UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

Motion Control of the Hybrid Wheeled-Legged Quadruped Robot Centauro

by

Arturo Laurenzi

Thesis submitted for the degree of *Doctor of Philosophy* (32nd cycle)

December 2019

Nikolaos G. Tsagarakis	Supervisor
Ferdinando Cannata	Head of the PhD program

Thesis Jury:

Prof. Auke Ijspeert, *Ecole Polytechnique Fédérale de Lausanne*	External examiner
Prof. Fabrizio Caccavale, *Università degli Studi della Basilicata*	External examiner
Prof. Andrea M. Zanchettin, *Politecnico di Milano*	External examiner

Istituto Italiano di Tecnologia (*HHCM* lab), and
Department of Informatics, Bioengineering, Robotics and Systems Engineering

Declaration

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

Arturo Laurenzi
January 2020

Acknowledgements

I would like to thank all the *Humanoids and Human Centred Mechatronics* team for contributing to a friendly and competent work environment, and my supervisor Nikos Tsagarakis for giving me the opportunity to work therein. I would also like to thank Dr. Luca Muratore for the fruitful collaboration which lead to the *XBotCore* real-time robotic middleware; Dr. Enrico Mingo Hoffman and Dr. Matteo Parigi Polverini for cooperating on the topic of prioritized force control.

Abstract

Emerging applications will demand robots to deal with a complex environment, which lacks the structure and predictability of the industrial workspace. Complex scenarios will require robot complexity to increase as well, as compared to classical topologies such as fixed-base manipulators, wheeled mobile platforms, tracked vehicles, and their combinations. Legged robots, such as humanoids and quadrupeds, promise to provide platforms which are flexible enough to handle real world scenarios; however, the improved flexibility comes at the cost of way higher control complexity. As a trade-off, hybrid wheeled-legged robots have been proposed, resulting in the mitigation of control complexity whenever the ground surface is suitable for driving. Following this idea, a new hybrid robot called *Centauro* has been developed inside the *Humanoid and Human Centered Mechatronics* lab at *Istituto Italiano di Tecnologia (IIT)*. Centauro is a wheeled-legged quadruped with a humanoid bi-manual upper-body. Differently from other platform of similar concept, Centauro employs customized actuation units, which provide high torque outputs, moderately fast motions, and the possibility to control the exerted torque. Moreover, with more than forty motors moving its limbs, Centauro is a very redundant platform, with the potential to execute many different tasks at the same time. This thesis deals with the design and development of a software architecture, and a control system, tailored to such a robot; both wheeled and legged locomotion strategies have been studied, as well as prioritized, whole-body and interaction controllers exploiting the robot torque control capabilities, and capable to handle the system redundancy. A novel software architecture, made of (i) a real-time robotic middleware, and (ii) a framework for online, prioritized Cartesian controller, forms the basis of the entire work.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

At the time of writing, and roughly sixty years after the very first robotic manipulators were fabricated, despite the vast majority of robotics being clearly focused on industrial applications, robotic platforms are indeed starting to expand from a mostly industrial focus to less structured environments. Such a progress has the potential to enhance the quality of human workers in fields such as agriculture and construction, by relieving humans from performing laborious and repetitive tasks, while at the same time reducing operational costs. Furthermore, and arguably more importantly, robots will be able to handle hazardous scenarios, gradually replacing humans in tasks that expose them to life risk, such as search-and-rescue inside hazardous environments (e.g. collapsed or on fire buildings, high pressure underwater scenarios), fire fighting, response to nuclear accidents, land mine clearance, and others.

As it will become more clear in the course of this chapter, many challenges still need to be overcome in order to deploy an effective robotic responder to the real world; these are related to both control and hardware implementation aspects. Focusing on the hardware design, it is worth noticing that while agile legged platforms do have the advantage of being able to locomote on unstructured surfaces at moderately high speed, they are usually not suitable in terms of physical performance for heavy interaction tasks that might realistically be needed in a disaster scenario, e.g. lifting weights, removing obstacles, turning stiff valves, and others. In order to address this issue, the *Humanoid and Human-Centered Mechatronics (HHCM)* lab at Istituto Italiano di Tecnologia (IIT) has developed the *Centauro* robot (Kashiri et al., 2019), a high-performance quadrupedal robot endowed with a humanoid torso. Centauro is capable of performing heavy manipulation as well as high-impact interaction with the environment, while retaining a compact form factor in the same range of a human being. Moreover, Centauro is fully *torque controlled*; as such, it can behave compliantly when

subjected to external disturbances, a feature that is likely to be essential for loco-manipulation in cluttered, unknown environments.

In addition to hardware-related challenges, many control problems need to be solved. Legged locomotion has the advantage of only requiring discrete available footholds; however, it gives rise to difficult problems such as balancing control and footstep planning. Compliance can be a powerful feature, yet whole-body compliant controllers for legged robots acting in the Cartesian space are of difficult implementation; indeed, such controllers typically struggle to provide satisfactory performance, especially in terms of robustness, mostly due to the high complexity of the system, and its under-actuated nature.

1.1 Thesis overview and objectives

The main goal of this thesis is to develop a software and control framework that will permit the Centauro platform to leverage on its highly redundant mobility and physical capabilities to perform complex whole body loco-manipulation and interaction tasks. This primary goal becomes instantiated with the following objectives that steered the work executed within this thesis:

- Realization of an effective software architecture for complex robotic systems, with the aim to ease the development of control algorithms which can run as close to the hardware as possible, minimizing latencies and jitter.
- Development of Cartesian control methods for whole body manipulation of highly redundant platforms. In this case, the goal is to ease the integration of Cartesian controllers, both programmatically and from a ROS-based distributed control system. Furthermore, such a framework should the user from the need to write and compile custom C++ code for the specific target platform.
- Realization of mobility control schemes enabling effective wheeled and legged locomotion, by fully exploiting the kinematic capabilities of the Centauro platform.
- Development of modules for interaction control, to enable the Centauro robot to operate in Cartesian space in a compliant way, also regulating the forces exchanged with the external environment.

