the differences between the formalisms, the following of this section compares the time spent by the control loop depending on solver tolerances.

Three experiments are repeated to confront the performances of the LQP-based controller. The first one is the standing up simulation with transitional tasks explained in Section 3.4.2, the second one is the walking sequence with root task described in Section 3.1.2, and the last one is the grabbing simulation of Section 3.1.3. The average solving time is extracted for each sequence, simulated with the two formalisms and for a tolerance which goes from $1e^{-6}$ to $1e^{-14}$.

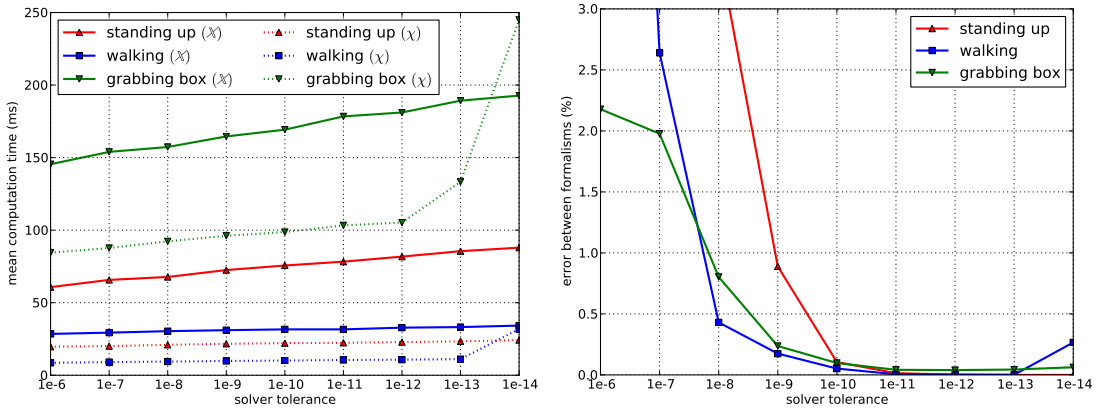


FIGURE 3.21.: Comparison between the two formalisms when performing different sequences, depending on the solver tolerance – Left : average computation time . Right : average torque error.

The results are exposed on Fig. 3.21 where the left part shows the solving times of the control loop and the right part gives the errors between the two formalisms. Using $\boldsymbol{\chi}$ is usually faster than using $\mathbb{X}$, and the results are quite similar for low tolerances (less than 1% for a tolerance of $1e^{-10}$). Hence, it seems very interesting to use $\boldsymbol{\chi}$. A more complete comparison is done in the next experiment.

3.5.2. Comparison with the special case

Finally, this last experiment of the low-level control is the comparison of the performances when one employs the expression of the special case, that is the task space dynamics representation explained in Section 2.3.3. To do so, the simulations used in the previous experiments are repeated here, but the homogenization and normalization matrices K_H, K_N introduced in Sections 2.3.1.4–2.3.1.5 are not equal to identity any more. Four cases are taken into account :

1. $K_H = I$, $K_N = I$, $\boldsymbol{x} = \mathbb{X}$,
2. $K_H = I$, $K_N = I$, $\boldsymbol{x} = \boldsymbol{\chi}$,
3. $K_H = \Lambda$, $K_N = \sqrt{\Lambda^{-1}}$, $\boldsymbol{x} = \mathbb{X}$,

4. $K_H = \Lambda$, $K_N = \sqrt{\Lambda^{-1}}$, $\boldsymbol{x} = \boldsymbol{\chi}$,

where Λ is the projection of the mass matrix in the related task space. The first case represents the previous problem, the third and fourth ones are the task space dynamics representation. The simulations are run with the same parameters, and the tolerance is set to $1e^{-10}$.

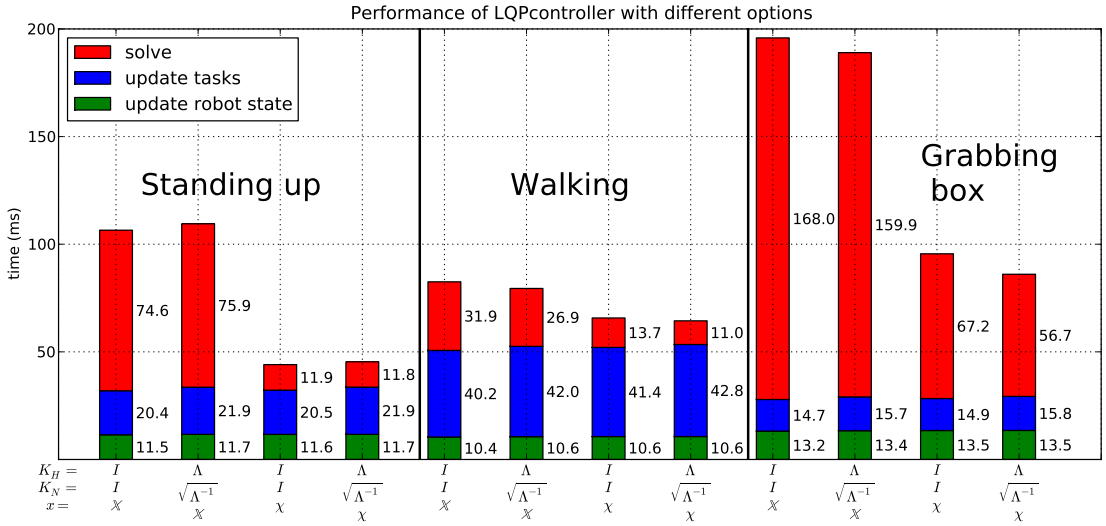


FIGURE 3.22.: Average computation time for different simulations in the four cases described in this section. Each bar represents the time of the control loop which is divided into three subparts : 1-update of the robot state, in green, mainly to prepare the problem matrices (the Jacobian of contact, the inverted mass matrix, the constant matrices for formalisms, etc.) ; 2-update of the task, in blue, to build the problem and obtain E and $\mathbf{f}$ for each task. 3-solve, in red, to format the problem, delete the redundant constraints, and obtain the input torque vector by solving LQP.

Generally, the computation time grows when the dimension of the problem increases, *i.e.* when the number of interaction forces increases. Notice also that even if the task space dynamics representation is more complicated, the computation time of the whole control loop is equivalent to the normal one. Finally, the comparison between these two representations is quite difficult to conduct because it does not lead to significant differences in the robot behavior, even if the first one is physically consistent.

3.6. Conclusion

This chapter presents some experiments on the control of the humanoid robot iCub with the LQP-based controller developed in the previous chapter. The first part focuses on low-level control which mainly deals with the computation of the optimal input torque vector depending upon the sets of tasks and constraints. This allows to perform some sequences with several (possibly conflicting) tasks, for instance “to stand up”, “to walk”, “to grab”. Besides, it illustrates the use of interesting controllers such as an impedance controller, and interesting constraints, for example related to the obstacle and collision avoidance. Then, the transitions in the set of tasks and constraints are discussed, showing the benefits provided by a weighting strategy. Finally, the computation times of this

LQP-based control loop are recorded in several simulations to compare the sequence and the two formalisms presented in Section 2.3.2.

4. High-Level control : coupling a decision making engine

Contents

The main drawback of the technique presented in the previous chapter to perform transitions between tasks (through the computation of Eq.(2.50)) lies in the fact that the choice of the weights is left to the programmer. This is a strong limitation regarding the complexity of the missions/goals that can be achieved. The same applies to hierarchy-based approaches where the choice of the hierarchy itself is also left to the programmer. So one needs a high level controller whose role would be to reactively elect tasks of interest based on contextual information. These tasks and their associated relative weights could then be accounted for at a lower, physics related control level.

Even if similar concerns have been tackled from an optimal control point of view in [Li2006], ways to combine such low and high level controllers have not been extensively explored in humanoid robotics. A recent work in this domain is proposed in [Philippsen2009] which introduces an architecture combining a *Whole-Body Control* (low) level and a *Reactive Symbolic Planning* (high) level through what is called a *Mobile Manipulation Database* and whose goal is to deal with information which is relevant to all components of the system. However, this architecture has not been tested as a whole yet and some works remain to be done. Other works such as in [Yoshida2005] offer dedicated mission-level planning methods for humanoids which are coupled, in an iterative fashion, to task-level controllers. Reactive decisions are only taken at the task level and the mission-level planner does not aim at playing the role of a controller, *i.e.* take decision based on control goals and perceptual feedback. The work of [Cambon2009] also approaches the problem from a planning perspective. It explicitly builds plans that relate

the mission with the motion levels, and where preconditions and effects are described at a geometric level. The last feature is only compatible with quasi-static worlds and does not fit the humanoid robotics framework where dynamics is intrinsically present in simple tasks such as "Standing".

An alternative paradigm is presented in this chapter which proposes a method to combine a low-level tasks controller (the LQP-based one presented in Section 2.3), able to deal with highly complex and constrained systems at the dynamics level, with a higher level decision making controller, dealing mostly with symbolic representations of what a robotic mission is. The contribution of the proposed work has to be interpreted from an "action selection" perspective, where no explicit plan is constructed, and where a mission is described in terms of high level goal, *e.g.* "Bring object O_2 to location L_1". This goal, associated to contextual information (percepts), *e.g.* "Object O_1, O_2 and O_3 are reachable", "Location L_1 is x meters far", and rules, *e.g.* "If target location is further than x meters, walk toward it", allow the triggering of high level actions, *e.g.* "Walk", "Grab", "Sit" which are a blend of lower level tasks : "control of the center of mass", "control of the right foot", "posture control"... This formulation allows the application of a rule-based high level controller in order to decide at each time which actions should be performed given a goal to reach and various percepts. This decision is automatically translated in terms of tasks instances along with context related parameters and relative weights (the LQP weights).

This work illustrates the use of such a high level controller in combination with a powerful task-level controller in environments where complex whole-body humanoid behaviors cannot be planned in a deterministic manner because of the involved dynamics and uncertainties, and thus have to be automatically and reactively generated in order to fulfill complex missions.

In this section, the first part is dedicated to a rather general description of the goal driven controller and of its implementation with the SpirOps AI software [Spirops]. The second part presents the results illustrating the versatility of such an approach.

4.1. Decision making engine

This section focuses on the decision making engine used to perform high-level complex behaviors in non-deterministic environments. Its aim is not to expose new methods that may make a significant breakthrough in the artificial intelligence field, but to present a feasible and suitable connection between a high-level controller and the LQP-based one using a weighting strategy. For that, it relies on some perceptual information to change the weights accordingly and achieve complex scenarii. The following sections present this connection.

4.1.1. Perceptual information

Achieving complex missions requires the blending of several tasks. This blending problem can be formulated at each time as : given a goal to achieve and a perceptual context, which task should be activated and what should be its relative weight with respect to others ? As shown in the previous experiments of Section 3, this can be done manually for rather simple missions and leads to rather satisfactory results when it exists one simple solution.

However, such a manual method finds its limitation with increasingly complex missions, and finding *a priori* the correct combination and sequence of tasks is even not possible when dynamic external events can occur during the mission. As an example, avoiding a moving obstacle cannot be planned *a priori*, as the obstacle trajectory is not known in advance and thus needs to be triggered reactively in accordance with the relevant percepts. Also, one may wish to specify a mission in terms of high-level goals : "Go to this location", "Grab this box", "Sit on this object"... without having to specify intermediate goals such as : "first Stand, then Walk and finally Grab".

The existence of a higher level decision making problem is obvious and a potential solution has to be able to decide reactively what are the robot actions to trigger and with which interests, based on :

- contextual information issued from the perceptions ("The box is close enough to be grabbed", "The location has been reached", "The object is too high to be sat on", ...) ;
- the higher level goal which is pursued ("Grab the red box"...).

A quite general decision making method that takes into account these data is described hereafter.

4.1.2. Decision making process

At the lowest level, the set of n_t **tasks** T which can potentially be assigned to the robot is defined. On one hand, the tasks can be operational frame control tasks. In the specific case of this work, 8 potentially interesting frames attached to the following parts of the humanoid bodies are retained : the CoM, the head, the pelvis, the gluteal, the left and right hands and the two feet. On the other hand, tasks can also be related to specific human body postures and in this work, 2 postures of interest are retained : sitting and standing. Some tasks may act on the same parts of the robots, but their goals are different. This set of tasks can be combined together in a physically consistent manner using the LQP formulation described in Section 2.3.5.

At an intermediate level, the set of n_a **actions** A is defined as some predefined tasks combinations. The actions refer to the verbs, "Walk", "Balance", "Grab", "Lift", "Stand up", "Sit" or "Stand"... and rely on the blending of the tasks which can control the CoM, pelvis, hand and feet. A given action is connected to a set of "children" tasks (but each task has only one "parent" action), and its role is to convert the high-level goal into objectives for its children, and to coordinate their executions. The blending of these tasks is described using **constant blending factors** $\beta_{i,j}$ whose values settle the importance of the achievement of task T_i to obtain action A_j.

At the highest level, a set of n_g **goals** G is defined in accordance to the objective of the mission. Each goal G_k acts on one and only one action A_j, and the **Desire** $\delta_{j,k}$ to perform this action and achieve this goal is the combination of its **adequacy** $\alpha_{j,k} \in [0, 1]$ ("I want") and its **opportunity** $\theta_{j,k} \in [0, 1]$ ("I can"). It is computed as $\delta_{j,k} = \alpha_{j,k}.\theta_{j,k}$. Although each goal has a unique "child" action, one action may have several "parent" goals.

Hence, the purpose of the decision making engine is to select the goals with the maximum desire but with exclusive actions. In other words, for one action A_j, it selects the "parent" goal G_k with the best desire as written in Eq.(4.1). Then, the goal parameters are transmitted to the action (*e.g.* the location to reach, the colored box to grab) which updates the goals of its "children" tasks T_i and computes their weights ω_i in Eq.(4.2).

$$\delta_j^* = \max_k(\delta_{j,k}) \tag{4.1}$$

$$\omega_i = \beta_{i,j}\delta_j^*. \tag{4.2}$$

To illustrate this method (see Fig. 4.1), one can imagine two goals which are G_y "Grab the yellow box" and G_b "Grab the blue box". They both use the action A_g "Grab" which controls the two tasks T_l, T_r related to the left and right hands. The decision making engine computes the desires $\delta_{g,y}, \delta_{g,b}$ to perform the action A_g, chooses the higher one (say $\delta_{g,y}^*$), collects the chosen goal parameters and spreads the objectives to the children tasks with the weight $(T_l, \beta_{l,g}\delta_{g,y}^*), (T_r, \beta_{r,g}\delta_{g,y}^*)$, then performed by the LQP-based controller. With this technique, the function $argmax$ is called n_a $(< n_g)$ times on lists whose dimension is less than n_g, and n_g desires are computed. Thus, the complexity of this method mainly depends on the number of independent actions specified for the robot.

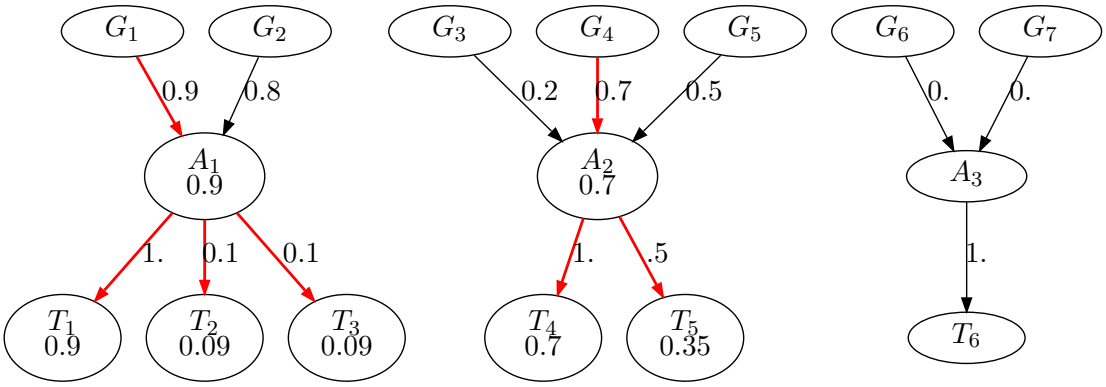


FIGURE 4.1.: Graph illustrating the decision process from the goals to the tasks. The red arrows link the elected goals (top), actions (middle) and tasks (bottom), with their respective weights. The value embedded in each task is its respective weight used during low-level control.

In the framework of this thesis, the computation of the desires is left to the **SpirOps AI** decision making toolbox which is described in the next section. SpirOps also provides to the programmer a framework for the description of rules, key percepts and action parameters. The choice of the blending factors is complex. It requires for example to answer to the following question : given a mission context, what is the importance of the CoM control with respect to the feet in a "Balancing" action ? The setting of these factors could probably be automated but this would require the use of machine learning techniques which is out of scope of the present work. As a matter of fact, this type of tuning is left to the experience of the programmer.

4.1.3. Introduction to SpirOps : a decision making toolbox

SpirOps AI tools [Spirops] provide an open environment to easily and incrementally design autonomous decision processes. A graphical editor is used to edit independent *Goals*. Minimal C++ code is needed to plug the decision process into the host application. This decision process pulls data from the host application as lists of typed objects

described by attributes, including the entity that uses the decision process itself. A simple call is used to ask the decision-making process to *think*. It outputs a report of all the *Desires* it has elected to be acted upon.

Each *Goal* is an action producer and represents an independent module responsible for a reflex, a need or a goal. It produces *Desires* as possible intentions of action. Each *Desire* includes a target, an action, and the different parameters used to produce a complex action. It has to compute its interest with respect to the pursued goal and the current context. The interest can be seen as a fuzzy value, and the *Goal* as an extended 0+ fuzzy rule [Magnus2005]. For each sensed object, the *Goal* computes the interest and the parameters for each action it produces.