The integration of such core components aims towards a robot that is ready to handle a real disaster scenario, and which addresses the limitations of currently used platforms, as discussed later in this chapter.

The remainder of this chapter is devoted to the introduction of the disaster response robotics field, as it represents the primary source of inspiration for this thesis work (Section 1.2), being a great opportunity for robotics to serve mankind in a deeply significant way. After a brief summary of the related state of the art, our prototype robotic platform is introduced in Section 1.3, i.e. the *Centauro* robot. Finally, Section 1.4 summarizes the main achievements of this thesis in terms of control for the Centauro platform, whereas Section 1.5 lists the main results of this thesis in terms of scientific disseminations.

1.2 Disaster-response robotics

A disaster can be defined as an event, either natural or man-made, which cannot be handled counting only on locally available resources. Disaster scenarios can generate several types of hazard, among which high temperature, radioactivity, risk of explosion, air toxicity, and others, causing high risk to people involved in the management of the disaster itself.

Disasters are commonly described in terms of different phases (Murphy, 2014): a *prevention* phase, which is mostly about preparation and training; the actual *response* phase, and finally a *recovery* phase. Response is clearly the most critical phase; most often, responders need to act as fast as possible, in order to maximize the chance to reach all potential survivors. Even though many recent natural and artificial disasters have highlighted the need for rapidly deployable robotic responders, the application of robotic platforms to this domain is made difficult by the limits of tele-operation technology, which often result in a slow task execution. On the other hand, using a human operator in the loop seems mandatory to handle the off-nominal conditions that characterize the uncertain, unstructured disaster scenarios. The lack of situation awareness caused by delays and poor perception makes it fundamental to keep operators constantly trained, thanks to realistic simulation scenarios and mock up facilities. Given these considerations, it seems that robots can be successfully applied especially to the recovery phase, where slow operation can be tolerated, and invaluable benefits are obtained by relieving human operators from the need to work in the hazardous environments that result from the disaster. Realistically, while disaster robots can not be expected to replace responders in the medium term, they can indeed be used to avoid unnecessary risks.

Despite robots have been used in several disasters, starting from the 2001 World Trade Center collapse, the 2011 Fukushima-Daichii nuclear accident is commonly regarded as a milestone in disaster robotics, both because robots actually managed to perform useful work in the contaminated area that would otherwise be done by humans, and especially because it highlighted important shortcomings in today's robotic technology, thus serving as inspiration

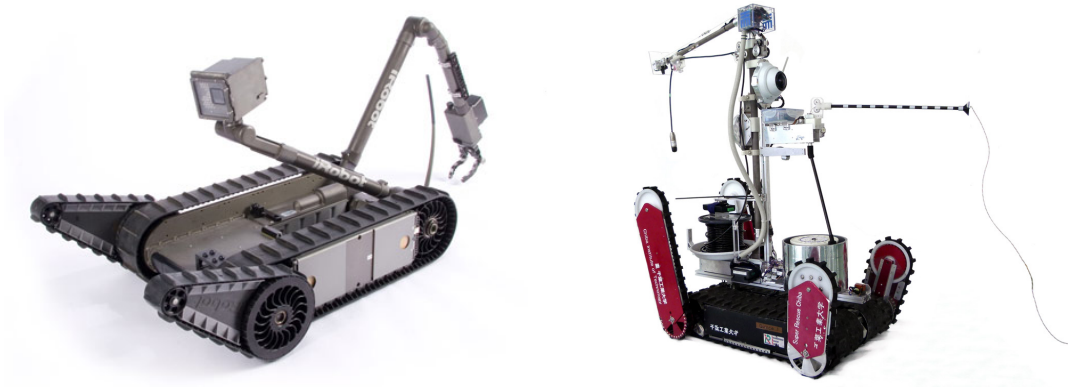
Figure 1.1 Unmanned construction machines removing radioactive debris (taken from Murphy et al. (2016))

and driving force for all robotic community towards the realization of more effective robotic systems. A brief description of the event follows, with the aim to understand how robots managed to serve the human responders, and also why, in some cases, they failed.

1.2.1 The Fukushima-Daiichi disaster

On the 11th of March, 2011, a fourteen meters high *tsunami* hit the Fukushima-Daichii nuclear facility, causing the failure of the back-up diesel generators actuating the reactors cooling water pumps. The reactors had been only recently shut down: without the cooling water flow, the increase in temperature caused several gas explosions and, eventually, the meltdown of three cores. Consequently, radioactive contamination was spread in the area surrounding the facility.

After the event, a four months lasting response phase started, during which robots managed to accomplish tasks related to the removal of radioactive debris through the use of unmanned (teleoperated) construction machines, as well as the aerial reconnaissance of the incident area, which allowed to get an understanding of the situation, also in terms of radiation measurement. Notably, *PackBot* robots from *iRobot* (Figure 1.2(a)) were the first to enter the reactor buildings 1 and 3 (roughly one month after the accident), which proved useful for assessing the level of radiation and air toxicity before the access of human operators. In addition, a similar robot *Quince* (Figure 1.2(b)) was also employed especially during the recovery phase, turning out to be the only platform able to climb the stairs and reach the upper floors. In both cases, an extensive training phase was required prior to the actual operation, also using the building of reactor 5 (which had not been damaged) as a cold test scenario.