SpirOps AI includes a complex action selection mechanism. The basics are simple. *Goals* are sorted according to their interests. The mutual exclusivity between *Goals* is automatically managed. The most interesting combination of non-exclusive *Goals* is then elected as the corresponding action to perform. More advanced interferences between *Goals* can be designed, such as inhibition, secondary decisions and parametric interferences.

4.1.4. Example : the iCub control

The decision making process is illustrated through the case of the iCub high-level control. The outline proposed in this section is used in the next experiments carried out in Section 4.2. Figs. 4.2 & 4.3 show an overview of the implementation interface of these fuzzy rules in Spirops. They give synthetic representations of the relations between goals and actions in the decision making process shown in Fig. 4.1. We will describe these representation as following. First, we enumerate the perceptual data (the evaluation of robot and items states) which gives means to drive iCub toward its objectives. Then, we look at the available actions given to the robot, and finally we focus on the goals, *i.e.* the fuzzy rules, which serve as basis to the decision making process.

4.1.4.1. Perceptual information

Perceptual information can be treated as raw values, as those described in Table (4.1), or they can be transformed as fuzzy values to ease implementation and to get more symbolic data, as in Table (4.2). Notation $\bar{\bullet}$ represents the complement value : $\overline{v} = \mathbf{NOT}(v) = 1 - v$. Note that positions are in meters (m), angles are in radians (rad), and that boolean values, as well as the `stepCompletion` variable, can be directly transformed in fuzzy variables.

4.1.4.2. Actions

The actions are the possible maneuvers the robot can handle and represent the intermediate level as presented in Section 4.1.2. Each action can have some parameters, for instance hands trajectories, desired CoM position or iCub destination. The available actions are listed in Table (4.3). Every action is monitored to evaluate its completion, and when problems or errors occur, for instance in the `Grab` action, a boolean value `ProblemGrab` raises and some corrective actions are provided in the decision making engine to increase the robot capabilities.

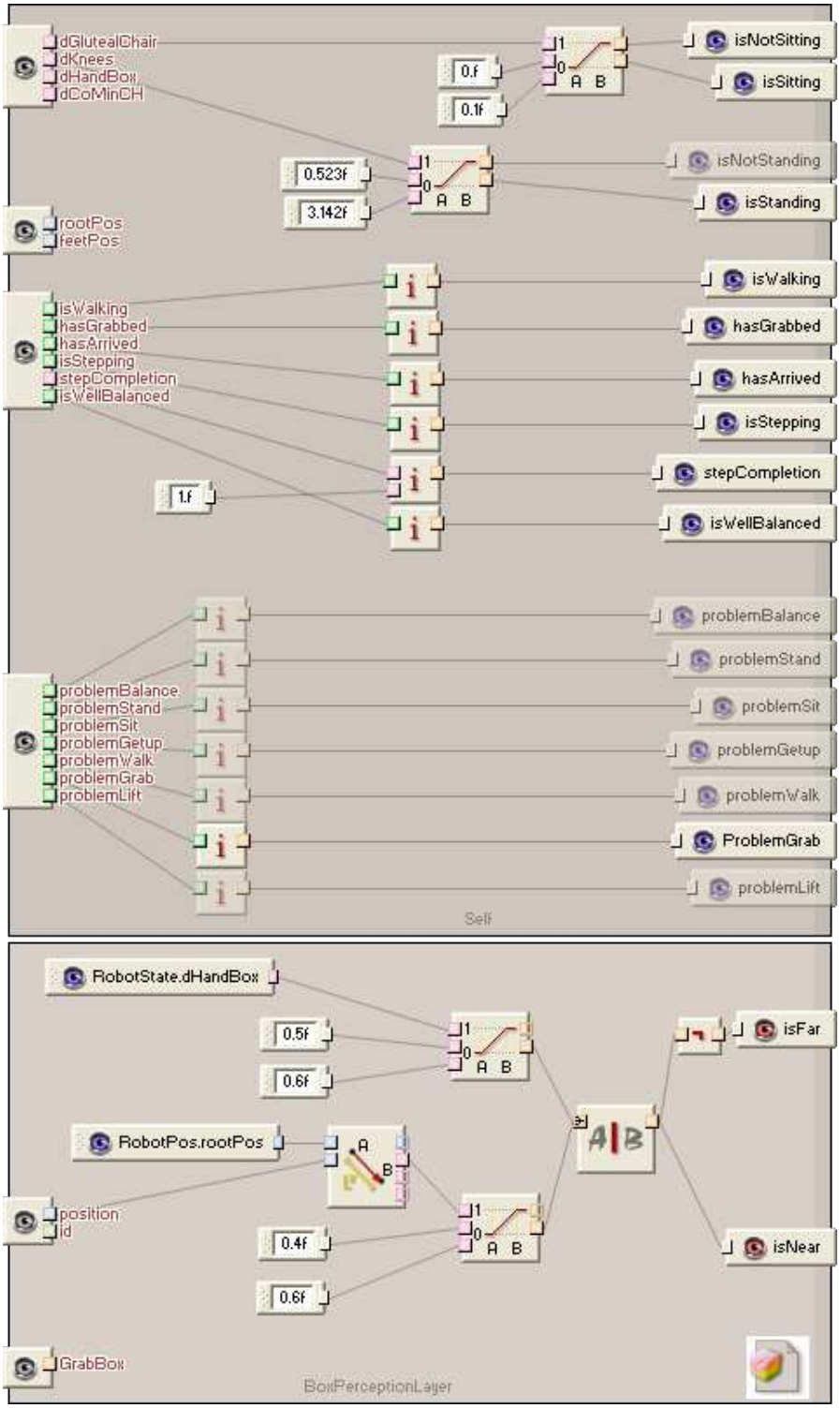


FIGURE 4.2.: Perceptual information in Spirops. The upper figure represents the proprioceptive sensations, that is the perceptions of its "Self". The lower figure is dedicated to the box perception located in the scene, so this module is named "BoxPerceptionLayer".

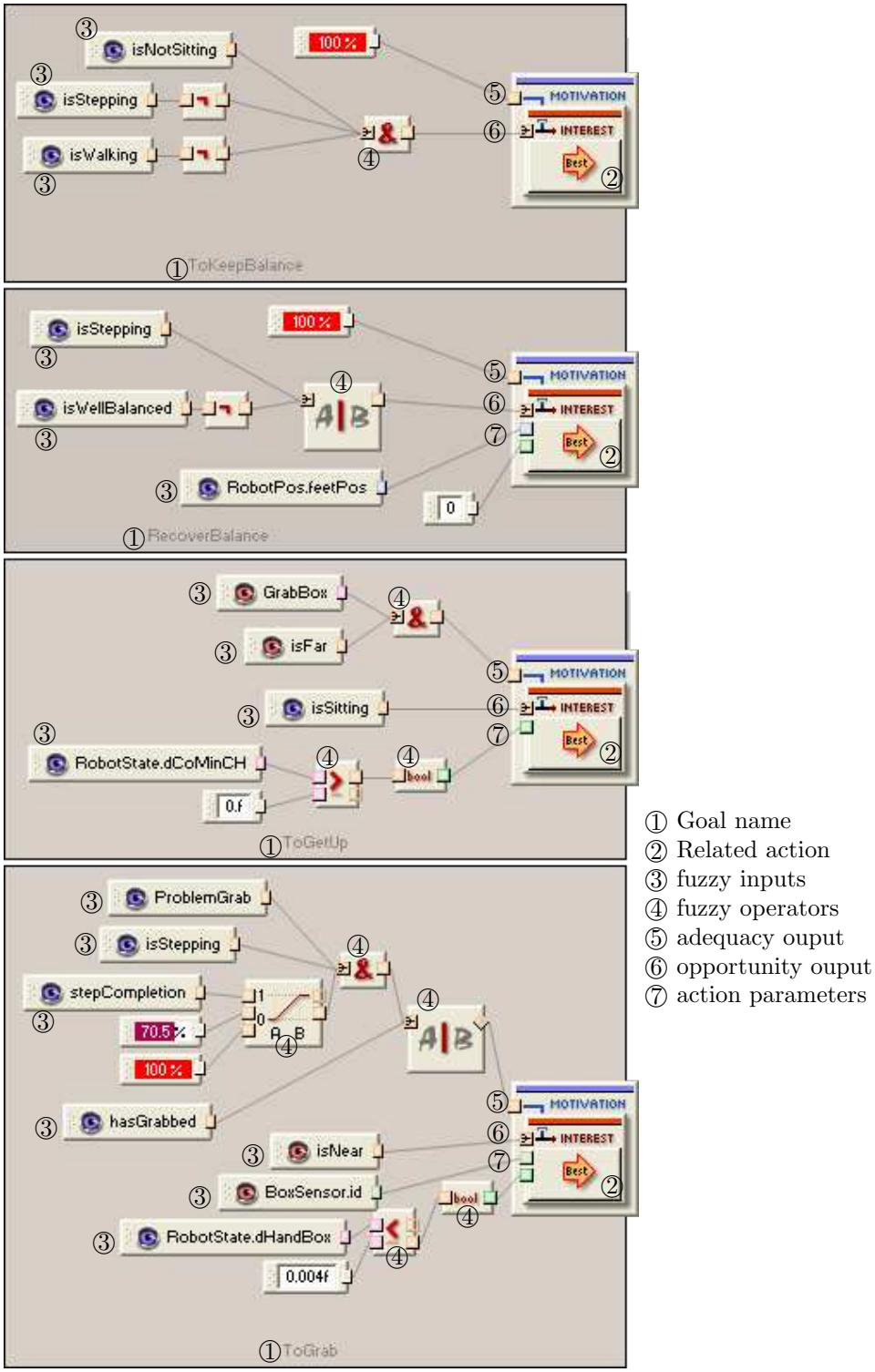


FIGURE 4.3.: Some goals or fuzzy rules in Spirops. The legend on the right explains what components take parts in these modules. ① are the fuzzy inputs shown in Fig. 4.2. ⑤ and ⑥ are respectively the adequacy and opportunity outputs needed for the decision making process described in Section 4.1.2.

name	type	description
dGlutealChair	float	distance between the robot bottom and the chair
dKnees	float	summation of the knees angular positions
dHandBox	float	minimal distance between the hands and the closest box
dCoMinCH	float	minimal distance between the CoM and the base of support borders, to have information about balance
rootPos	2D vector	root position (the pelvis) projected onto horizontal plane
feetPos	2D vector	feet position. We take the central point of the base of support
boxPos	2D vector	box positions. There are as many variables as boxes
boxId	integer	box number. There are as many variables as boxes
isWalking	boolean	true when robot is walking
isStepping	boolean	true when robot is stepping (when it makes one step for correcting action)
isWellBalanced	boolean	true as long as CoM is not to close to the base of support borders
hasGrabbed	boolean	true when iCub has caught the desired item
hasArrived	boolean	true when iCub has arrived to the desired location
stepCompletion	$\in [0, 1]$	percentage that indicates the step completion, while performing corrective action

TABLE 4.1.: Perceptual information in the iCub decision making process. This table enumerates available raw variables.

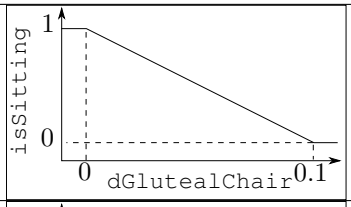
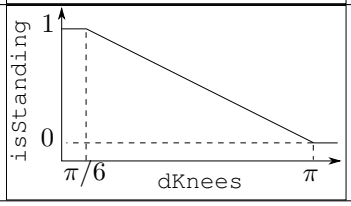
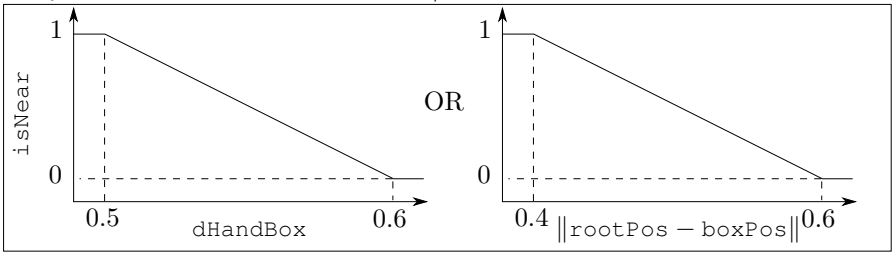
name	description
isSitting	fuzzy definition of the sitting position : 
isStanding	fuzzy definition of the standing position : 
isNear $= \overline{\texttt{isFar}}$	fuzzy definition of the distance robot/box : 

TABLE 4.2.: Perceptual information in the iCub decision making process. This table enumerates available fuzzy variables.

name	parameters	tasks
`Balance`	None	control ZMP to be at the center of the base of support
`Stand`	None	control the robot standing posture
`Sit`	None	control the robot sitting posture
`Walk`	goal (2D vector)	control the walking task to go to a desired location
`GetUp`	CanBreakUp (boolean)	control CoM and bottom contacts to get up
`Grab`	CanGrab (boolean) BoxId (integer)	control arms and hands contacts to grab the desired box
`Lift`	None	control arms to lift the box
`MakeStep`	goal (2D vector)	walk (make a single step) to correct a position

TABLE 4.3.: Available Actions in the iCub decision making process.

4.1.4.3. Goals : the fuzzy rules

The goals represent the needs that will drive the robot toward the achievement of its main objective. They are based on fuzzy rules which combine **adequacies** and the **opportunities** (cf. Section 4.1.2). We present in Table (4.4) all the rules used in the experiments. In this table, the function $\mathbf{LIN}(v_0, var, v_1)$ allows to write in a condensed manner a fuzzy output such as :

$$\mathbf{LIN}(v_0, var, v_1) = \begin{cases} 0 & var \leq v_0 \\ (var - v_0)/(v_1 - v_0) & v_0 \leq var \leq v_1 \\ 1 & v_1 \leq var \end{cases}$$

name	action	Desire = adequacy AND opportunity
`To Keep Balance`	`Balance`	adequacy : 1 opportunity : **NOT**(`isSitting` **OR** `isStepping` **OR** `isWalking`)
`To Stand`	`Stand`	adequacy : 1 opportunity : `isStanding`
`To Sit`	`Sit`	adequacy : 1 opportunity : `isSitting`
`Recover Balance`	`MakeStep`	adequacy : 1 opportunity : `isStepping` **OR** $\overline{\texttt{isWellBalanced}}$
`To Get Up`	`GetUp`	adequacy : `GetBox` **AND** `isFar` opportunity : `isSitting`
`To Walk`	`Walk`	adequacy : `GetBox` **AND** (`isFar` **OR** `isWalking` **OR** $\overline{\texttt{hasArrived}}$) opportunity : $\overline{\texttt{isSitting}}$
`To Grab`	`Grab`	adequacy : `GetBox` **AND NOT**(`hasGrabbed` **OR** (`ProblemGrab` **AND** `isStepping` **AND LIN**(0.75, `StepCompletion`, 1))) opportunity : `isNear`
`To Lift`	`Lift`	adequacy : `GetBox` opportunity : `hasGrabbed`
`Correct Position`	`MakeStep`	adequacy : `ProblemGrab` opportunity : $\overline{\texttt{isWalking}}$

TABLE 4.4.: Fuzzy rules defined in the iCub decision making process.

4.2. Experiments

The efficiency of the connection between the SpirOps AI software and the LQP-based controller is demonstrated through some experiments where iCub has to adapt its behavior according to its perception of the surrounding world. In the first case, the robot performs some missions with different geometric conditions, then it deals with some uncertainties in its perceptions, and finally, it has to realize some missions while being subject to some dynamic disturbances.

4.2.1. Geometric conditions

First, the robot must be able to achieve and repeat any mission, even when geometric conditions are different. This can be for instance the initial conditions (robot posture, box position, box height) or the nature of the objects (box dimension). This issue is illustrated through two examples where the robot has to grab a box on one hand, and it has to open a door in the other hand.

4.2.1.1. Grabbing a box

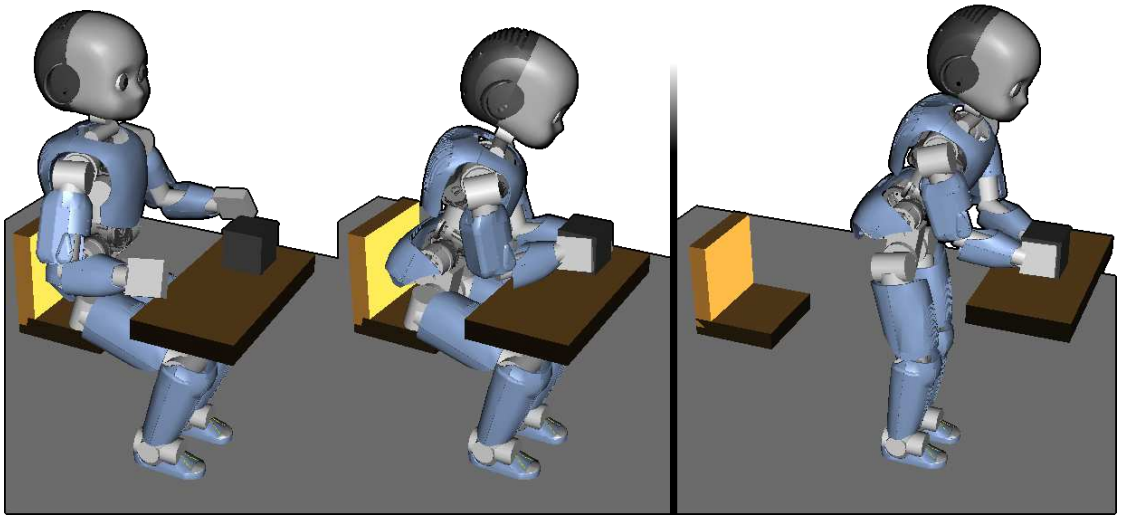


FIGURE 4.4.: Experiment 1 – Left-hand side : The box can be grabbed without walking – Right-hand side : A walking phase is necessary to grab the box.

In this experiment, the mission is to grab a box. This mission is tested in two cases where initial conditions are different. In both cases, the robot is initially sitting but the box is much further from the chair in the second case than in the first one. In the first case [1], the box is grabbed without a walk phase (see Fig. 4.4, left). In the second case [2], the robot has to walk (see Fig. 4.4, right) and Fig. 4.5 illustrates two points :

– actions are activated in a smooth manner without specifying any explicit transition task in contrast to what is proposed in Section 3,

1. Corresponds to Attachment `07_01_grabbing_box_near.avi`.
2. Corresponds to Attachment `07_02_grabbing_box_far.avi`.

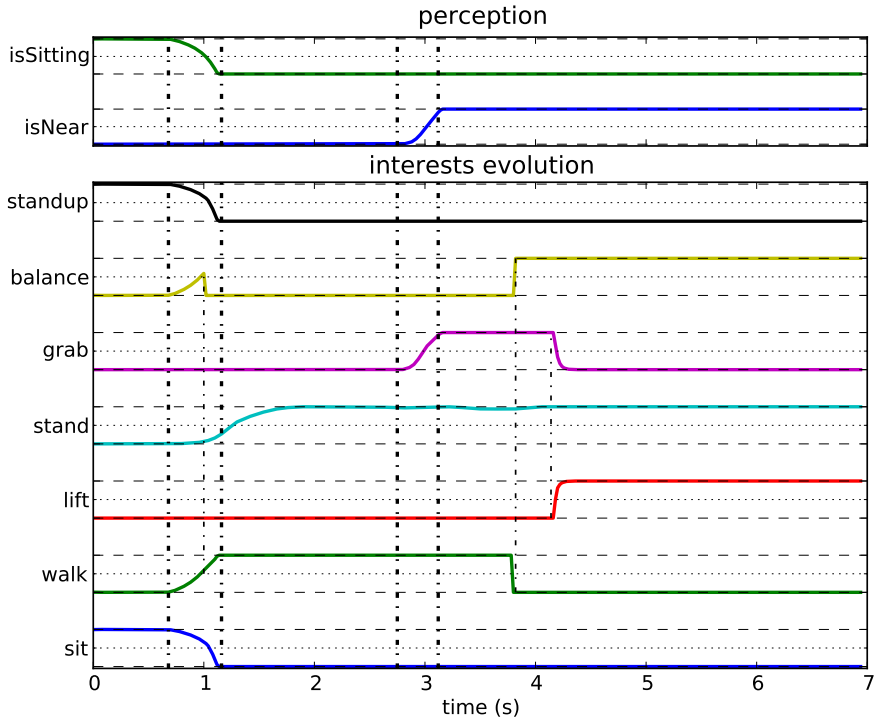


FIGURE 4.5.: Experiment 1 – Interests for actions and normalized distance to the box as a function of time. Bold dashed red lines highlight the correlations between percepts and action triggering. Thin dashed red lines highlight the correlations between actions.

– the obtained sequence of actions was not manually specified : it is automatically and reactively generated by SpirOps based on rules defined off-line, percepts such that the "isSitting" or "isNear" information, and the pursued goal.

4.2.1.2. Opening a door

In this second experiment, the robot opens a door many times in the same sequence. It pushes the door at the maximum, and when the task cannot be performed correctly, the robot adapts its position and posture, uses the closest hand to grab the door knob and pushes the door again. Figure 4.6 illustrates this experiment [3], and the perceptions of the robot are described in Fig. 4.7.

4.2.2. Dealing with uncertainties

In the previous experiments, the minimum grabbing distance between the robot and an object is well defined by the programmer. In some cases, it is not easy to define such values beforehand and this experiment shows how the high-level controller is able to call upon a corrective action in order to deal with an erroneous value. To illustrate this possibility, the robot is asked to grab a box but the minimum grabbing distance is not accurate and it is actually not small enough to grab the box (see Fig. 4.8) [4]. To

3. Corresponds to Attachment `08_opening_a_door.avi`.
4. Corresponds to Attachment `09_dealing_with_uncertainties.avi`.

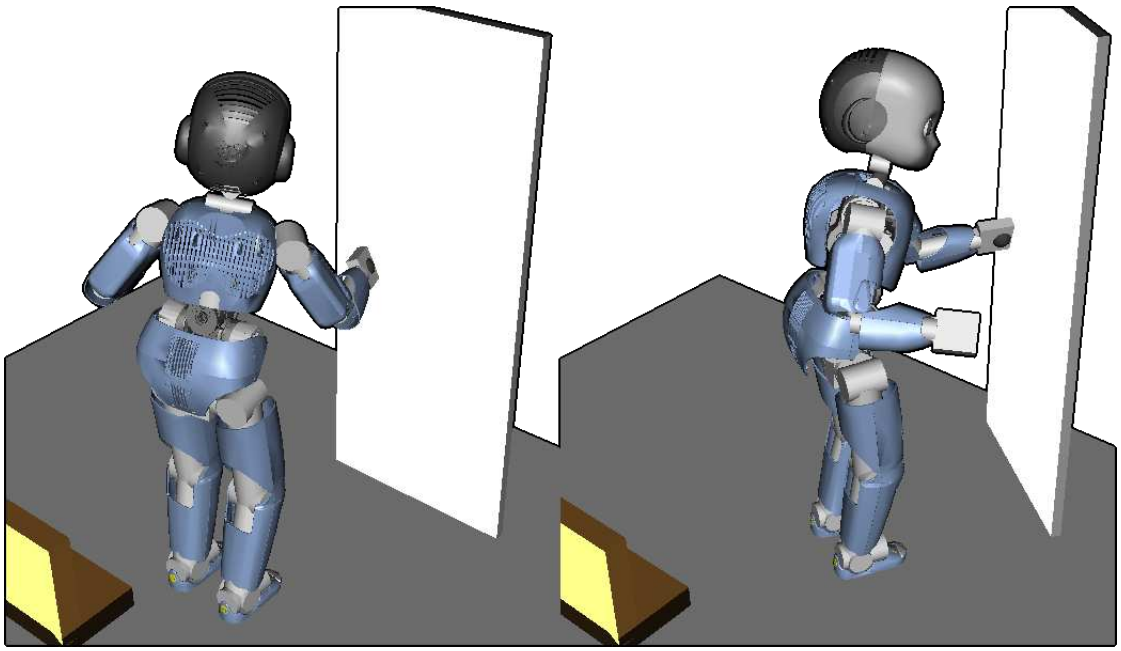


FIGURE 4.6.: Experiment 2 – The robot opens a door many times. It uses its left or right hand, depending on the distance between the hands and the knob.

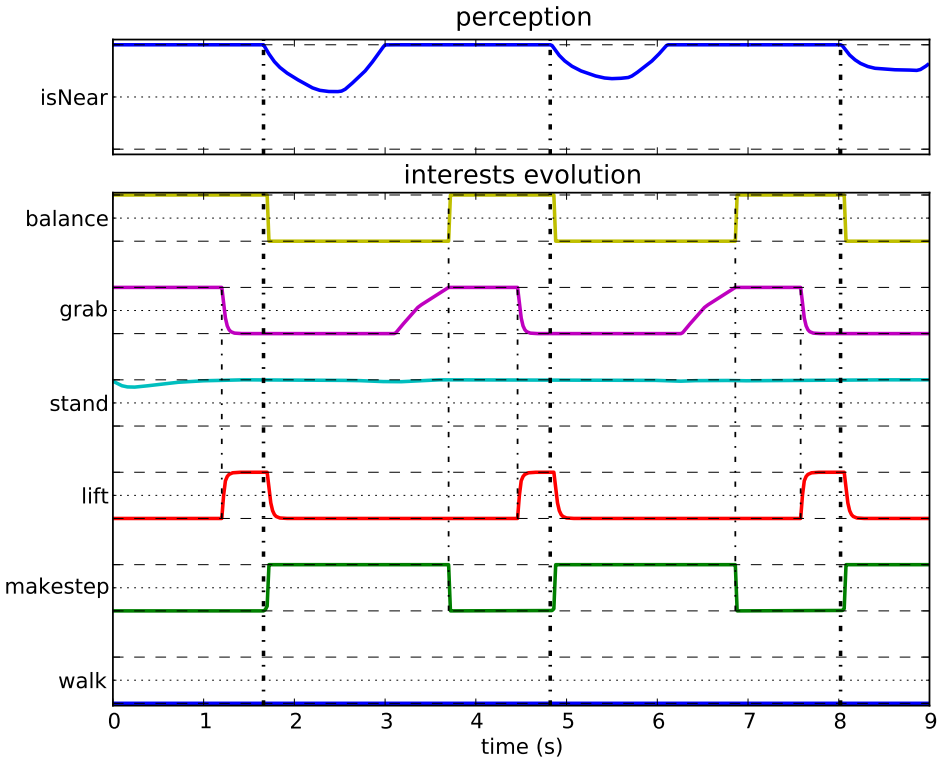


FIGURE 4.7.: Experiment 2 – Evolution of the actions interests according to perceptions. Bold dashed red lines show the correlation with perception, the thin ones highlight the relations between actions.

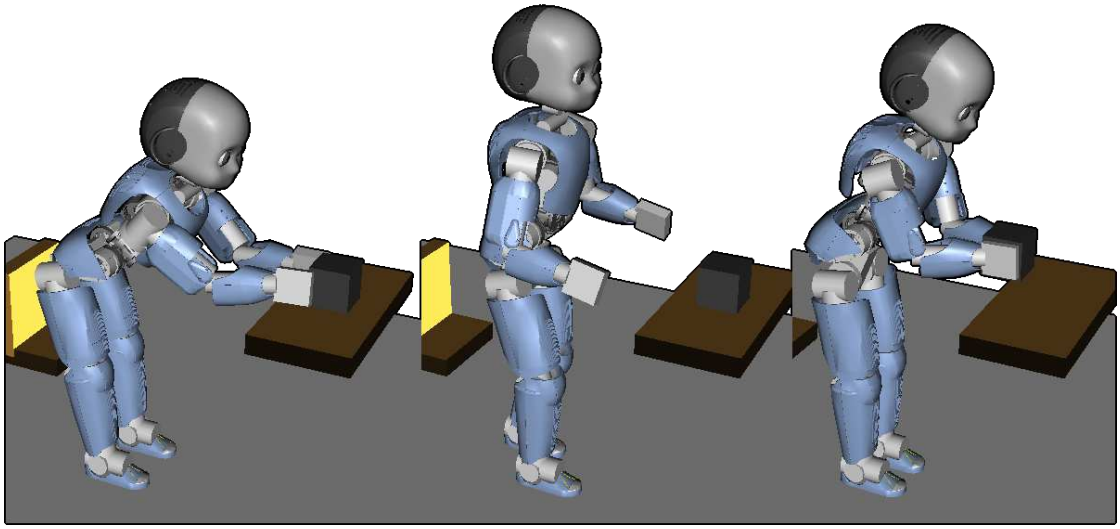


FIGURE 4.8.: Experiment 3 – Left-hand side : The robot fails to grab because of an erroneous choice of the "reachable" distances – Right-hand side : a corrective step allows it to manage this situation.

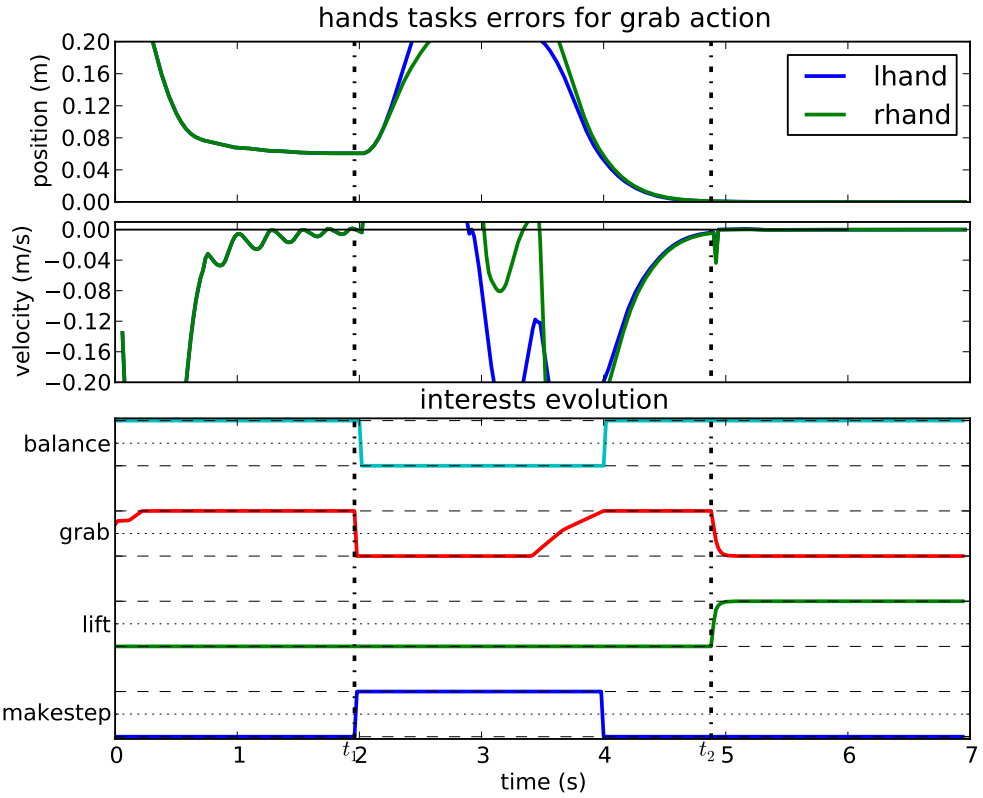


FIGURE 4.9.: Experiment 3 – Interests for actions, distance and velocity of hands toward the box as a function of time.

circumvent this kind of problem, a rule is added to trigger a stepping action whenever the distance to the box cannot be reduced by just standing. This is illustrated by Fig. 4.9 : at time t_1 the box is too far and its distance with respect to the hands is not evolving (velocity is $\approx 0.0m/s$). The stepping action is activated in order to manage this situation. At time t_2 the box is grabbed and can be lifted. This rule can be formalized as
CONDITION : IF $d(\text{Box}) > d_{min}$ AND $\dot{d}(\text{Box}) > \dot{d}_{min}$
ACTION : WALK ONE STEP TOWARD Box

Even though this additional rule is added by the programmer, its nature is rather general and a limited number of such corrective rules may suffice for the system to cope with similar situations which are characterized by the non-convergence of some error signal.

4.2.3. Dynamic adaptation

The previous experiment illustrates the need for corrective rules when dealing with purely geometric information (the distance to an object), and the triggering of the stepping action could have been decided *a priori*. However when dealing with the balance of a humanoid, corrective rules are based on the state of the system. This state cannot be predicted beforehand since it is the result of the evolution of the system, subjected to unknown disturbances, over time. As a matter of fact, one needs to define corrective rules/actions based on information related to the dynamics of the system. The proposed example illustrates this concept in the case of a manipulation task. The robot is supposed to grab a box and to drop it on the other side of the table. However, the final position of the box and its weight may be such that achieving the task can potentially be incompatible with equilibrium conditions and thus may call upon corrective actions. The rule triggering this action is based on the following equilibrium test : given the current state (position and velocity) of the CoM and the closest critical location of the ZMP (on the convex hull), is it possible to decelerate the CoM so that it actually stops before reaching this critical ZMP location ? This test is performed using a prediction of

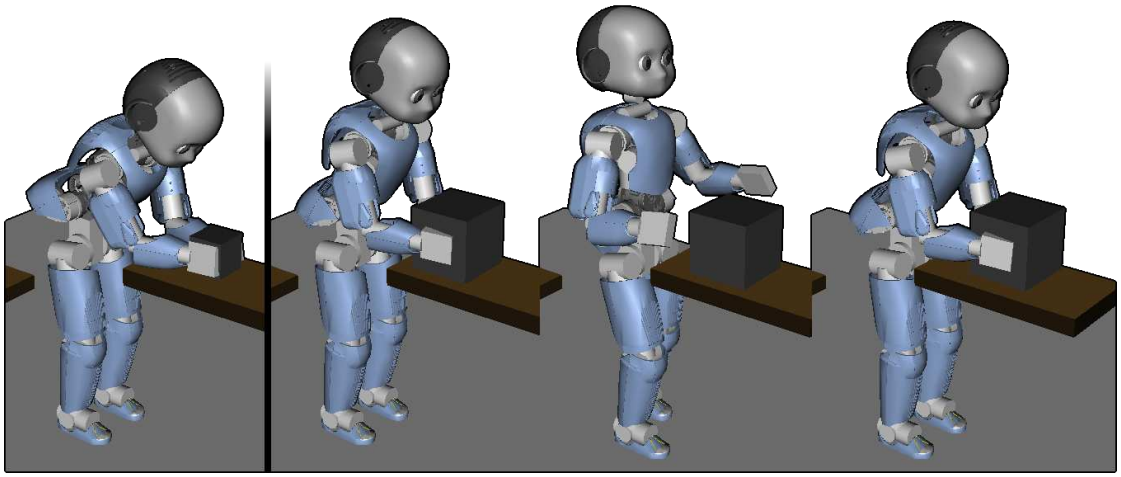


FIGURE 4.10.: Experiment 4 – Left-hand side : the robot has to move a $0.10Kg$ box and does it without corrective actions – Right-hand side : the robot has to move a $2.50Kg$ box and cannot do it without corrective actions.