(a) PackBot (b) Quince

Figure 1.2 Tracked robots proved to implement the most useful robot morphology in Fukushima, being small and agile, able to handle stairs without resorting to complex perception-based planning.

Despite these success stories, it is generally acknowledged how robotics fell short of expectations during both the response and recovery phase. It was recognized how mobility did represent a major issue for all platforms: some robots failed to climb metal stairs, or to negotiate complex terrains due to the presence of debris. Other failures happened because of the sensory system being obstructed by fog, or due to the robot tether remaining caught in the rubble. In terms of footprint and weight, successful platforms were either very small and light but with very limited manipulation skills (especially in terms of payload capacity), as in the case of the PackBot robot, or physically strong but very large and heavy (e.g. construction machines). It was therefore clear that to operate within infrastructures that had been designed for humans, a robot should possess a rich repertoire of human-like skills, such as agility, physical power, locomotion and manipulation capabilities, as well as the possibility to interact with a harsh environment.

A significant step towards addressing these shortcomings has been taken with the *CENTAURO* European Project[1], which ended November 2018 after more than three years of development activity. Beside proposing to advance the state of the art in fields such as semi-autonomous tele-manipulation and locomotion, the project produced a novel robotic platform called Centauro, which combines state of art compliant actuation with a hybrid wheeled-legged quadrupedal mobility concept. The multi-legged topology aims to address the limitations of wheeled or tracked mobile robots in rough terrains, since they can deal with isolated footholds, plus the legs configuration and the robot posture can be controlled and adapted for maintaining the balance according to the terrain variations. At the same time,

[1]The CENTAURO project website: https://centauro-project.eu/

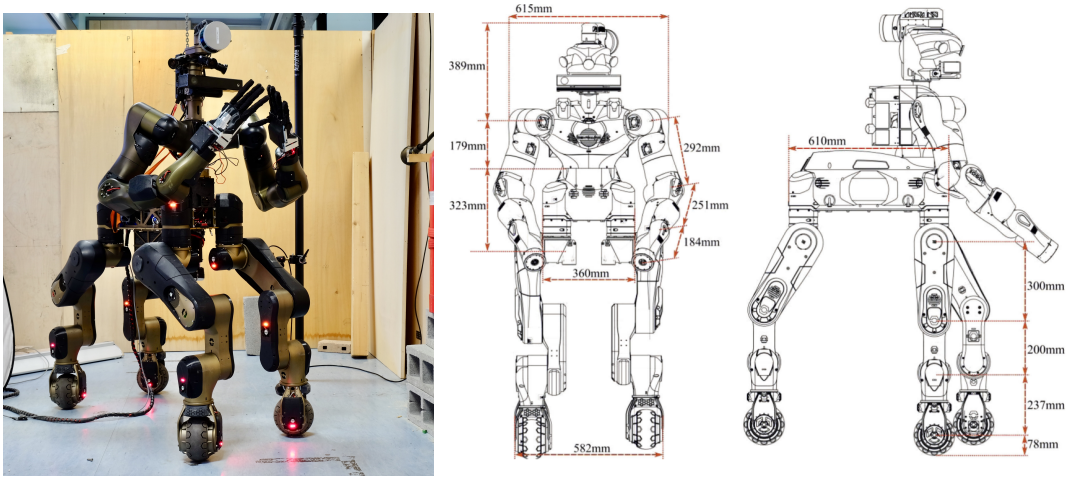


Figure 1.3 The Centauro robot, a hybrid wheeled-legged quadruped with a humanoid upper-body.

wheeled motion can be conveniently used to move the robot on smooth surfaces in a more energy efficient way. More details about the Centauro platform, which represents the main target of the research proposed in this thesis, are given in the following section.

1.3 The Centauro robot

The Centauro robot (Kashiri et al., 2019) is an hybrid wheeled-legged quadruped, which is also equipped with a humanoid, bi-manual upper body. Each leg is driven by four actuators, allowing the tip to be positioned in 3D, while also providing an additional degree of freedom for partial orientation control. It implements a *spider-like* kinematics, with a yaw as the most proximal joint allowing to reduce the static (gravitational) load. The leg terminates with a wheel which is controlled by two motors: the first one provides the steering motion about a vertical axis, whereas the second one is responsible for the rolling of the wheel about its spinning axis. Overall, the chosen leg kinematics allows to control the position of the wheel, while independently keeping the steering axis perpendicular to the rolling surface.

The Centauro robot is designed not only for flexible locomotion over difficult terrain, but also for heavy manipulation. Such capability is achieved thanks to an upper body comprising a torso, which can rotate about the yaw axis, on which two arms are mounted. Each arm is driven by seven motors in order to achieve a level of kinematic redundancy with respect to a position and orientation placement task.