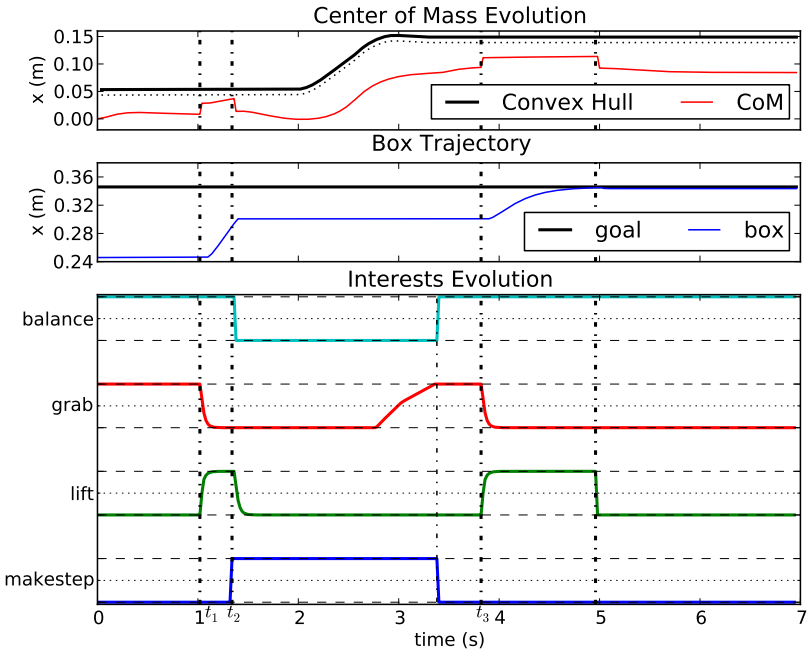


FIGURE 4.11.: Experiment 4, second case – Interests for actions, distance between the box and its final location, projection of the CoM position on the ground in the forward direction (red) and convex hull critical limit (real (plain black) and with safety margin (dashed black)) as a function of time.

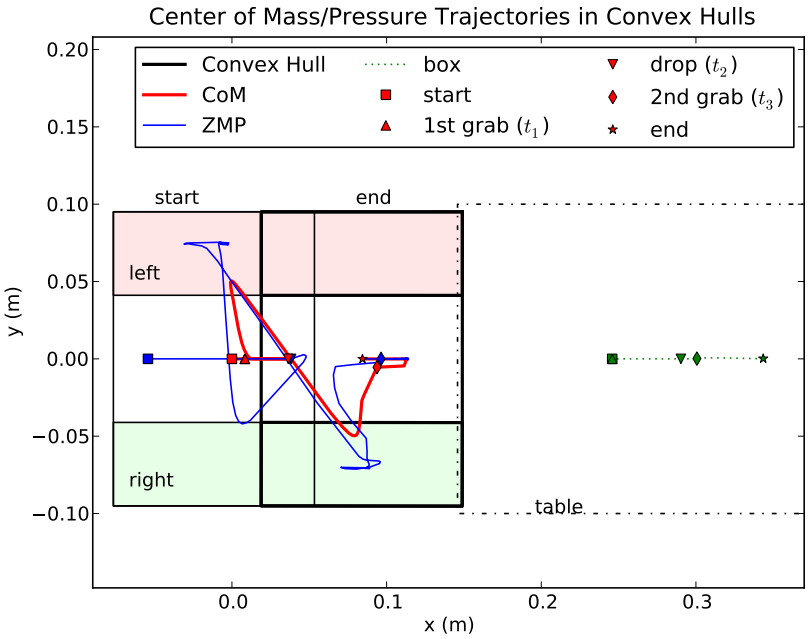


FIGURE 4.12.: Experiment 4, second case – Trajectories of the ground projections of the CoM position (plain bold red), ZMP position (plain blue) / Box positions (dashed green) / Foot placement (colored zones) and convex hulls (plain black) before and after corrective actions.

the future states of the CoM (based on [Kajita2003]) and a safety margin is taken with respect to the actual convex hull. The associated corrective action is to drop the load (if present) and to step toward the critical direction.

Two cases are tested [5]. In the first case (see Fig. 4.10, left), the box weighs $0.10Kg$. The box lifting action does not induce a large modification of the CoM location and as matter of fact the task is achieved without having to resort to any corrective action. In the second case (see Fig. 4.10, right), the box weighs $2.50Kg$ which induces a shift of the CoM projection at time t_1 (as shown in Figs. 4.11 & 4.12). So given the desired box motion, this leads to a situation where equilibrium may not be ensured any longer. This situation is detected using the described equilibrium test and leads to the dropping of the box at time t_2 thus inducing an improvement of the stability margin. The second corrective action is a step in the critical direction which brings the robot to a location where the box can finally be lifted and left at the correct location at time t_3 without risks in terms of equilibrium.

4.3. Conclusion

This chapter presents high-level control whose purpose is to adjust the parameters of this low-level control based on the perceptual information of iCub. This controller uses fuzzy logic to elect the best couples "Goal/Action" according to some simple rules, and their "Interests" (fuzzy value between 0 and 1) are particularly suitable to set tasks weights of the low-level when using a weighting strategy. This association is illustrated through different experiments where iCub adapts its actions according to geometric conditions, uncertainties and dynamic effects. The purpose of this second part is to demonstrate that the connection between the two levels of control is possible and intuitive when using a weighting strategy.

5. Corresponds to Attachments `10_01_dyn_light.avi` & `10_02_dyn_heavy.avi`.

Conclusion & perspectives

5.1. Conclusion

Humanoid robots are very sophisticated mechanical systems which are intended in the future to work in human environment. Although there are many advances in humanoid robotics, controlling such mechanisms is still very challenging, for it requires effective algorithms to achieve missions subject to many constraints. For instance, multitasking and task sequencing are important issues when one wants to generate complex behaviors like humans. Analytical methods allow to perform many tasks simultaneously, but they are not suitable when inequality constraints should be taken into account. Thus, this thesis focuses on the use of constrained optimization tools to control humanoids, as well as the synthesis of complex behaviors with high level controllers, while ensuring the control vector continuity.

To do so, many tools and methods have been developed. First, a new dynamic simulator, Arboris-python, is presented. Based on the Matlab version developed by Alain Micaelli, later joined by Sébastien Barthélemy and the author, this simulator written in the python language is very suitable for designing and prototyping robots and controllers thanks to a weakly typed language and no compilation. The equations of motion of mechanical systems composed of rigid bodies interconnected with joints are recalled in matrix forms, and they are computed from the Boltzmann-Hamel equations which allow to take into account more complex joints, as the ball and socket joint. The integration is done using time-stepping and a semi-implicit Euler integration scheme to have a good stability, and constraints are then solved with a Gauss-Seidel algorithm. The main advantage of Arboris-python is the use of generalized coordinates to compute the equations of motion by integrating joints configuration in the model. It reduces matrices dimension, speeds up the calculation and prevents numerical errors. A quick example of the simulation process is developed at the end of the first chapter.

Then, the control of mechanical systems with optimization tools is developed. There are many constraints in robotics which have to be integrated to generate efficient control, both internal like equations of motion, and external like interactions. These constraints, especially inequalities, are difficult to handle with analytical methods, but can be treated with optimization-based controllers. So, methods to solve convex problems subject to equality and inequality constraints are recalled. Constraints type, as well as dynamic control requirements, lead to the development of a LQP-based controller which gives a way to perform dynamic control of humanoid robots when they evolve in constrained environments.

However, it remains to deal with the task definition, the achievement of several tasks simultaneously and their sequencing, in order to synthesize complex behaviors. Inspired from the task function approach [Espiau1990], the tasks defined in this thesis can be

modified to get a physically consistent cost function, and their achievement can follow weighting or hierarchical strategies. In the literature, some authors have proposed hierarchical methods to perform multiple tasks, for instance Kanoun which controls a humanoid robot in kinematics [Kanoun2009b]. We propose in this thesis to use a weighting strategy while performing multitasking in dynamics, whose advantages are the following :

- it speeds up computation,
- it allows monitoring of input torque vector continuity when changes happen in the set of tasks and constraints,
- it gives some means to set a simple connection between low and high-level controllers.

Indeed, a hierarchical strategy requires to solve as many optimization programs as tasks, which can be restrictive for large tasks set and may make real-time applications unfeasible, whereas weighting strategy computes a single optimization program per call.

Besides, the whole-body control must be feasible for all sequences composed of several tasks, and one has to focus on transition to get continuity of the input torque vector evolution. This can happen when changes occur in the set of tasks and constraints. The proposed solution is to use a weighting strategy and to change the tasks weights in a continuous manner during a short period of time. This is also a way to manage transitions during contact breaking, and more generally when the kinematic chain structure is modified.

This LQP-based low-level controller allows to perform many sequences of tasks, but more complex ones may need high-level control. In this thesis, this connection is done thanks to fuzzy logic based rules and the decision making engine SpirOps, the tasks weights becoming some fuzzy variables quite easy to handle.

Validation of the LQP-based controller is done through many experiments carried out with the simulator Arboris-python. The virtual model is based on the iCub robot. It performs multitasking with the different task controllers, for instance to control impedance while interacting or to avoid some obstacles. Then, transitions in the set of tasks and constraints are studied by comparing weighting and hierarchical strategies on one hand, and by varying the weights continuously during the constraints transition period on the other hand. It shows that sharp evolutions in the input torque vector are avoided and that the control of transitions is easily performed. The last experiments compare the different task formalisms through three different cases.

Finally, the LQP-based low-level controller and a high-level controller are connected to perform more complex behaviors. This thesis does not intend to solve artificial intelligence problems but to highlight the link between these two controllers in an easy and efficient way. To do so, the decision process is performed with fuzzy logic based rules by taking into account the robot goals, the perceptual information, and the tasks weights as fuzzy variables. The core of the decision making engine relies on the available actions which select the goals to achieve, and adjust the tasks accordingly. This decision process is done with SpirOps.

To illustrate this connection, some experiments are carried out with the virtual iCub robot where the tasks sequencing is not known in advance, so the robot has to adapt. The first experiments deal with geometric considerations, for instance if initial conditions are not the same, and they show the fuzzy variables evolution function to perceptions. Then

the decision process takes into account some uncertainties about the rules to properly achieve robot mission. Finally, the robot achieves some missions with different dynamic conditions, with boxes of different weights.

5.2. Perspectives

In the future, some interesting points should be investigated to extend and improve the capabilities of optimization-based control.

5.2.1. Problem resolution and modeling

The first one is about LQP resolution. In this thesis, the solver CVXOPT uses interior-point methods that require some tolerance parameters to find the solution. This may lead to uncertainties and affect the results found in Section 3.5.1 with the two formalisms. One way would be to use active constraint-set methods, as presented in Section 2.2.3, because it transforms the original inequality problem into many equality problems which are solved analytically when they are quadratic. This implies that no more tolerance are necessary in the algorithm, and uncertainties are left to the numerical computation of the solution. Besides, the number of active inequality constraints during simulations is low in practice, which should lead to few iterations.

If the emphasis is set on the problem modeling, one can look for SDP or LCP. These methods may be more time consuming but they integrate more precise model. For instance, SDP can be used for integrating LMIs, which give more accurate representation of the Coulomb cone of friction (see Eq.(2.9)), and LCP can manage unilateral contact more efficiently, without any assumption about contacts state.

5.2.2. Tasks and strategy

One can wonder about the influence of the different tasks types used in control. The experiments carried out in this thesis do not compared the effects of matrices K_H and K_N because it is difficult to extract significant differences during simulations. Some complementary studies are necessary, wishing that using such transformations give other interesting benefits on problem resolution (better condition number, sparsity) or robotic control (singularity management, better torque distribution).

Another interesting point can be addressed to LQP building. Indeed, iCub interacts with the world, but the contact points and kinematic loops occurring during experiments are not always active, so they can be deleted from the problem. In the opposite, equality constraints are added to the problem when contacts appear, these equations (and more generally all equality constraints of the problem) can be integrated in the problem variable by substitution, reducing the problem dimension. Generally, optimization literature does not recommend this technique in order to keep the useful structure of large dimensional problems, but those proposed in this thesis are quite small, and variable substitution from one formalism to the other gives encouraging results in terms of computation time (see Section 3.5.2).

A more difficult point to handle is about the boundary definition between tasks and constraints. As presented in Section 2.3, a task can be expressed as a constraint, in the hierarchical strategy for instance, and a constraint can become a task, admitting some error. Some constraints seem immutable (*e.g.*the equations of motion), but others are disputable. For instance, when a robot carries a box with handles, interactions can be modeled as bilateral closure constraints, however it may be secondary to other tasks like "maintaining balance", and should be deleted when the error is too high. Hence, this issue should be investigated to better understand this boundary.

One last point is about the strategy used in this thesis. Experiments in Chapters 3–4 have shown interesting results with weighting strategy. However, a hybrid controller can be an alternative to take advantages of both hierarchy and trade-off, if tasks have constant relative importances (*i.e.* no shifting between them). For instance, a balancing task is generally more important than a grabbing task, which can be treated with a hierarchical strategy, whereas left and right arms tasks have generally the same importance, and thus can be treated with a weighting strategy. Besides, this technique would allow to integrate "inequality tasks", as presented in [Kanoun2009b].

5.2.3. Toward more complex behaviors

Robot behaviors can be improved by replacing or associating SpirOps with other high-level controllers. For instance, a planning module would give to the robot some means not to fall into "traps" that cannot be avoided reactively. Besides, the decision making engine can be completed with tools from the artificial intelligence, for instance dynamic programming to select better politics. The most interesting point would be to use reinforcement learning to improve the set of task weights, to obtain efficient values before and during transitions.

One can also expect to extend the LQP-based controller with more task controllers, as proposed in Section 2.3.4.3. The impedance controller described in this thesis is only used with upper limbs without walking. An idea would be to join impedance and walking to mix manipulation and balancing by taking into account disturbances, anticipated or not.

Finally, some works have already started about controllers based on Central Pattern Generator (CPG) [Hooper2001], to perform the walking task in a different manner. It generates some coupled oscillating trajectories tracked by the lower limbs joints, as shown in Fig. 5.1. This method is very interesting because it does not need to know in advance the feet positions and it requires few parameters to tune the robot gait.

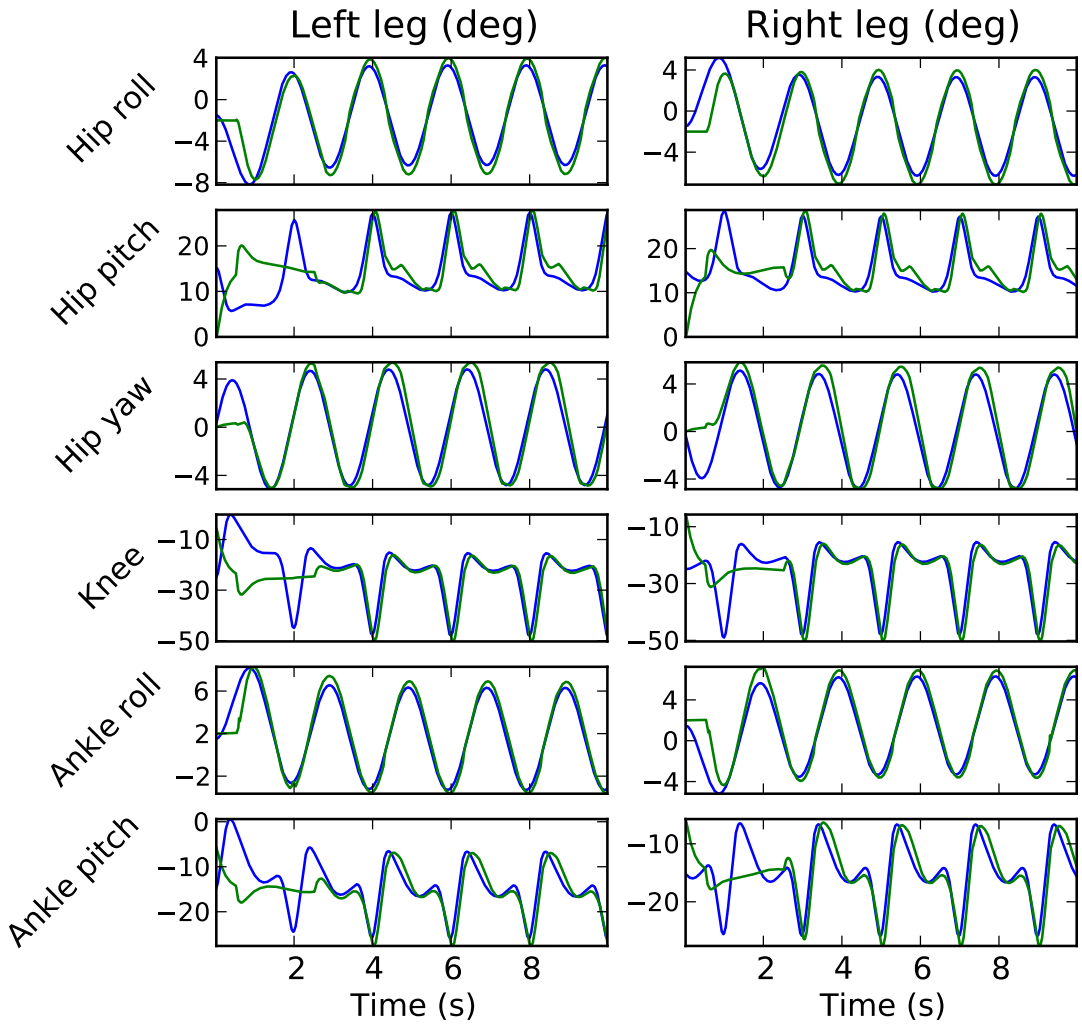


FIGURE 5.1.: Joints motions generated by the CPG controller. The blue and green lines are respectively the reference and effective trajectories.

Bibliographie

[Abe2007] Yeuhi Abe, M. da Silva, and Jovan Popović. Multiobjective control with frictional contacts. In *ACM SIGGRAPH/Eurographics symposium on Computer animation. Proceedings*, pages 249–258, 2007.