A picture of the Centauro robot is shown in Figure 1.3, together with a CAD-generated image reporting its dimensions. From these figures it is clear that Centauro, weighing about

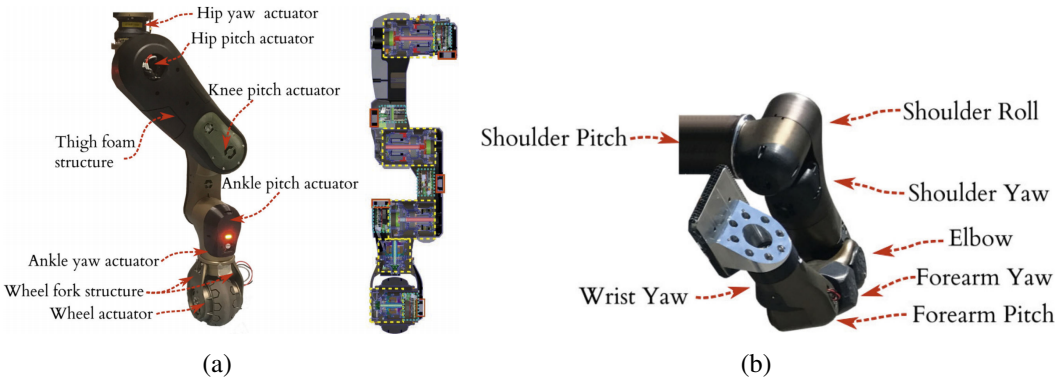


Figure 1.4 Kinematic structure of the Centauro robot leg and arm.

Figure 1.5 Torque controlled integrated actuators used inside HHCM robots. Different sizes, when combined with different transmission ratios, allow to select a trade-off between weight, speed and torque.

90 kg, has comparable footprint with respect to a human. This notwithstanding, it delivers high manipulation performance, each arm being able to manipulate over 10 kg over its full workspace. Furthermore, the legs are strong enough to sustain an additional payload on the trunk of roughly 70% the overall robot weight. These results have been achieved thanks to a custom-made actuation system: more specifically, a family of five actuators covering a wide range of rated torque and speed has been designed and developed, as depicted in Figure 1.5. All such actuators incorporate a sensor measuring the torque exerted to the link side, thus enabling precise torque control. The implementation of torque sensing relies either on *strain gauges* or on an encoder measuring the deflection of a torsional bar located at the harmonic gearbox output. In the second case, the motor behaves as *Series Elastic Actuator (SEA)*, with the passive compliance providing additional protection of the transmission against impacts.

A strong and flexible hardware platform is not useful without its control system. The present work of thesis is related for the most part to the development of such a framework, starting from the middleware architecture, to wheeled and legged locomotion controllers, passing through whole-body Cartesian control. A detailed summary of such contributions is given in the next section.

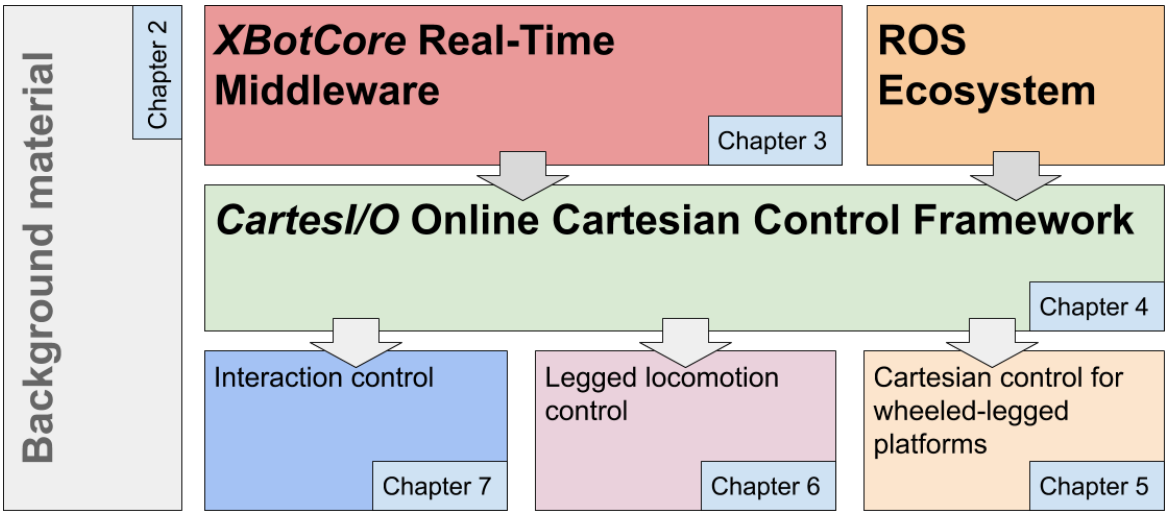


Figure 1.6 Thesis structure and dependencies between modules, indicated by arrows.

1.4 Contribution

As it was illustrated in the previous section, the Centauro robot is a rather peculiar platform combining powerful manipulation capabilities with a reliable quadrupedal hybrid wheeled-legged locomotion skills. Moreover, it leverages on an advanced proprioceptive actuation system to regulate the force exchanged with the environment. Therefore, the main contributions of these thesis are driven and related to these particular features of the Centauro robot. Figure 1.6 illustrates the main contributions of this thesis as logical blocks, with arrows representing dependencies between them.