[Adams] Adams Multibody Dynamics Simulation. http://www.mscsoftware.com/Products/CAE-Tools/Adams.aspx.

[Albert1972] Arthur Albert. *Regression and the Moore-Penrose pseudoinverse*. Elsevier, 1972.

[Albu2002] F. Albu, J. Kadlec, N. Coleman, and A. Fagan. The gauss-seidel fast affine projection algorithm. In *IEEE Workshop on Signal Processing Systems*, pages 109–114, oct. 2002.

[Altmann1986] S.L. Altmann. *Rotations, quaternions, and double groups*. Clarendon Press Oxford, 1986.

[Anitescu1997] M. Anitescu and F.A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14 :231–247, 1997.

[Arboris] S. Barthélemy, J. Salini, and A. Micaelli. Arboris-python. https://github.com/salini/arboris-pyhton.

[Baraff1996] David Baraff. Linear-time dynamics using lagrange multipliers. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 137–146, 1996.

[Bartels1969] Richard H. Bartels and Gene H. Golub. The simplex method of linear programming using lu decomposition. *Communication of the ACM*, 12 :266–268, may 1969.

[Barthelemy2008] S. Barthélemy and P. Bidaud. Stability measure of postural dynamic equilibrium based on residual radius. *RoManSy'08 : 17th CISM-IFToMM Symposium on Robot Design, Dynamics and Control*, 2008.

[Bemporad2002] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1) :3–20, 2002.

[Bertsekas2003] D.P. Bertsekas, A. Nedić, and A.E. Ozdaglar. *Convex analysis and optimization*. Athena Scientific optimization and computation series. Athena Scientific, 2003.

[Boyd2004] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Bullet] Bullet Physics Library Game Physics Simulation. http://bulletphysics.org/wordpress/.

[Buss1997] M. Buss, L. Faybusovich, and J.B. Moore. Recursive algorithms for real-time grasping force optimization. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 1, pages 682–687, apr. 1997.

[Caccavale1995] F. Caccavale, P. Chiacchio, and S. Chiaverini. Kinematic control of a seven-joint manipulator with non-spherical wrist. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 50–55, oct. 1995.

[Cambon2009] S. Cambon, R Alami, and F Gravot. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, 28 :104–126, jan. 2009.

[Charbonnier1995] F. Charbonnier, H. Alla, and R. David. The supervised control of discrete event dynamic systems : a new approach. *34th IEEE Conference on Decision and Control. Proceedings*, 1 :913–920, dec. 1995.

[Chardonnet2009] Jean-Rémy Chardonnet. *Modèle Dynamique Temps-Réel pour l'Animation d'Objets Poly-Articulés dans les Environnements Contraints, Prise en Compte des Contacts Frottants et des Déformations Locales : Application en Robotique Humanoïde et aux Avatars Virtuels*. PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc, 2009.

[Chevallier2006] D. Chevallier. *Introduction à la théorie des groupes de Lie réels*. Mathématiques à l'université. Ellipses, 2006.

[Chiaverini1997] S. Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transactions on Robotics and Automation*, 13(3) :398–410, jun. 1997.

[Choi2000] Su Il Choi and Byung Kook Kim. Obstacle avoidance control for redundant manipulators using collidability measure. *Robotica*, 18 :143–151, mar. 2000.

[Collette2009] Cyrille Collette. *Commande dynamique d'humains virtuels : équilibre robuste et gestion des tâches*. PhD thesis, INRS - Institut National de Recherche et de Sécurité, 2009.

[Cvxopt] J. Dahl and L. Vandenberghe. cvxopt - python software for convex optimization. http://abel.ee.ucla.edu/cvxopt/.

[DaSilva2008] M. da Silva, Yeuhi Abe, and J. Popović. Simulation of human motion data using short-horizon model-predictive control. *Computer Graphics Forum*, 27(2) :371–380, apr. 2008.

[Dauer1991] J. P. Dauer. Optimization over the efficient set using an active constraint approach. *Mathematical Methods of Operations Research*, 35 :185–195, 1991.

[Dimitrov2009] D. Dimitrov, P.-B. Wieber, O. Stasse, H.J. Ferreau, and H. Diedam. An optimized linear model predictive control solver for online walking motion generation. In *IEEE International Conference on Robotics and Automation*, pages 1171 –1176, may 2009.

[Dorato1995] Peter Dorato, Chaouki T. Abdallah, and Vito Cerone. *Linear Quadratic Control : An Introduction*. Prentice-Hall, 1995.

[Doty1992] Keith L. Doty. An essay on the application of weighted generalized-inverses in robotics. In *5th Conference on Recent Advances in Robotics. Proceedings*, Florida Atlantic University, Boca Raton, 1992.

[Doty1993] Keith L. Doty, Claudio Melchiorri, and Claudio Bonivento. A theory of generalized inverses applied to robotics. *International Journal of Robotics Research*, 12 :1–19, feb. 1993.

[Duindam2006] Vincent Duindam. *Port-based modeling and control for efficient bipedal walking robots*. PhD thesis, University of Twente, 2006.

[Duindam2007] V. Duindam and S. Stramigioli. Lagrangian dynamics of open multibody systems with generalized holonomic and nonholonomic joints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3342 –3347, nov. 2007.

[Escande2010] A. Escande, N. Mansard, and P.-B. Wieber. Fast resolution of hierarchized inverse kinematics with inequality constraints. In *IEEE International Conference on Robotics and Automation*, pages 3733 –3738, may 2010.

[Espiau1990] Bernard Espiau, François Chaumette, and Patrick Rives. Une nouvelle approche de la relation vision-commande en robotique. Rapport de recherche RR-1172, INRIA, 1990.

[Faverjon1987] B. Faverjon and P. Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 4, pages 1152–1159, mar 1987.

[Featherstone1983] R. Featherstone. The calculation of robot dynamics using articulated-body inertias. *International Journal of Robotics Research*, 2 :13–30, May 1983.

[Flash1985] Tamar Flash and Neville Hogans. The coordination of arm movements : An experimentally confirmed mathematical model. *Journal of neuroscience*, 5 :1688–1703, 1985.

[Greenwood2003] D.T. Greenwood. *Advanced dynamics*. Cambridge University Press, 2003.

[Han2000] Li Han, J.C. Trinkle, and Z.X. Li. Grasp analysis as linear matrix inequality problems. *IEEE Transactions on Robotics and Automation*, 16(6) :663–674, dec. 2000.

[Havok] Havok physics. http://www.havok.com/products/physics.

[Hindi2006] H. Hindi. A tutorial on convex optimization ii : duality and interior point methods. In *American Control Conference*, jun. 2006.

[Hogan1985] Neville Hogan. Impedance control : An approach to manipulation. In *American Control Conference*, pages 304–313, 1985.

[Hogan1987] Neville Hogan. Stable execution of contact tasks using impedance control. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 4, pages 1047–1054, mar. 1987.

[Hooper2001] Scott L Hooper. *Central Pattern Generators*. John Wiley & Sons, Ltd, 2001.

[HuManS] Martine Eckert, Mitsuhiro Hayashibe, David Guiraud, Pierre-brice Wieber, and Philippe Fraisse. Simulating the human motion under functional electrical stimulation using the humans toolbox. In Nadia Magnenat-Thalmann, Jian J. Zhang, and David D. Feng, editors, *Recent Advances in the 3D Physiological Human*, pages 121–131. Springer London, 2009.

[Ibanez2011] Aurélien Ibanez. Planification dynamique de tâches complexes pour mannequin virtuel. Master's thesis, Université Pierre et Marie Curie, Paris 6, 2011.

[Jean1999] M. Jean. The non-smooth contact dynamics method. *Computer Methods in Applied Mechanics and Engineering*, 177(3-4) :235–257, 1999.

[JonghoonPark2005] Jonghoon Park and Wan-Kyun Chung. Geometric integration on euclidean group with application to articulated multibody systems. *IEEE Transactions on Robotics*, 21(5) :850–863, oct. 2005.

[Kajita2003] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 2, pages 1620–1626, 2003.

[Kajita2006] Shuuji Kajita, Mitsuharu Morisawa, Kensuke Harada, Kenji Kaneko, Fumio Kanehiro, Kiyoshi Fujiwara, and Hirohisa Hirukawa. Biped walking pattern generator allowing auxiliary zmp control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 2993–2999, oct. 2006.

[Kanehiro2008] Fumio Kanehiro, Florent Lamiraux, Oussama Kanoun, Eiichi Yoshida, and Jean paul Laumond. A local collision avoidance method for non-strictly convex polyhedra. In *Proceedings of Robotics : Science and Systems IV*, 2008.

[Kanoun2009a] Oussama Kanoun, Florent Lamiraux, Pierre-Brice Wieber, Fumio Kanehiro, Eiichi Yoshida, and Jean-Paul Laumond. Prioritizing linear equality and inequality systems : Application to local motion planning for redundant robots. In *IEEE International Conference on Robotics and Automation*, pages 2939–2944, may 2009.

[Kanoun2009b] Oussama Kanoun. *Contribution à la planification de mouvement pour robots humanoïdes*. PhD thesis, Université de Toulouse III - Paul Sabatier, 2009.

[Karmarkar1984] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[KawadaHRP2] AIST. Humanoid robotics project, with kawada industries, inc. www.is.aist.go.jp/humanoid or http://global.kawada.jp/mechatronics/.

[Keith2009] François Keith, Nicolas Mansard, Sylvain Miossec, and Abderrahmane Kheddar. Optimization of tasks warping and scheduling for smooth sequencing of robotic actions. In *IEEE/RSJ international conference on Intelligent robots and systems. Proceedings*, pages 1609–1614, 2009.

[Khatib1985] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 2, pages 500–505, mar. 1985.

[Khatib1987] Oussama Khatib. A unified approach for motion and force control of robot manipulators : The operational space formulation. *IEEE Journal of Robotics and Automation*, 3(1) :43–53, feb. 1987.

[Khatib2002] Oussama Khatib, Oliver Brock, Kyong-Sok Chang, Francois Conti, Diego Ruspini, and Luis Sentis. Robotics and interactive simulation. *Commun. ACM*, 45 :46–51, mar. 2002.

[Khatib2008] Oussama Khatib and Luis Sentis. A unified framework for whole-body humanoid robot control with multiple constraints and contacts. *European Robotics Symposium*, pages 303–312, 2008.

[Kolen2001] J. Kolen and S. Kremer. *Representation of Discrete States*. Wiley-IEEE Press, 2001.

[Kreutz1989] K. Kreutz. On manipulator control by exact linearization. *IEEE Transactions on Automatic Control*, 34(7) :763–767, jul. 1989.

[LQPctrl] Joseph Salini. LQP-based controller. https://github.com/salini/LQPctrl.

[LaValle2006] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[Li2006] W. Li. *Optimal Control for Biological Movement Systems*. PhD thesis, University of California, San Diego, USA, 2006.

[Liegeois1977] Alain Liégeois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Transactions on Systems, Man and Cybernetics*, 7(12) :868–871, dec. 1977.

[Liu1999] Yun-Hui Liu. Qualitative test and force optimization of 3-d frictional form-closure grasps using linear programming. *IEEE Transactions on Robotics and Automation*, 15(1) :163–173, feb. 1999.

[Liu2004] Guanfeng Liu and Zexiang Li. Real-time grasping-force optimization for multifingered manipulation : theory and experiments. *IEEE/ASME Transactions on Mechatronics*, 9(1) :65–77, mar. 2004.

[Liu2005] T. Liu and M.Y. Wang. Computation of three dimensional rigid body dynamics of multiple contacts using time-stepping and gauss-seidel method. *IEEE Transactions on Automation Science and Engineering*, nov. 2005.

[Luthi1985] Hans-Jakob Lüthi. On the solution of variational inequalities by the ellipsoid method. *Mathematics of Operations Research*, 10(3) :pp. 515–522, 1985.

[MRS] Microsft® Robotics Developer Studio. http://www.microsoft.com/robotics/.

[Maciejewski1985] A.A. Maciejewski and C.A. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *International Journal of Robotics Research*, 4(3) :109, 1985.

[Magnus2005] P. D. Magnus. *forallx : An Introduction to Formal Logic*. University at Albany, State University of New York, 2005.

[Mahony2009] R. Mahony, T. Hamel, J. Trumpf, and C. Lageman. Nonlinear attitude observers on so(3) for complementary and compatible measurements : A theoretical study. In *48th IEEE Conference on Decision and Control, held jointly with the 2009 28th Chinese Control Conference. Proceedings*, pages 6407–6412, dec. 2009.

[Mansard2007] Nicolas Mansard and Francois Chaumette. Task sequencing for high-level sensor-based control. *IEEE Transactions on Robotics*, 23(1) :60–72, 2007.

[Mansard2009] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robots : The stack of tasks. In *International Conference on Advanced Robotics, ICAR 2009*, pages 1 –6, june 2009.

[Merlhiot2009] Xavier Merlhiot. *Une contribution algorithmique aux outils de simulation mécanique interactive pour la maquette numérique industrielle.* PhD thesis, Université Pierre et Marie Curie, Paris 6, 2009.

[Merlhiot2011] Xavier Merlhiot. On some industrial applications of time-stepping methods for nonsmooth mechanical systems : issues, successes and challenges. In *Euromech Colloquium [516] - Nonsmooth contact and impact laws in mechanics*, jul. 2011.

[Metta2006] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. Yarp : Yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1) :43–48, March 2006.

[Metta2010] Giorgio Metta, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, Luciano Fadiga, Claes von Hofsten, Kerstin Rosander, Manuel Lopes, José Santos-Victor, Alexandre Bernardino, and Luis Montesano. The icub humanoid robot : An open-systems platform for research in cognitive development. *Neural Networks*, 23(8-9) :1125 – 1134, 2010.

[Miller2004] A.T. Miller and P.K. Allen. Graspit ! a versatile simulator for robotic grasping. *IEEE Robotics Automation Magazine*, 11(4) :110–122, dec. 2004.

[Mirtich1996] Vincent Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* PhD thesis, University of California at Berkeley, 1996.

[Morel1994] G. Morel. *Programmation et adaptation de l'impédance de manipulateurs au contact.* PhD thesis, Université Pierre et Marie Curie, Paris 6, 1994.

[Morel1998] G. Morel, E. Malis, and S. Boudet. Impedance based combination of visual and force control. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 2, pages 1743–1748, may 1998.

[Muico2009] Uldarico Muico, Yongjoon Lee, J. Popović, and Zoran Popović. Contact-aware nonlinear control of dynamic characters. *ACM Transactions on Graphics*, 28(3) :1–9, jul. 2009.

[Muller2003] A. Müller and P. Maißer. A lie-group formulation of kinematics and dynamics of constrained mbs and its application to analytical mechanics. *Multibody System Dynamics*, 9 :311–352, 2003.

[Murray1994] R.M. Murray, Zexiang Li, and S.S. Sastry. *A mathematical introduction to robotic manipulation.* CRC, 1994.

[Nakanishi2007] J. Nakanishi, M. Mistry, and S. Schaal. Inverse dynamics control with floating base and constraints. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1942–1947, apr. 2007.

[NewtonDyn] Newton Game Dynamics Open-Source Physics Engine. http://newtondynamics.com/forum/newton.php.

[ODE] Open Dynamics Engine. http://www.ode.org/.

[OpenHRP] OpenHRP3 Official Site. http://www.openrtp.jp/openhrp3/en/index.html.

[Padois2005] Vincent Padois. *Enchaînement dynamiques de tâches pour des manipulateurs mobiles à roues*. PhD thesis, Ecole Nationale d'Ingénieurs de Tarbes, 2005.

[Park2001] Jonghoon Park, Youngjin Choi, Wan Kyun Chung, and Youngil Youm. Multiple tasks kinematics using weighted pseudo-inverse for kinematically redundant manipulators. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 4, pages 4041–4047, 2001.

[Park2005] Jonghoon Park. Principle of dynamical balance for multibody systems. *Multibody System Dynamics*, 14(3-4) :269–299, nov. 2005.

[Park2006] Jaeheung Park. *Control strategies for robots in contact*. PhD thesis, Stanford University, 2006.

[Park2007] J Park, J Han, and FC Park. Convex optimization algorithms for active balancing of humanoid robots. *IEEE Transactions on Robotics*, 23(4) :817–822, aug. 2007.

[Philippsen2009] Roland Philippsen, Negin Nejati, and Luis Sentis. Bridging the gap between semantic planning and continuous control for mobile manipulation using a graph-based world representation. In *1st Int. Work. on Hybrid Control of Autonomous Systems held in conj. with the Int. Joint Conf. on AI*, Pasadena, USA, jul. 2009.

[PhysX] GeForce® PhysX. http://www.geforce.com/Hardware/Technologies/physx.

[RobotCub] RobotCub. the robotcub project. http://www.robotcub.org/.

[Rubrecht2010a] S. Rubrecht, V. Padois, P. Bidaud, and M. de Broissia. *Constraint Compliant Control for a Redundant Manipulator in a Cluttered Environment*, pages 367–376. Slovenia, jun. 2010.

[Rubrecht2010b] S. Rubrecht, V. Padois, P. Bidaud, and M. De Broissia. Constraints compliant control : constraints compatibility and the displaced configuration approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems. Proceedings*, Taipei, Taiwan, oct. 2010.

[SAI] SAI Simulation and Active Interfaces. http://ai.stanford.edu/~conti/sai.html.

[Saab2011] Layale Saab. *Generating Whole Body Movements for Dynamic Anthropomorphic Systems under Constraints*. PhD thesis, Université de Toulouse III - Paul Sabatier, 2011.

[Salaun2010] C. Salaün. *Learning Models To Control Redundancy In Robotics*. PhD thesis, Université Pierre et Marie Curie, Paris 6, 2010.

[Samson1991] Claude Samson, Bernard Espiau, and Michel Le Borgne. *Robot Control : The Task Function Approach*. Oxford University Press, 1991.

[Sandini2007] G. Sandini, G. Metta, and D. Vernon. The icub cognitive humanoid robot : An open-system research platform for enactive cognition. In *50 Years of Artificial Intelligence*, Lecture Notes in Computer Science, chapter 32, pages 358–369. Springer, 2007.

[Sardain2004a] Philippe Sardain and Guy Bessonnet. Forces acting on a biped robot. center of pressure-zero moment point. *IEEE Transactions on Systems, Man and Cybernetics, Part A : Systems and Humans*, 34(5) :630–637, 2004.

[Sentis2005] Luis Sentis and Oussama Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(4) :505–518, dec. 2005.

[Sentis2007] Luis Sentis. *Synthesis and control of whole-body behaviors in humanoid systems*. PhD thesis, Stanford University, 2007.

[Shen2007] W. Shen and J. Gu. Multi-criteria kinematics control for the PA10-7C robot arm with robust singularities. In *IEEE International Conference on Robotics and Biomimetics*, pages 1242–1248, 2007.

[Shieh1988] L. S. Shieh, H. M. Dib, and S. Ganesan. Linear quadratic regulators with eigenvalue placement in a specified region. *Automatica*, 24 :819–823, nov. 1988.

[Siciliano1990] Bruno Siciliano. Kinematic control of redundant robot manipulators : A tutorial. *Journal of Intelligent & Robotic Systems*, 3 :201–212, 1990.

[Siciliano1991] Bruno Siciliano and J.J.E. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *5th International Conference on Advanced Robotics*, pages 1211–1216, 1991.

[Spirops] Axel Buendia. Spirops AI, a behavior production tool. http://www.spirops.com/SpirOpsAI.php.

[Stasse2008] O. Stasse, A. Escande, N. Mansard, S. Miossec, P. Evrard, and A. Kheddar. Real-time (self)-collision avoidance task on a hrp-2 humanoid robot. In *IEEE International Conference on Robotics and Automation*, pages 3200–3205, may 2008.

[Stewart1996] D. E. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal for Numerical Methods in Engineering*, 39(15) :2673–2691, 1996.

[Stramigioli2001a] Stefano Stramigioli. *Modeling and IPC control of interactive mechanical systems - A coordinate-free approach*. Springer Berlin / Heidelberg, 2001.

[Stramigioli2001b] Stefano Stramigioli and Herman Bruyninckx. *Geometry and Screw Theory for Robotics (Tutorial T9)*. 2001.

[Szewczyk1998] Jérôme Szewczyk. *Planification et contrôle de la manipulation coordonnée par distribution d'impédance*. PhD thesis, Université Pierre et Marie Curie, Paris 6, 1998.

[Tevatia2000] G. Tevatia and S. Schaal. Inverse kinematics for humanoid robots. In *IEEE International Conference on Robotics and Automation. Proceedings*, volume 1, pages 294–299, 2000.

[Trinkle2001] J. C. Trinkle, J. A. Tzitzouris, and J. S. Pang. Dynamic multi-rigid-body systems with concurrent distributed contacts. *Philosophical Transactions : Mathematical, Physical and Engineering Sciences*, 359(1789) :2575–2593, 2001.

[Tsagarakis2007] N.G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A.J. Ijspeert, M.C. Carrozza, and D.G. Caldwell. icub : the design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics*, 21(10), 2007.

[Uno1989] Y. Uno, M. Kawato, and R. Suzuki. Formation and control of optimal trajectory in human multijoint arm movement. *Biological Cybernetics*, 61 :89–101, 1989.

[Vukobratovic1973] Miomir Vukobratovic. How to control artificial anthropomorphic systems. *Systems, Man and Cybernetics, IEEE Transactions on*, 3(5) :497–507, 1973.

[Wang2006] Qiyu Wang, Jianping Yuan, and Zhanxia Zhu. The application of error quaternion and pid control method in earth observation satellite's attitude control system. In *1st International Symposium on Systems and Control in Aerospace and Astronautics*, pages 128–131, 2006.

[Webots] Webots™ 6 fast prototyping and simulation of mobile robots. http://www.cyberbotics.com/.

[Wieber2006] P.B. Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. *6th IEEE-RAS Humanoid Robots*, pages 137–142, 2006.

[XDE] XDE eXtended Dynamic Engine. xde-support@saxifrage.cea.fr.

[Yoshida2005] E. Yoshida, I. Belousov, C. Esteves, and J.-P. Laumond. Humanoid motion planning for dynamic tasks. In *IEEE/RAS International Conference on Humanoid Robots. Proceedings*, Tsukuba, Japan, 2005.

[Yuan1988] JS Yuan. Closed-loop manipulator control using quaternion feedback. *IEEE Journal of Robotics and Automation*, 4(4) :434–440, 1988.

[Zhang1985] Jianzhong Zhang, Nae-Heon Kim, and L. Lasdon. An improved successive linear programming algorithm. *Management Science*, 31(10) :1312–1331, oct. 1985.

[iCubSim] ICub Simulator Installation. http://eris.liralab.it/wiki/ODE.

A. Arboris-python

A.1. Modeling mechanical structures with Arboris-python

Section 1.2 addresses the simulation principles in Arboris-python. Here, the aim is to describe the construction of the world (the kinematic chains) and its constituting elements to obtain the matrices in Eq.(1.41) and achieve a good control of the system. Actually, one can consider that the **world** is a robotic system represented by a tree-structure graph, where the nodes are the **bodies** and the links are the **joints** connecting some **frames** of different bodies. The kinematic loops are modeled in Arboris-python with velocity **constraints**, typically the contacts between two **shapes**.

Hence, the bases of Arboris-python are set. The elements written in bold above are included in the `World` instance, and the methods that recover these data are given hereafter

```
from arboris.core import World  # import the World class from the core module.
world = World()                 # world instance, the core of the system, which
                                # gives access to the following elements:
bodies = world.getbodies()              # list of bodies
frames = world.getframes()              # ....... frames
joints = world.getjoints()              # ....... joints
shapes = world.getshapes()              # ....... shapes
constraints = world.getconstraints()    # ....... constraints
```

The variables `bodies, frames, joints, shapes` and `constraints` are **named lists**, which means that if the first body is named `"Arm"`, this instance can be obtained with the following lines [1]

```
body_Arm = bodies[0]
body_Arm = bodies["Arm"]
```

This method is case-sensitive, and that the position of bodies in the list is not straightforward due to the fact that tree-structures are not serial. Consequently, the use of the second line is encouraged.

It also regroups all information related to simulations and dynamics of the world. These parameters are

```
t      = world.current_time     # current time, updated by the simulation
M      = world.mass             # generalized mass matrix
N      = world.nleffects        # nonlinear effects matrix
gvel   = world.gvel             # generalized velocity of the world
ndof   = world.ndof             # total number of DoF
gforce = world.gforce           # generalized force of the world
```

where `t` is the current time, `M, N` are respectively the matrices M, N in Eq.(1.39), `gvel` is the vector $\dot{\boldsymbol{q}}$, `ndof` is the number of DoF in the `world` instance, and `gforce` is the input torque vector $\boldsymbol{\tau}$.

1. In python language, similarly to C++, containers index starts from 0.

But Arboris-python does not simply retrieve all these values, it also extends Eq.(1.39) with two other parameters,

```
1 B           = world.viscosity       # generalized viscosity matrix
2 admittance  = world.admittance      # admittance of the world
```

where `B` refers to the generalized viscosity matrix B (which leads to a fluid friction when multiplied by $\dot{\boldsymbol{q}}$), and `admittance` refers to the mechanical admittance of the system.

Now that the access to these elements is clarified, the following sections describe more precisely the world components.

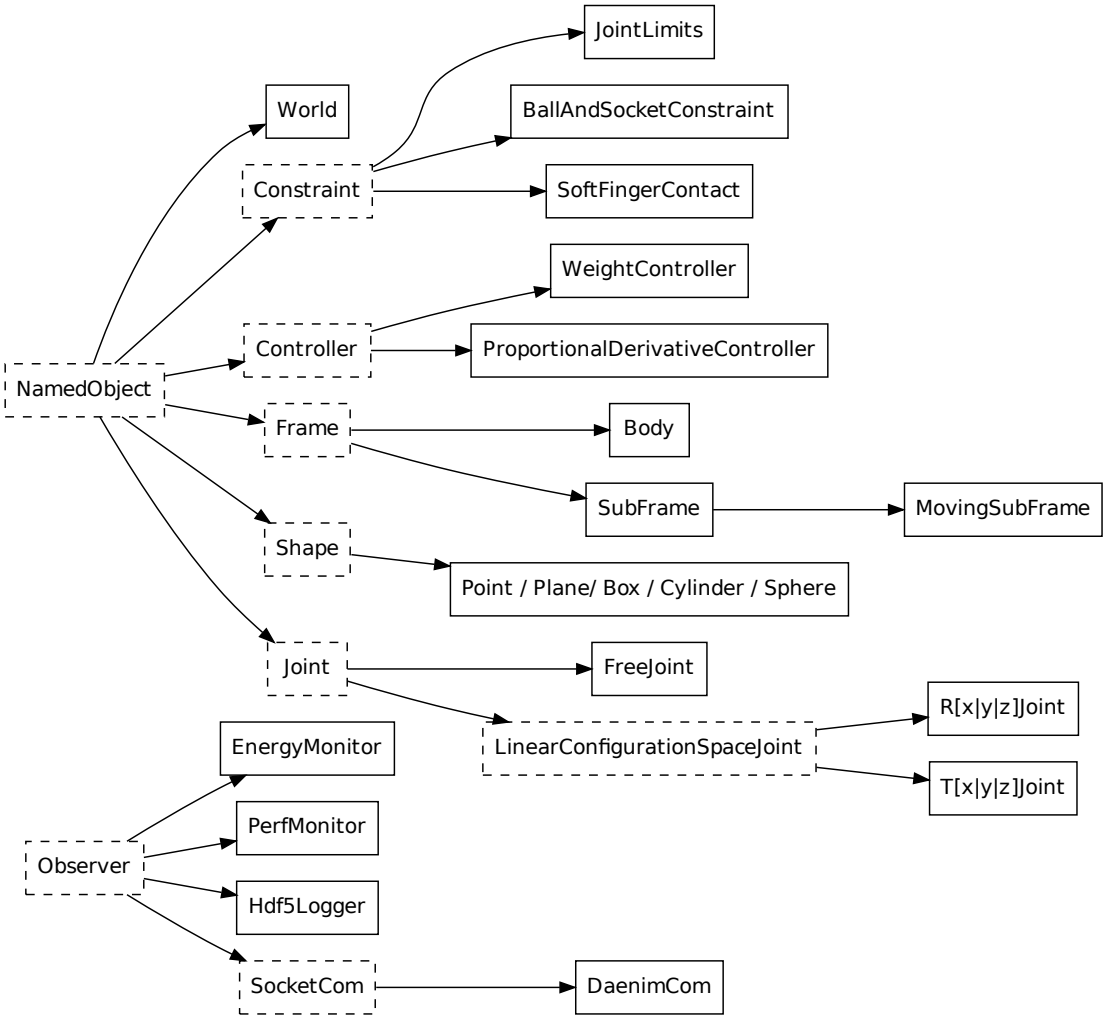


FIGURE A.1.: Arboris-python diagram. Dashed boxes represent abstract classes, and plain boxes represent concrete classes.

A.1.1. Modeling bodies

The Frame class The frame is **an abstract class** which defines the position and orientation of a particular part of the system relative to the Galilean reference frame of the world. The frame class has three concrete subclasses, `Body`, `SubFrame` and

`MovingSubFrame` (meaning that they can also be considered as `Frame`), and the methods and properties described below are common to all of them. The Galilean reference frame is defined by the ground frame Ψ_0 which is a `World` instance property

```
galilean_frame = world.ground
```

Actually, this parameter is a `Body` instance with no mass. The description of this class is detailed below.

In Arboris-python, each `Frame` Ψ_a is connected to one `Body` Ψ_b, and all frames instances (denoted by `frame`) contain the following properties

```
body      = frame.body          # parent body
pose      = frame.pose          # pose relative to the ground
bpose     = frame.bpose         # pose relative to its body
twist     = frame.twist         # twist
jacobian  = frame.jacobian      # Jacobian
djacobian = frame.djacobian     # time derivative Jacobian
```

The first line returns the pointer to the associated body, the second and third line return the frame poses relative to the ground and the body, which are the homogeneous matrices $H_{0,a}$ and $H_{b,a}$. The fourth line returns the frame twist relative to the ground expressed in its own coordinate system, which is denoted by $\boldsymbol{T}^a$, and the two last lines return the Jacobian and its time derivative $J_a(\boldsymbol{q})$ and $\dot{J}_a(\boldsymbol{q}, \dot{\boldsymbol{q}})$, which give the following equations

$$\begin{aligned}\boldsymbol{T}^a &= J_a(\boldsymbol{q})\dot{\boldsymbol{q}} \\ \dot{\boldsymbol{T}}^a &= J_a(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{J}_a(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}\end{aligned}$$

with $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ the generalized coordinates, velocity and acceleration vectors defined in Section 1.1.4.

The Body class This subclass incorporates the inertia and viscosity properties of the nodes contained in the tree-structure graph, and as a part of the kinematic chain, it gives information of its relations with the links in the graph. Given an instance `body` of class `Body`, these properties are accessible as follows

```
from arboris.core import Body           # import Body class.
from arboris.massmatrix import sphere   # import a function that computes the
                                        # predefined mass matrix of a sphere.

M_sphere = sphere(radius=.1, mass=1.)                       # SI units
body = Body(mass=M_sphere, viscosity=None, name="myBody")   # the body instance

M = body.mass                   # get inertia matrix
N = body.nleffects              # ....... nonlinear effects matrix
B = body.viscosity              # ....... viscosity matrix
p_joint  = body.parentjoint     # ....... parent joints (link in the graph)
c_joints = body.childrenjoints  # ....... children joints
```

where the matrices `M, N, B` $\in \mathbb{R}^{6\times6}$ represent respectively the inertia M^b, nonlinear effects N^b and viscosity matrices B^b of the body expressed in Ψ_b. `p_joint` is the `Joint` instance which represents the parent link in the tree-structure, and `c_joints` is a list containing the possible children `Joint` instances, detailed in the next section. The `world.ground` body is considered as the fist node of the tree-structure, so it is a `Body` instance with no parent joint.

When the `Body` class is instantiated (line 6), the first argument is the mass matrix expressed at the body frame Ψ_b, but the predefined mass matrices denoted by M^c (available in the module `arboris.massmatrix`) are all expressed at the inertial principal frame Ψ_c, which is generally not coincident with Ψ_b. The displacement from M^c to M^b is explained in Eq.(1.31), and this computation is left to the user discretion if he would like to use his own mass matrix. Arboris-python has a predefined method to compute this displacement. It is also possible to get the transformations from body frames to inertial principal frames based on mass matrices.

```
1  from arboris.massmatrix import sphere, transport, principalframe

3  M_c = sphere(radius=.1, mass=1.)  # mass expressed in center of mass
4  H_c_b = array([[1, 0, 0,.1],      # homogeneous matrix from center of mass to body
5                 [0, 0,-1, 0],
6                 [0,-1, 0,.2],
7                 [0, 0, 0, 1]])
8  M_b = transport(M_c, H_c_b)       # mass expressed in body
9  H_b_c = principalframe(M_b)       # homogeneous matrix from body to center of mass
```

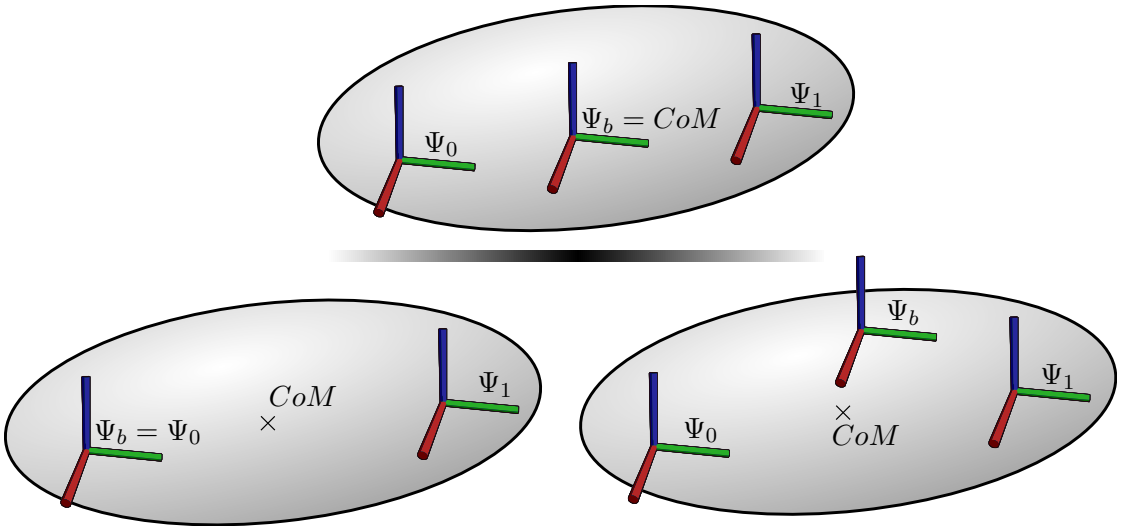


FIGURE A.2.: Presentation of three different constructions of the same body. On top, the body frame Ψ_b is located at the center of mass position, implying a diagonal mass matrix. At the bottom, the two other constructions show that Ψ_b can be at the joint frame ($\Psi_b = \Psi_0$) or anywhere else, the mass matrix being computed accordingly thanks to the method `transport`.