The first topic of this work of thesis is related to the development of a **control software architecture** for complex robotic systems, such as the Centauro platform. Previous robots that were designed inside the HHCM lab (e.g. *Coman* and *Walkman* (Tsagarakis et al., 2017a)) were entirely powered by the YARP middleware (Natale et al., 2016), which would cover the whole software architecture stack, ranging from the *hardware abstraction layer (HAL)*, to the inter-process communication (IPC) between high-level, low rate processes. While such an architecture proved to be effective, it could not handle deterministic execution of high-rate high-priority tasks. To address this shortcoming, a novel real-time architecture has been developed, i.e. the *XBotCore* middleware. This software framework provides tools enabling the *real-time (RT)* control of a complex robot, from a flexible hardware abstraction layer to *application programming interfaces (API)*, which allow the user/developer to write generic robot-independent and framework-agnostic control code, that can then be executed inside a RT loop. Real-time guarantees are handled by the underlying operating system (*Xenomai 3.0*

or *Rt-Preempt*) as well as by careful coding practices which eliminate all RT-unsafe operation from the high-priority threads. Such operations, which are necessary in order to integrate with the rest of the distributed control system, are offloaded to low-priority companion threads through appropriate (possibly lock-free) synchronization primitives. Thanks to this effort, the XBotCore framework nicely integrates with the ROS ecosystem (Quigley et al., 2009b); ROS-related primitives (such as topics and services) can be used from the RT domain thanks to specific classes hiding all complexity related to the use of the aforementioned non-RT companion threads. Finally, thanks to our generic API, control code can be moved any time to a standard ROS node, whenever it turns out not to need high-priority execution. This work has been carried out jointly with Dr. Luca Muratore; beside the general architecture design, which is a product of team work, the focus of the author has been especially devoted to the high-level API design, and ROS integration.

In parallel to the development of XBotCore, this thesis contributed with a ROS-based software framework for on-line, **real-time** capable **Cartesian control**, which was named *CartesI/O*. This piece of software extends a previous work from the HHCM lab *OpenSoT* (Hoffman et al., 2017), which is a C++ library implementing a stack-of-tasks strategy (Escande et al., 2014a) to the solution of prioritized quadratic programs, with special application to the Cartesian control (velocity-level, acceleration-level, torque, . . .) of complex robots. With CartesI/O, a uniform interface has been given to Cartesian controllers, as well as provide them with an auto-generated ROS-based API allowing them to interact with the distributed control system through topics, services, and action. Furthermore, OpenSoT-based controllers can be automatically set up from textual configuration files, as opposed to manually writing and compiling C++ code. The resulting library can be thought of as an analogue of the venerable *MoveIt!* motion planning library, that is instead dedicated to prioritized online control. Details on the design and validation of both the XBotCore and CartesI/O architectures are presented in Chapter 3 and 4.

The second topic of this work addresses the **legged locomotion** capabilities of the Centauro robot. Such a topic has been subject of several studies in the existing literature, also considering the strong similarity with the gait generation problem for bipedal platforms. It can be observed how the full problem is usefully split into two separate stages, i.e. a *footstep planning* and a *center of mass (CoM) planning* stage. Having done so, both problems can be turned into linearly constrained QPs, that can be solved exactly by off-the-shelf libraries. However, such a separation neglects the coupling between footstep planning and center of mass motion; this can ultimately lead to excessive CoM motion, and to instability as well. Treating both stages together inside a single optimization problem permits, in principle, to

address this effect, at the price of making the whole problem non-convex. As such, the solution might be only a local minimum with potentially high associated cost. It was therefore proposed to apply a convex approximation to the overall gait generation problem, which can be solved exactly and in less time. The corresponding methodology is described in Chapter 6.

As a third contribution of this thesis, a study has been done on the topic of whole-body **Cartesian control in the context of articulated wheeled robots**. More in detail, it was observed how given the full wheeled mobility of the robot as allowed by its four independently steerable wheels, the choice of some local frame (in addition to the global world) is required in order to express tasks that are naturally defined in a robot-centric fashion. Such study highlighted the fact that trivially selecting such a frame as the robot trunk leads to sub-optimal results in terms of motion capabilities; as main contribution, we therefore propose a comparative analysis among three different choices of local frame, and demonstrate that in order to retain all advantages from the whole-body control domain, the kinematic model of the robot must be augmented with an additional virtual frame, which provides for an useful choice of local frame. The outcome of our method is a Cartesian controller with the ability to freely mix local adjustments with global control inside a unified whole-body framework. Theory and validation of this approach are presented in Chapter 5.

The last topic treated in this thesis is in the field of **whole-body torque control** of floating base systems, where this thesis contributes with an extension of a previous work which was proposing a formulation for prioritized Cartesian impedance controller (Hoffman et al., 2018). Such family of controllers is well-known, especially concerning the control of modern torque-enabled manipulators. Extending the Cartesian impedance principle (or other torque-based control laws) to floating systems requires handling the *contact forces* which ultimately permit the robot motion, which is usually done by regarding such forces as additional control inputs that are either pre-optimized based on a reduced model (e.g. the well-known *centroidal dynamics*), or jointly optimized with the joint acceleration vector. In this context, the main contribution was a novel pipeline which first assumes the robot to be a fully-actuated fixed base system (i.e. with an actuated base), whereas a later stage converts the obtained torques to the actual under-actuated model, by performing a *post-optimization* of contact forces. Theory and validation of this technique are presented in Chapter 7, with application to a heavy object pushing task exploiting contacts with the environment.

In order to provide the necessary **background knowledge**, Chapter 2 provides material on topics such as the modeling of floating base systems, inverse kinematics, and dynamics, in an effort to make this work of thesis self-contained. Finally, Chapter 8 concludes the

work with a summary of achieved results, comments on dissemination and exploitation, and possible future directions.

1.5 Contributed papers

The outcome of the research work done in this thesis has been published or accepted in several relevant scientific conferences or journals. A summary of the most relevant publications follows.

- Architectural article on the XBotCore framework (see Chapter 3):

  Muratore, L., Laurenzi, A., Hoffman, E. M., Rocchi, A., Caldwell, D. G., and Tsagarakis, N. G. (2017b). Xbotcore: A real-time cross-robot software platform. In *Robotic Computing (IRC), IEEE International Conference on*, pages 77–80. IEEE

- Architectural paper on the CartesI/O framework (see Chapter 4):

  Laurenzi, A., Mingo Hoffman, E., Muratore, L., and Tsagarakis, N. G. (2019c). CartesI/O: A ROS Based Real-Time Capable Cartesian Control Framework. In *IEEE International Conference on Robotics and Automation (ICRA)*

- Integration work involving XBotCore, CartesI/O, and a perception pipeline:

  Laurenzi, A., Kanoulas, D., Hoffman, E. M., Muratore, L., and Tsagarakis, N. (2019a). Whole-body stabilization for visual-based box lifting with the COMAN+ robot. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 445–446. IEEE

- Article on the control of wheeled-legged platforms (see Chapter 5)

  Laurenzi, A., Mingo Hoffman, E., Parigi Polverini, M., and Tsagarakis, N. G. (2020). An Augmented Kinematic Model for the Cartesian Control of the Robot CENTAURO. *IEEE Robotics and Automation Letters*. Accepted, to appear

- Article on walking pattern generation for quadrupeds (see Chapter 6):

  Laurenzi, A., Hoffman, E. M., and Tsagarakis, N. G. (2018b). Quadrupedal walking motion and footstep placement through linear model predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2267–2273. IEEE

- Work on prioritized force control, with application to Cartesian impedance control (see Chapter 7):

Mingo Hoffman, E., Laurenzi, A., Tsagarakis, N. G., and Caldwell, D. G. (2018). Multi-priority cartesian impedance control based on quadratic programming optimization. In *IEEE International Conference on Robotics and Automation, ICRA*

- Article on balancing control for legged robots based on post-optimization of contact forces (see Chapter 7):

Laurenzi, A., Hoffman, E. M., Polverini, M. P., and Tsagarakis, N. G. (2018a). Balancing control through post-optimization of contact forces. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 320–326. IEEE

Chapter 2

Background on Cartesian Control

This appendix presents the notation that is used throughout this thesis whenever Cartesian control is concerned, as well as background material on the topics of inverse kinematics and dynamics aware control.

2.1 Reference frames representation

Let us consider a Cartesian world frame W and an additional Cartesian frame A, whose origin coincides with W. The orientation of A w.r.t. W can then be expressed by specifying the directions of A's coordinated axes w.r.t. W frame, i.e. ${}^{W}\boldsymbol{i}_A$,${}^{W}\boldsymbol{j}_A$, and ${}^{W}\boldsymbol{k}_A$. These can be conveniently assembled into a rotation matrix as follows,

$$ {}^{W}\boldsymbol{R}_A = \begin{bmatrix} {}^{W}\boldsymbol{i}_A & {}^{W}\boldsymbol{j}_A & {}^{W}\boldsymbol{k}_A \end{bmatrix}, \tag{2.1} $$

which maps vectors expressed in A coordinates into vectors in W coordinates, i.e.

$$ {}^{W}\boldsymbol{p} = {}^{W}\boldsymbol{R}_A \, {}^{A}\boldsymbol{p}. \tag{2.2} $$

Because coordinated axes are mutually orthogonal, have unit length, and form a right-handed triplet (i.e. $\boldsymbol{i} \times \boldsymbol{j} = \boldsymbol{k}$), a rotation matrix is actually an orthogonal matrix ($\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I}$) with determinant equal to $\det(\boldsymbol{R}) = 1$. Therefore, a rotation matrix lives in a manifold, embedded in $\mathbb{R}^{3\times 3}$, that is called *special orthogonal group*, denoted by $\mathrm{SO}(3)$. Notice that, because of the aforementioned constraints, the SO(3) manifold has only dimension equal to three, out of the nine numbers that make up a general three-by-three matrix.

If origins do not coincide, then (2.2) must take the offset into account as follows:

$$ {}^{W}\boldsymbol{p} = {}^{W}\boldsymbol{R}_A \, {}^{A}\boldsymbol{p} + {}^{W}\boldsymbol{o}_A, \tag{2.3} $$

where ${}^{W}\boldsymbol{o}_A \in \mathbb{R}^3$ is the origin of A in W coordinates. The rotation matrix and the offset vector are conveniently packed into a four-by-four matrix (homogeneous matrix) ${}^{W}\boldsymbol{T}_A$

$$ {}^{W}\boldsymbol{T}_A = \begin{bmatrix} {}^{W}\boldsymbol{R}_A & {}^{W}\boldsymbol{o}_A \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}, \tag{2.4} $$

which is an element of the *special Euclidean group* $\mathrm{SE}(3)$.

2.1.1 Motion of rigid bodies

For robot control purposes, it is of paramount importance to understand and describe how rigid bodies move in space. Clearly, the position and motion of a rigid body can be identified with that of a Cartesian frame attached to it. Therefore, the rotation matrix derivative $\dot{\boldsymbol{R}}$ can be used which, together with the origin linear velocity $\boldsymbol{v}_O$, forms an element of the tangent space to SE(3). However, such a description is redundant, since we are using nine numbers to describe an element which is tangent to SO(3), whose dimension is three. Indeed, it turns out that the rotation matrix derivative at the generic point $\boldsymbol{R} \in \mathrm{SO}(3)$ is given by

$$ \dot{\boldsymbol{R}} = \boldsymbol{\Omega}\boldsymbol{R}, \tag{2.5} $$

where $\boldsymbol{\Omega}$ is the three-by-three skew-symmetric matrix defining the cross product by the rigid body angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$, expressed in world coordinates, i.e. $\boldsymbol{\Omega}\boldsymbol{x} = \boldsymbol{\omega} \times \boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^3$. To conclude, the motion of a rigid body is specified by the pair $(\boldsymbol{v}_O, \boldsymbol{\omega})$, which is conveniently assembled into a six-dimensional vector called velocity twist $\boldsymbol{v} \in \mathbb{R}^6$:

$$ \boldsymbol{v} = \begin{bmatrix} \boldsymbol{v}_O \\ \boldsymbol{\omega} \end{bmatrix}. \tag{2.6} $$

It is important to notice how, despite the fact that the angular velocity describes the rate of change of an element of SO(3), it is not equal to the derivative of any other quantity (non-holonomy of the angular velocity), and its time integral does not carry any physical meaning.