Unlike the constant matrices M^b and B^b, N^b has to be updated when the body configuration is modified, as it is shown in Eq.(1.28). Hence, it is possible to extend Eq.(1.28) and to write the Newton-Euler equation of rigid body [Murray1994] expressed in any frame Ψ_b

$$M^b \dot{\boldsymbol{T}}^b + \left(N^b + B^b\right) \boldsymbol{T}^b = \boldsymbol{W}^b \tag{A.1}$$

where $\boldsymbol{W}^b$ is the resulting wrench applied on this frame.

SubFrame & MovingSubFrame These classes allow to represent interesting frames other than bodies, and all of them must be connected to a body. They usually represent

some specific parts of robots, for instance the tips of the hand, some sketching elements of modeling, or some specific objectives for goals achievement during the robot control. The instantiation of these classes is done as follows

```
from arboris.core import SubFrame, MovingSubFrame
from arboris.homogeneousmatrix import transl
H_b_f = transl(.1,.2,.3)     # translation homogeneous matrix from body to subframe

frame = SubFrame(body=world.ground, bpose=H_b_f, name="myFrame") # bpose is constant

mframe = MovingSubFrame(world.ground, name="myMovingFrame")
mframe.bpose = H_b_f         # possible only with MovingSubFrame instance
```

A.1.2. Modeling joints

The RigidMotion Class This **abstract class** allows the representation of rigid motions between two frames belonging to different bodies (denoted by Ψ_p, Ψ_c for the parent and child frames). It regroups all the basic methods required by joints to describe their configurations, but these methods are also useful for studying any rigid motions of the world. Given `rm`, an instance of a `RigidMotion` subclass, one can access the following methods

```
pose      = rm.pose()        # its inverse: ipose = rm.ipose()
twist     = rm.twist()       # ............ itwist
adjoint   = rm.adjoint()     # ............ iadjoint
adjacency = rm.adjacency()   # ............ iadjacency
dadjoint  = rm.dadjoint()    # ............ idadjoint
```

which respectively return the pose $H_{p,c}$ from Ψ_p to Ψ_c, the twist $\boldsymbol{T}^c_{p,c}$ of Ψ_c relative to Ψ_p expressed in Ψ_c, the adjoint $Ad_{p,c}$ built from $H_{p,c}$, the "adjacency" matrix $ad^c_{p,c}$ built from the twist $\boldsymbol{T}^c_{p,c}$ and finally the time derivative of the adjoint $\dot{A}d_{p,c}$. The two last variables are useful to compute the derivative of the twist $\dot{\boldsymbol{T}}^c_{p,c}$ computed such as

$$\boldsymbol{T}^c_{p,c} = \begin{bmatrix} \boldsymbol{\omega}^c_{p,c} \\ \boldsymbol{v}^c_{p,c} \end{bmatrix} \qquad ad^c_{p,c} = \begin{bmatrix} \widehat{\boldsymbol{\omega}}^c_{p,c} & 0 \\ \widehat{\boldsymbol{v}}^c_{p,c} & \widehat{\boldsymbol{\omega}}^c_{p,c} \end{bmatrix} \qquad \dot{A}d_{p,c} = Ad_{p,c}ad^c_{p,c}$$

$$\dot{\boldsymbol{T}}^p_{p,c} = \dot{A}d_{p,c}\boldsymbol{T}^c_{p,c} + Ad_{p,c}\dot{\boldsymbol{T}}^c_{p,c} \tag{A.2}$$

The Joint Class The `Joint` class is also an **abstract subclass** which is derived from the `RigidMotion` class extended with an implementation of the joint properties, by constraining the relative motions between two frames Ψ_p (parent) and Ψ_c (child). This implementation is based on ideal joints described in Section 1.1.4.1, and in addition to the `RigidMotion` methods it gives access to the following properties

```
gpos      = joint.gpos          # generalized coordinates in joint subspace
gvel      = joint.gvel          # generalized velocity
ndof      = joint.ndof          # number of DoF
dof       = joint.dof           # slice of the DoF in the world instance
jacobian  = joint.jacobian      # inner Jacobian
djacobian = joint.djacobian     # inner time derivative Jacobian
```

where `joint` is an instance of a concrete subclass. `gpos` and `gvel` are the generalized position and velocity [2], referring respectively to Q_j and $\boldsymbol{\nu}_j$ of joint j (see the joint description in Section 1.1.4.1), `ndof` is its number of DoF and `dof` is a slice which maps its DoF into the generalized velocity parameter of the world. The terms `jacobian` and `djacobian` are improperly used because it does not compute a twist from the generalized velocity vector but the matrix $X(Q_j)$ and its derivative instead, such as $\boldsymbol{T}^c_{p,c} = X(Q_j)\boldsymbol{\nu}_j$.

An abstract subclass called `LinearConfigurationSpaceJoint` is defined to parameterize the configuration of joint j with a vector $\boldsymbol{q}_j \in \mathbb{R}^{n_{dof}}$ instead of the matrix Q_j. It describes most of the joints present in the real world, and their configurations integration with respect to time is simpler.

The following concrete subclasses are implemented in Arboris-python :

- `TxJoint,TyJoint,TzJoint` are prismatic joints along the axes x, y and z,
- `RxJoint,RyJoint,RzJoint` are revolute joints along the axes x, y and z,
- `RzRyJoint,RzRxJoint,RyRxJoint` are the composition of two consecutive revolute joints respectively along the axes y then z, x then z, and x then y,
- `RzRyRxJoint` is the composition of three consecutive revolute joints along axes x, y then z,
- `FreeJoint` is a non-constrained motion between two bodies.

Notice that except for the `FreeJoint` which is a `Joint` subclass, all other joints are `LinearConfigurationSpaceJoint` subclasses. The following snippet of code presents some examples on how to instantiate these concrete classes

```
from arboris.joints import RyRxJoint, FreeJoint
joint = RyRxJoint(gpos=[.1,.2], gvel=[.0,.1], name="theRyRxJoint")

gpos = array([[1,0,0,.1],
              [0,1,0,.2],
              [0,0,1,.3],
              [0,0,0, 1]])
free_joint = FreeJoint(gpos=gpos, gvel=[.1,.2,.3,.4,.5,.6], name="theFreeJoint")
```

A.1.3. Modeling shapes

In Arboris-python shapes are some simple primitives used either to decorate the visualization and to determine if contacts or collisions occur during simulations.

The subclasses of the abstract class `Shape` already defined in Arboris-python are the `Point`, the `Plane`, the `Sphere`, the `Cylinder` and the `Box`. Given `frame` an instance of a `Frame` defined above, they are instantiated as follows

```
from arboris.shapes import Point, Plane, Sphere, Cylinder, Box
shape = Point(frame=frame, name="thePoint")
shape = Plane(frame=frame, coeffs=(0,1,0,.1), name="thePlane")
shape = Sphere(frame=frame, radius=.1, name="theSphere")
shape = Cylinder(frame=frame, length=1., radius=.1, name="theCylinder") # along z axis
shape = Box(frame=frame, half_extents=(.1, .2, .3), name="theBox")
```

2. In a general manner, the generalized coordinates cannot be described by an element of a vector space. This prevents the use of the variables $\boldsymbol{q}_j$ and $\dot{\boldsymbol{q}}_j$.

A.1.4. Modeling constraints

The tree-structure modeling of a system is not sufficient when the graph is composed of kinematic loops. Several methods have been developed to cope with this problem. In Arboris-python, these kinematic loops are treated through the **abstract class** `Constraint` which generates a virtual wrench between two frames of different bodies to ensure that the related closure is properly performed. These constraints can be unilateral, *e.g.* contacts with the environment, or bilateral, *e.g.* ball and socket constraints. For a given concrete instance `const` derived from the `Constraint` class, it is possible to access the following methods and properties

```
const.ndol          # number of degrees of 'linkage' = 6 - ndof
const.gforce        # generalized force generate by the constraint
const.jacobian      # Jacobian of the constraint

const.enable()      # to enable the constraint
const.disable()     # to disable .............
const.is_enabled()  # return the state of the constraint: True/False
const.is_active()   # return whether the kinematic loop closure is active or not
```

`ndol` is the number of "degrees of linkage". If `ndof` is the number of DoF not constrained in this kinematic loop, then one has `ndol = 6 - ndof`. The parameter `gforce` refers to the generalized force applied by the constraint to the system $\boldsymbol{\tau}_c$, and finally the parameter `jacobian` is J_c which gives the link between $\boldsymbol{\tau}_c$ and the wrench generated by the constraint $\boldsymbol{W}^c$ such as $\boldsymbol{\tau}_c = J_c^{\mathsf{T}} \boldsymbol{W}^c$.

The methods `enable`, `disable` and `is_enabled` set or get information about the user-defined **availability** of the constraint. Note that these methods are different from `is_active` which returns the **activity** of the kinematic loop closure induced by the constraint, *i.e.* when the two frames are close enough and the closure has to be satisfied.

Three subclasses are provided in Arboris-python. `JointLimits` bounds the range of joints that evolve in linear configuration spaces, `BallAndSocketConstraint` forces the position of two frames to be coincident, and `SoftFingerContact` simulates frictional contact. They are instantiated as follows

```
joint = ... # must be concrete LinearConfigurationSpaceJoint subclass
const = JointLimits(joint=joint, min=min_values, max=max_values, name="theJointLimits")

f0,f1 = ... # a couple of frames
const = BallAndSocketConstraint(frames=(f0,f1), name="theBaSConstraint")

s0,s1 = ... # a couple of shapes
const = SoftFingerContact(shapes=(s0,s1), friction_coeff= 1., name="theSFContact")
```

However, for now the `SoftFingerContact` constraint has limited collision solvers, meaning that only some couples of shapes are candidates. Hopefully, this module can easily be extended with user-defined collision solvers.

A.1.5. Modeling structures

All the tree-structures elements are presented above, it remains to explain how to build kinematic chains. The connection between two bodies is done through the method `add_link` of the `world` instance as follows

```
f_parent = ... # parent frame
f_child  = ... # child frame
```

```
joint    = ... # joint which create the link
world.add_link(f_parent, joint, f_child)
```

The use of this method ensures that no kinematic loop is created accidentally with any `Joint` instance in the graph.

However, the user has to know that this method only registers the elements of the system that are useful to create the structure, which means that some elements need to be registered afterward. For example, the frames used as arguments in the `add_link` method become part of the structure and are directly registered with their related bodies, but the remaining frames, shapes and constraints are unknown by the `world` and are left over as illustrated in Fig. A.3.

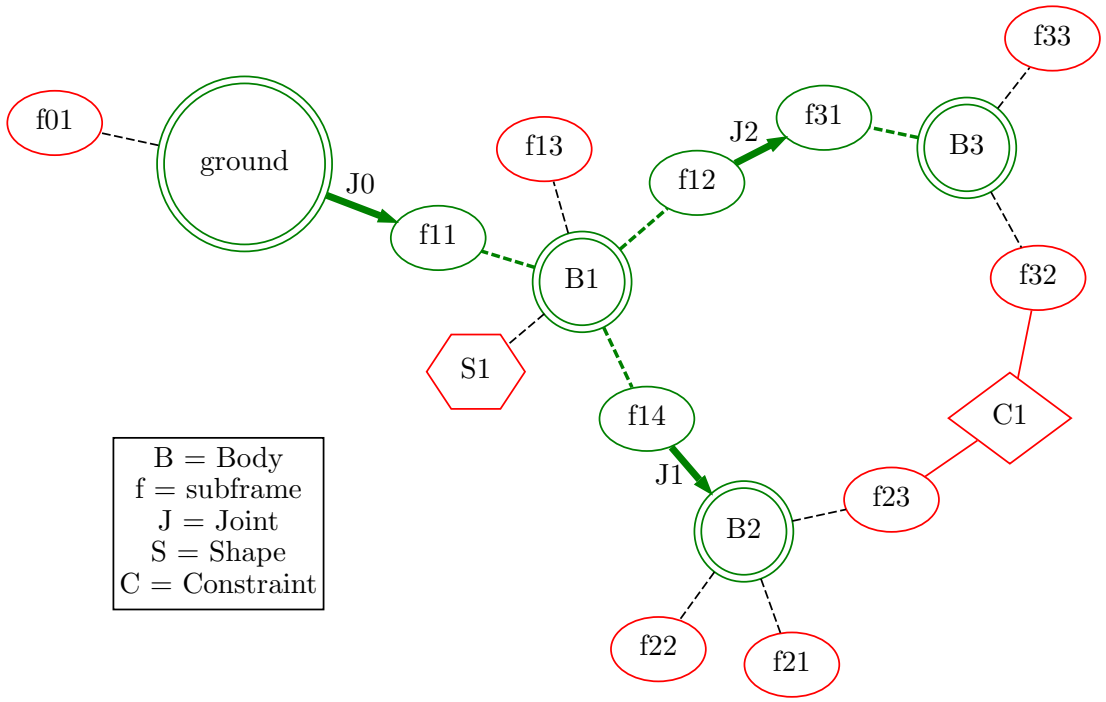


FIGURE A.3.: Modeling a structure composed of three bodies. The joints create connections between frames of different bodies thanks to the `add_link` method. The elements in green are then registered in the world. It remains to register the red elements.

They must be registered, and in this purpose one has to call the method `register` of the `world` instance

```
f_shape = ... # frame that supports a shape
shape   = ... # a shape
const   = ... # a constraint
world.register(f_shape)
world.register(shape)
world.register(const)
...
```

If the kinematic chain is built correctly, one can call the following methods without raising any errors

```
world.init()                # initialize the world. should be called when ...
                            # ... the modeling of the structure is finished.
```

```
world.update_geometric()     # compute forward kinematics
world.update_dynamic()       # compute forward dynamics
```

where `init` method initializes all the `world` parameters, `update_geometric` computes the forward kinematics and gives access to all the `pose` properties (but not the `twist` ones), and the last call computes the forward velocity model and also updates the matrices related to the dynamic of the chain.

A.1.6. Controlling a multibody system : Controller class

Implementation of a Controller The modeling of the robot gives access to the geometric, kinematic and dynamic data, but without any control it is just a snapshot of the world. Here, the purpose of controllers is to generate appropriate sets of wrenches at the joint level, which means some generalized force vectors to achieve a desired goal or behavior.

Arboris-python allows the control of the system through the derivation of the **abstract class** `Controller`. A minimalist subclass, for instance which does nothing, can be written as follows

```
from arboris.core import Controller     # basic Controller class
from numpy import zeros                 # to generate arrays composed of 0

class MinimalistController(Controller):
    def __init__(self):
        pass
    def init(self, world):
        self.world = world      # get a pointer of the world
    def update(self, dt):       # dt is the time step of the simulation
        n = self.world.ndof     # get the number of DoF in the world
        gforce = zeros(n)       # generalized force generated by the controller
        imp    = zeros((n,n))   # the 'impedance' generated by the controller
        return gforce, impedance
...
world.register(MinimalistController())
```

There are three mandatory methods. The first one, `__init__`, is the class constructor and may have several user-defined arguments, the second one, `init`, initializes the controller relative to the `world` instance with fixed arguments, and the third one, `update`, is the main function which updates the input generalized force, again with fixed arguments.

One has to keep in mind the following points : the `init` method is called by the `init` method of the `world` instance, and thus may be called more than one time before the beginning of the simulation, then the `update` method is called only once for each simulation loop with the time step argument `dt`, and finally one can of course add many other user-defined methods. Like the other world components, the controllers have to be registered to be taken into account during the simulation, as written in the last line of the snippet of code above.

Hence, the dynamic control of a particular joint is possible, for instance through the use of the following subclass

```
from arboris.core import Controller
from numpy import zeros

class MyController(Controller):
    def __init__(self, joint_name):
        self.name = joint_name                    # save the name of the joint
    def init(self, world):
```

```
8           self.world = world                          # get the pointer of the world
9           self.joint = world.getjoints()[self.name]   # ..................... joint
10      def update(self, dt):
11          n = self.world.ndof
12          gforce = zeros(n)
13          imp    = zeros((n,n))
14          gforce[self.joint.dof] = self._get_objective(dt)    # set the desired force
15          return gforce, impedance
16  ...
17  world.register(MyController("Shoulder"))
```

where the method `get_objective` at line 14 has to be implemented by the user. Of course, more complex controllers can easily be written, for example an inverse dynamic controller [Nakanishi2007, Salaun2010]

```
1   from arboris.core import Controller
2   from numpy import zeros, dot

4   class InverseDynamicController(Controller):
5       def __init__(self, frame_name, goal):
6           self.name = frame_name                      # save frame name
7           self.goal = goal                            # .... frame goal
8       def init(self, world):
9           self.world = world                          # get the pointer of the world
10          self.frame = world.getframes()[self.name]   # ..................... frame
11      def update(self, dt):
12          n = self.world.ndof
13          M  = self.world.mass                # generalized mass matrix
14          N  = self.world.nleffects           # nonlinear effect matrix
15          dq = self.world.gvel                # generalized velocity vector
16          ddq_des = self._get_ddq_des(self.frame, self.goal)  # has to be user-defined
17          gforce = dot(M, ddq_des) + dot(N, dq)
18          imp    = zeros((n,n))
19          return gforce, impedance
```

Again, the method `_get_ddq_des` at line 16 has to be defined by the user to get the desired generalized acceleration vector.

Some predefined controllers are supplied with Arboris-python. For instance, one can use the `WeightController` which simulates the effects of gravity on the system, or the `ProportionalDerivativeController` to quickly control some joints toward some desired goals with virtual spring-damper systems.

A.1.7. Observing the world : Observer class

Unlike controllers, the instances of the **abstract class** `Observer` (as the name suggests) should not interfere with the conduct of the simulation, *i.e.* its only purpose should be to print, save or spread some information resulting from the evolution of the system. But whereas controllers have no information about the starting and ending times, observers have access to the simulation timeline and can allocate memory before the beginning of the simulation. A simple observer is described hereafter

```
1   from arboris.core import Observer

3   class MinimalistObserver(Observer):
4       def __init__(self):                 # constructor, with user-defined arguments
5           pass
6       def init(self, world, timeline):    # initialization, with fixed arguments.
7           self.joints = world.getjoints() # get the pointer to the joints list
8       def update(self, dt):
9           for j in self.joints:           # loop which print for each joint:
```

```
10              print j.name, j.gpos          # its name and its generalized position.
11      def finish(self):                     # finish method: must be always overwrite,
12          pass                              # to finalize some procedures.
```

Its structure is quite similar to the `Controller` class, plus the timeline argument at line 6 and the `finish` method at line 11. Some predefined observers are supplied with Arboris-python, for instance the `EnergyMonitor` to plot the mechanical, potential and kinetic energies of the system with respect to time, the `PerfMonitor` to print information about the time performance of the simulation process, the `Hdf5Logger` to save the simulation data in a hdf5 file, and the `DaenimCom` to connect the simulation with an external viewer which shows the evolution of the world during the simulation.

As an observer is not part of the world, it must be used in the simulation loop as described in the simulation method included in the `arboris.core` module

```
1  def simulate(world, timeline, observers):
2      world._current_time = timeline[0]   # initialize the current time
3      world.init()                        # .......... the world
4      for obs in observers:               # .......... all the observers
5          obs.init(world, timeline)
6      for next_time in timeline[1:]:      # simulation loop:
7          dt = next_time - world._current_time    # compute time interval dt
8          world.update_dynamic()          # update dynamic of the world
9          world.update_controllers(dt)    # ...... generalized force of controllers
10         world.update_constraints(dt)    # ...... constraints for kinematic loops
11         for obs in observers:           # ...... all the observers
12             obs.update(dt)
13         world.integrate(dt)             # integrate the world to get new configuration
14     for obs in observers:               # finalize all the observations
15         obs.finish()
```

where lines 2 to 5 initialize the simulation components, lines 14 and 15 execute the observers post-process, and lines 6 to 13 run the core of the simulation loop.

Notice that user-defined methods can be used to perform the simulation, as long as the order of methods `update_dynamic`, `update_controllers`, `update_constraints` and `integrate` is respected.

A.2. Examples

A.2.1. Modeling and simulating a simple robot

The following snippet of code creates a robot composed of three revolute joints, and simulates its motion when it is not actuated and falls down due to gravity.

```
1  ############## CREATE THE WORLD ##############
2  from arboris.core import World, Body, SubFrame
3  from arboris.joints import RzJoint              # Revolute joint about z axis
4  from arboris.homogeneousmatrix import transl    # translation homogeneous matrix
5  from arboris.massmatrix import box              # return box mass matrix

7  L = .1                                          # rod half length (meter)
8  m = 1.                                          # rod mass (kg)
9  M = box(half_extents=(L, L/10., L/10.), mass=m) # 6x6 mass matrix defined at CoM

11 w = World()                                     # create the world

13 B0 = Body(name="B0", mass=M)                    # create bodies
14 B1 = Body("B1", M)
15 B2 = Body("B2", M)
```

```
17  J0 = RzJoint(gpos=[0.], gvel=[0.], name="J0")    # create joints
18  J1 = RzJoint(name="J1")
19  J2 = RzJoint(name="J2")

21  SF00 = SubFrame(body=B0, bpose=transl(-L,0,0), name="SF00") # create subframes
22  SF01 = SubFrame(B0, transl( L,0,0), name="SF01")
23  SF10 = SubFrame(B1, transl(-L,0,0), name="SF10")
24  SF11 = SubFrame(B1, transl( L,0,0), name="SF11")
25  SF20 = SubFrame(B2, transl(-L,0,0), name="SF20")
26  SF21 = SubFrame(B2, transl( L,0,0), name="SF21")

28  w.add_link(w.ground, J0, SF00)                  # create the kinematic chain
29  w.add_link(SF01    , J1, SF10)
30  w.add_link(SF11    , J2, SF20)
31  w.register(SF21)                # do not forget to register the tree leaves
32  w.init()                        # IMPORTANT to create the DoF in the world
33  w.update_dynamic()              # update the dynamic properties


36  ############### MODIFY INITIAL POSITION ##############
37  joints = w.getjoints()
38  joints[0].gpos[:] = [0.]        # be aware that joints 0 may be different of J0
39  joints["J1"].gpos[:] = [0.2]    # it changes the joint with name "J1" (safer)
40  joints["J1"].gvel[:] = [0.1]


43  ############### ADD CONTROLLERS ##############
44  from arboris.controllers import WeightController
45  w.register(WeightController(gravity=-9.81, name="weight"))


48  ############### ADD OBSERVERS ##############
49  from arboris.observers import PerfMonitor, Hdf5Logger, DaenimCom
50  observers = []
51  observers.append(PerfMonitor(log=True))             # display sim performance
52  observers.append(Hdf5Logger("sim.h5", mode='w'))    # save sim data in hdf5 file

54  from arboris.visu_collada import write_collada_scene
55  write_collada_scene(w, "scene.dae")                 # write scene in dae file
56  observers.append(DaenimCom(r"daenim", "scene.dae")) # link deanim with scene ...
57                                                      # ... for visualization

59  ############### SIMULATE THE WORLD ##############
60  from arboris.core import simulate
61  from numpy import arange
62  simulate(w, arange(0, 1., .001), observers) # simulate from 0 to 1s (dt= 0.001s)


65  ############### END OF SIMULATION: RESULTS ##############
66  print observers[0].get_summary() # print summary of simulation performance

68  from arboris.visu_collada import write_collada_animation
69  write_collada_animation("anim.dae", "scene.dae", "sim.h5") # save dae animation
```

A.2.2. Add shapes and constraints

This example shows how to add some shapes in the world, and how to retrieve them. It also shows how to add some constraints, for instance a joint limit or a frictional contact between two shapes.

```
1  ############### CREATE THE WORLD ##############
2  from arboris.core import World
3  from arboris.homogeneousmatrix import transl    # translation homogeneous matrix
4  from arboris.robots import simplearm            # model of a 3R robot
```

```
5  from arboris.shapes import Plane, Sphere

7  w = World()                      # create the world
8  simplearm.add_simplearm(w)       # add 3R robot

10 sh = Sphere(w.getframes()["EndEffector"], radius=.1, name="sphere") # add sphere
11 w.register(sh)                                                      # register
12 w.register(Plane(w.ground, (0,1,0,-.6), name="floor"))  # add a floor

14 w.update_dynamic()               # update the dynamic properties


17 ############## MODIFY INITIAL POSITION ##############
18 joints = w.getjoints()
19 shapes = w.getshapes()
20 joints["Shoulder"].gpos[:] = [1.5]  # change initial configuration
21 joints["Elbow"].gpos[:] = [1.5]
22 joints["Wrist"].gpos[:] = [1.5]

24 from arboris.constraints import JointLimits, SoftFingerContact
25 w.register(JointLimits(joints["Elbow"], -2, 2)) # Limit joint range
26 w.register(SoftFingerContact([shapes["sphere"], shapes["floor"]], 1.)) # contact

28 consts = w.getconstraints()         # if needed later for user defined methods


31 ############## ADD CONTROLLERS ##############
32 from arboris.controllers import WeightController
33 w.register(WeightController(gravity=-9.81, name="weight"))


36 ############## ADD OBSERVERS ##############
37 from arboris.observers import PerfMonitor, Hdf5Logger, DaenimCom
38 observers = []
39 observers.append(PerfMonitor(log=True))


42 ############## SIMULATE THE WORLD ##############
43 from arboris.core import simulate
44 from numpy import arange
45 simulate(w, arange(0, 1., .001), observers)
```

B. LQP-based control

This chapter briefly presents the LQP-based controller developed throughout this thesis. Like Arboris-python, this controller has been written in Python and is available at this address [LQPctrl]. The purpose of this chapter is to introduce the ways to instantiate the controller, its tasks and its constraints, through some examples where an iCub robot is controlled in Arboris-python. They are minimalist scripts derived from the experiments in Section 3. One can refer to the examples included in the `LQPctrl` package for more details about this controller.

B.1. LQP-based controller parameters

The LQP-based controller can handle many constraints and perform many tasks. However, it requires to set some parameters to work properly. Default values are already defined to ease the handling of the controller, but all the available options are described in the following snippet of code to allow more precise control.

```
lqpc = LQPcontroller(
    gforcemax,      # {'jointname': torquemax} set torque limit
    dgforcemax,     # {'jointname': dtorquemax} set torque derivative limit
    qlim,           # {'jointname': (qmin, qmax)} set position limit
    vlim,           # {'jointname': vmax} set velocity limit
    tasks,          # list of tasks
    events,         # list of events
    data, options, solver_options,    # see below
    name=None)      # controller name

data = {
    'bodies'          : None,   # body list to get CoM, ZMP and const Jacobian
    'Hfloor'          : eye(4), # floor orientation, for walking task
    'cop constraints' : []}     # to compute the CoP of the system.

options = {
    'pos horizon'      : None,    # anticipation coefficient for position const
    'vel horizon'      : None,    # anticipation coefficient for velocity const
    'avoidance horizon': None,    # anticipation coefficient for avoidance const
    'avoidance margin' : 0.,      # distance margin for avoidance const
    'npan'          : 8,          # number of side of linearized friction cone
    'solver'        : 'cvxopt',   # for now, the only solver available
    'base weights'  : (1e-7, 1e-7, 1e-7), # weights of whole minimization task T0
    'cost'          : 'normal',# or 'wrench consistent'  # tasks cost
    'norm'          : 'normal',# or 'inv(lambda)'        # tasks normalization
    'formalism'     : 'dgvel chi'} # or 'chi'            # tasks formalism

solver_options = {
    'show_progress' : True,   # show details of cvxopt resolution
    'abstol'        : 1e-8,   # absolute tolerance
    'reltol'        : 1e-7,   # relative tolerance
    'feastol'       : 1e-7,   # feasible tolerance
    'maxiters'      : 100}    # maximum iterations number
```

B.2. Controlling iCub with LQP-based controller

The following snippet of code shows how to create an iCub robot in the `World` instance with unilateral contact constraints on the floor, and how to control it with the LQP-based controller.

```
1  from arboris.robots import icub
2  from numpy import pi, arange
3  from arboris.controllers import WeightController
4  from arboris.observers import PerfMonitor
5  from arboris.core import World, simulate
6  from arboris.shapes import Plane
7  from arboris.constraints import SoftFingerContact

9  ############## CREATE WORLD & INITIALIZATION ##############
10 w = World()                                    # create the world
11 w._up[:] = [0,0,1]                             # set the "up" vector along z
12 icub.add(w)                                    # add icub in world

14 w.register(Plane(w.ground, (0,0,1,0), "floor")) # create the floor
15 w.register(WeightController())                 # add gravity

17 joints = w.getjoints()
18 shapes = w.getshapes()

20 joints['root'].gpos[0:3,3] = [0,0,.598]        # modify initial joint positions
21 for name, value in [('shoulder_roll', pi/8), ('elbow_pitch', pi/8),
22                      ('knee'         , -.1 ), ('ankle_pitch', -.1 )]:
23     joints['l_'+name].gpos[:] = value
24     joints['r_'+name].gpos[:] = value

26 floor_const = [SoftFingerContact((shapes[s], shapes['floor']), 1.5, name=s)
27                for s in ['lf1', 'lf2', 'lf3', 'lf4', 'rf1', 'rf2', 'rf3', 'rf4']]
28 for c in floor_const:                           # create contact constraints ...
29     w.register(c)                               # ... between feet and floor

31 ############## CREATE TASKS, EVENTS & LQP controller ##############
32 add_lqp_controller(w)

34 ############## SET OBSERVERS & SIMULATE ##############
35 simulate(w, arange(0,3,0.01), obs)
```

Note that at line 32, the method `add_lqp_controller(w)` must be user-defined. It must integrate the tasks to perform by the controller and register the latter in the `World`. The next sections give some examples about this method implementation.

B.2.1. Standing

The following code allows to perform a basic task, that is to keep the current posture and to stand upright.

```
1  def add_lqp_controller(w):
2      from arboris.robots import icub
3      joints = w.getjoints()
4      icub_joints = [joints[n] for n in icub.get_joints_data()] # get icub joints
5      pose = [j.gpos[0] for j in icub_joints]                   # save config

7      ############## TASKS ##############
8      from LQPctrl.task import MultiJointTask
9      from LQPctrl.task_ctrl import KpCtrl
10     tasks = []
11     tasks.append(MultiJointTask(
12                     icub_joints,            # joints controlled
```

```
13                      KpCtrl(pose, Kp=10),     # KP task controller
14                      [],                      # DoF to control ([] means all)
15                      1,                       # weight (for weighting)
16                      0,                       # level  (for hierarchy)
17                      True,                    # activity
18                      "standing_pose"))        # name

20      ############## LQP CONTROLLER ##############
21      from LQPctrl.LQPctrl import LQPcontroller
22      gforcemax = icub.get_torque_limits()                # joint torque limit
23      lqpc = LQPcontroller(gforcemax, tasks=tasks)        # create controller
24      w.register(lqpc)                                    # register controller
```

B.2.2. Swinging

The purpose of this code is to perform a swinging task by controlling the ZMP of the iCub robot.

```
1   def add_lqp_controller(w):
2       from arboris.robots import icub
3       joints = w.getjoints()
4       bodies = w.getbodies()
5       icub_joints = [joints[n] for n in icub.get_joints_data()] # get icub joints
6       icub_bodies = [bodies[n] for n in icub.get_bodies_data()] # get icub bodies
7       pose = [j.gpos[0] for j in icub_joints]                   # save  config

9       ############## ZMP TRAJECTORY ##############
10      from numpy import zeros, arange, sin, pi
11      T, dt, amp  = 1., .01, .02    # period (s), sampling time (s), amplitude (m)
12      t = arange(0, 3., dt)
13      y = amp*sin(t*2*pi/T)         # create a sinusoidal trajectory along y
14      zmp_traj = zeros((len(y),2))  # no trajectory along x
15      zmp_traj[:,1] = y

17      ############## TASKS ##############
18      from LQPctrl.task import MultiJointTask, CoMTask
19      from LQPctrl.task_ctrl import KpCtrl, ZMPCtrl
20      tasks = []
21      # create the ZMP controller
22      zmp_ctrl = ZMPCtrl(zmp_traj,    # the ZMP trajectroy defined above
23                         QonR=1e-6,   # ratio between state and input tracking
24                         horizon=1.7, # horizon of tracking
25                         dt=dt,       # sampling time
26                         cdof=[0,1])  # give trajectory plane: here (0=>x,1=>y)
27      # add the ZMP (swinging) task
28      tasks.append(CoMTask(icub_bodies, zmp_ctrl, [0,1], 1, 0, True, "swinging"))
29      # add the posture task
30      tasks.append(MultiJointTask(icub_joints, KpCtrl(pose, 10),
31                                  [], 1e-4 , 0, True, "pose"))

33      ############## LQP CONTROLLER ##############
34      from LQPctrl.LQPctrl import LQPcontroller
35      gforcemax = icub.get_torque_limits()                # joint torque limit
36      lqpc = LQPcontroller(gforcemax, tasks=tasks)        # create controller
37      w.register(lqpc)                                    # register controller
```

B.2.3. Walking

This last example shows how to perform a walking task with a bipedal robot. The algorithm is mainly based on [Wieber2006].

```
1   def add_lqp_controller(w):
```

```
    from arboris.robots import icub
    w.update_dynamic()
    joints = w.getjoints()
    bodies = w.getbodies()
    frames = w.getframes()
    consts = w.getconstraints()
    icub_joints = [joints[n] for n in icub.get_joints_data()] # get icub joints
    icub_bodies = [bodies[n] for n in icub.get_bodies_data()] # get icub bodies
    pose = [j.gpos[0] for j in icub_joints]                   # save  config
    l_const = [c for c in consts if c.name[0:2] == 'lf']      # get l foot const
    r_const = [c for c in consts if c.name[0:2] == 'rf']      # get r foot const
    l_frame, r_frame = frames['l_sole'], frames['r_sole']     # get feet frames

    ############## WALKING PARAMETERS ##############
    from arboris.homogeneousmatrix import rotx
    from numpy import pi
    goal = {"action": "goto", "pos": [-0.2,0]}     # goal info
    zmp = {'QonR':1e-6, 'horizon':1.7, 'dt':.01, 'cdof':[0,1]} # zmp info
    # information on feet trajectory tracking:
    feet = {"Kp":150, "Kd":None,                   # KP params
            "l_frame":l_frame, "r_frame":r_frame,  # feet frames
            "l_const":l_const, "r_const":r_const,  # feet consts
            "weight":1,                            # tracking weight
            "R0":rotx(pi/2.)}                      # floor orientation
    # information on steps:
    step={"length":.05, "side":.05, "height":.01,  # step distances (m)
          "time": .8,                              # period for one step (s)
          "ratio":.7,                              # simple/double support ratio
          "start": "left"}                         # starting foot

    ############## TASKS ##############
    from LQPctrl.task import MultiJointTask, JointTask
    from LQPctrl.task_ctrl import KpCtrl
    from LQPctrl.walk import WalkingCtrl, WalkingTask

    tasks = []
    # create the walking controller
    wctrl = WalkingCtrl(goal, zmp, feet, step)
    # add the walking task
    tasks.append(WalkingTask(icub_bodies, wctrl, [], 1., 0, True, "walk"))
    # add the posture task
    tasks.append(MultiJointTask(icub_joints, KpCtrl(pose, 10),
                                [], 1e-4 , 0, True, "pose"))

    ############## LQP CONTROLLER ##############
    from LQPctrl.LQPctrl import LQPcontroller
    gforcemax = icub.get_torque_limits()                # joint torque limit
    lqpc = LQPcontroller(gforcemax, tasks=tasks)        # create controller
    w.register(lqpc)                                    # register controller
```