2.1.2 Spatial algebra

Velocity twists provide a meaningful and intuitive description of the motion of a rigid body; yet, it turns out that they do not represent an optimal choice to develop (and program) kinematics and dynamics algorithms. Intuitively, the shortcoming of a twist is that it depends on the choice of a *reference point*, i.e. the rigid body origin O; when dealing with systems that are made of multiple bodies (e.g. robotic systems), each body i will have a different origin position O_i, which makes computations harder and more verbose by requiring several mathematical steps just to deal with reference point offsets. This can already be observed when computing a quantity as simple as the relative twist between two bodies, as follows. Let $\boldsymbol{T}_A \in \mathrm{SE}(3)$ be the pose of body A w.r.t. the world, and $\boldsymbol{T}_B \in \mathrm{SE}(3)$ be the pose of body B; moreover, let $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ denote the velocity twists of the two bodies w.r.t. the world. The relative velocity twist cannot be computed by simply subtracting $\boldsymbol{v}_A$ from $\boldsymbol{v}_B$, as the two linear velocities contained in each twist do not refer to the same point in space. Hence, one must first shift the reference point for the twist of A to coincide with that of B, and only then the subtraction can be carried out.

The motion of a rigid body can be equivalently be described by a *spatial vector* (or 6-D vector), denoted with a *hat* symbol (e.g. $\hat{\boldsymbol{v}}$) as described e.g. in Featherstone (2010). Intuitively, spatial vectors describe the rigid body motion by specifying its angular velocity, plus the linear velocity of a point which is attached to the rigid body, and always coincides (instantaneously) with the world origin. By doing so, spatial vectors manage to describe the motion in a self-contained way, whereas twists always need the specification of the corresponding reference point in order to make physical sense. Moreover, spatial vectors can be combined in a simpler way, e.g. a relative spatial vector is just obtained by subtraction:

$$\hat{\boldsymbol{v}}_{\mathrm{rel}} = \hat{\boldsymbol{v}}_B - \hat{\boldsymbol{v}}_A. \tag{2.7}$$

However, it can be argued that the spatial description of motions (and forces) is less intuitive than the twist (and wrench) based representations. For example, a wheel rolling on a plane with constant velocity, from the perspective of spatial vectors, is constantly accelerating upwards (see Figure 2.1)! For this reason, in this work of thesis, we have decided to work with twist vectors whenever it is necessary to represent the motion of a rigid body.

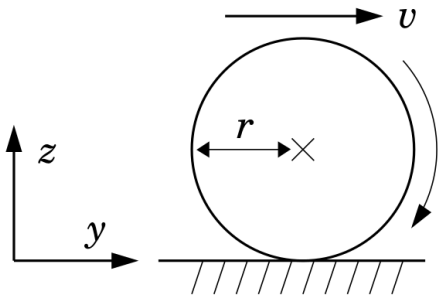


Figure 2.1 Wheel rolling on the y axis without slipping. Its spatial velocity is $\hat{\boldsymbol{v}} = \begin{bmatrix} 0 & 0 & y\frac{v}{r} & \frac{v}{r} & 0 & 0 \end{bmatrix}$, and its spatial acceleration is $\hat{\boldsymbol{a}} = \begin{bmatrix} 0 & 0 & \dot{y}\frac{v}{r} & 0 & 0 & 0 \end{bmatrix}$, which is constant and directed upwards. This figure was adapted from http://royfeatherstone.org/spatial/index.html.

2.2 Robot kinematics

A robot can be generally regarded as a collection of rigid bodies (called *links*), all of which are connected together in a tree-like fashion through so-called *joints* (see Figure 2.2). Each joint enables the relative motion of its *child link* with respect to *parent link*, along some *motion subspace* which depends on the type of joint. For instance, 1-dof prismatic joints only allow for the relative translation about some defined axis, whereas floating joints enable the full 6-dof relative motion between the parent and child links. Let us immediately observe that, while such a choice leaves parallel mechanisms out, such topologies can be re-introduced by imposing *holonomic* constraints between the robot links.

The kinematic state of a given joint is described by the corresponding *joint variable*, denoted by the symbol $\boldsymbol{q}_i \in \mathscr{C}_i$, where i is an index, and $\mathscr{C}_i$ is the joint configuration space (in general, a manifold in $\mathbb{R}^{n_{q,i}}$). Given the type of joint and the value of the joint variable, it is possible to define the joint transform $\boldsymbol{T}_{Ji}(\boldsymbol{q}_i) \in \mathrm{SE}(3)$ between the joint *predecessor frame*, which is attached to the parent link, and the *successor frame*, which coincides with the child link frame (body frame), as shown in Figure 2.3. In order to complete our robot kinematic state description, we need to specify the pose of each joint predecessor frame with respect to its parent link, which we call *tree transform* and denote it with the symbol $\boldsymbol{T}_{Ti} \in \mathrm{SE}(3)$. Notice how, for a given robot, tree transforms are constant and independent of the value of joint variables. Once joint variables are known, the pose of all links w.r.t. the world frame can be propagated from the kinematic tree root by suitably multiplying tree transforms and joint transforms, yielding the so-called *forward kinematics* function $\boldsymbol{f}_{\mathrm{fk}} : \mathscr{C} \to \mathrm{SE}(3)$ mapping the

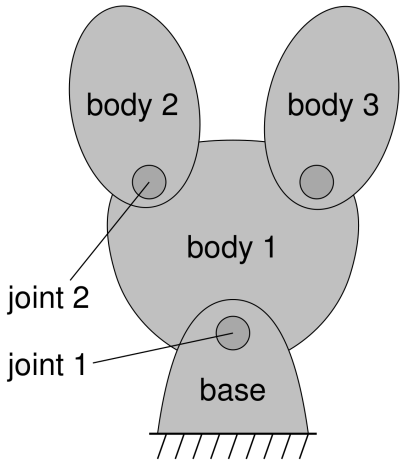


Figure 2.2 Example of articulated body, i.e. a collections of rigid body (links) connected through joints in order to form a kinematic tree. This figure was adapted from http://royfeatherstone.org/spatial/index.html.

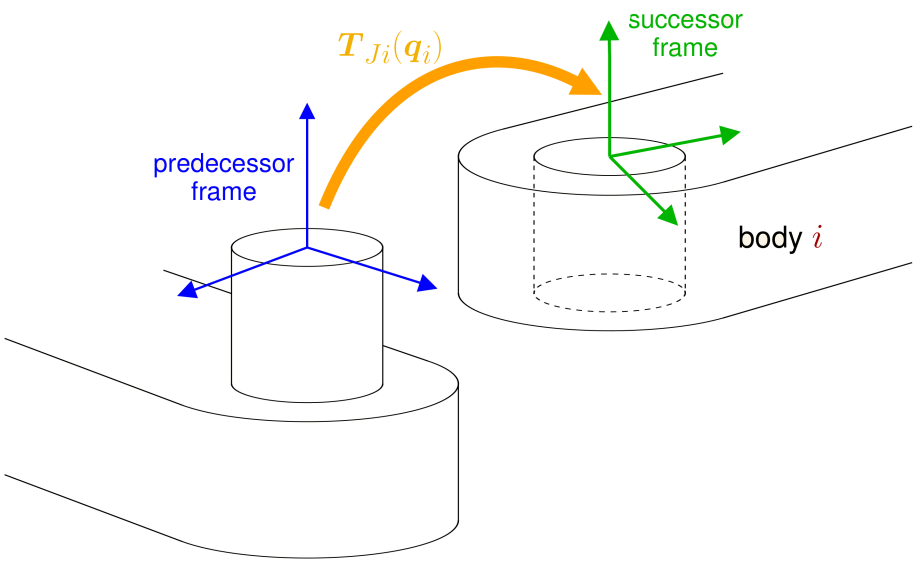


Figure 2.3 Pictorial representation of a 1-dof revolute joint connecting two links. Its joint transform specifies the offset between the successor frame (i.e. the child link frame), and the predecessor frame (which is attached to the parent link). This figure was adapted from http://royfeatherstone.org/spatial/index.html.

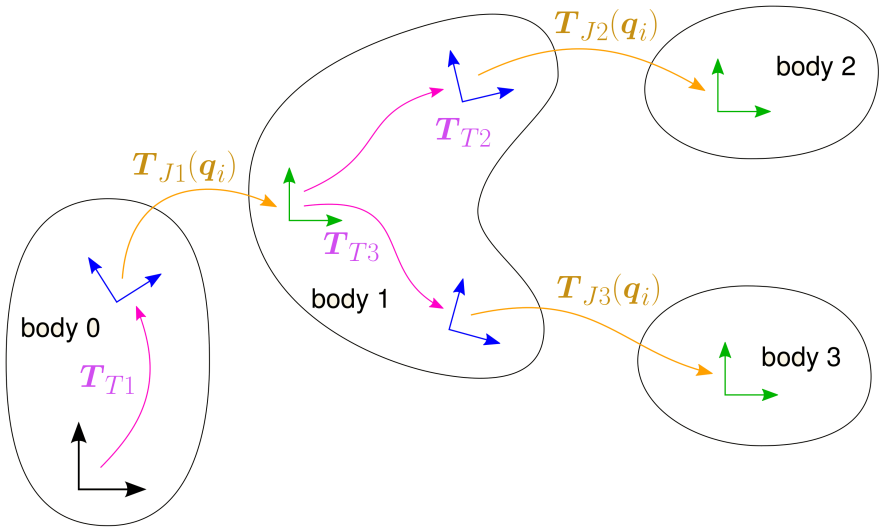


Figure 2.4 Kinematic state description for a simple robot, involving joint transforms and tree transforms. The forward kinematics function is evaluated by composing such quantities, starting from the tree root and following the arrows. This figure was adapted from http://royfeatherstone.org/spatial/index.html.

robot configuration $\boldsymbol{q}$ to the Cartesian pose of the i-th link T_i w.r.t. the world frame:

$$\boldsymbol{f}_{\text{fk}}(\boldsymbol{q}) = \boldsymbol{T}_i(\boldsymbol{q}), \tag{2.8}$$

with $\boldsymbol{q}$ being obtained by collecting all joint variables into a tuple. As it was previously mentioned, at the velocity level, each joint allows for a relative *motion* between the relevant links. The motion state of the joint is parametrized by the joint velocity variable $\boldsymbol{v}_i \in \mathbb{R}^{n_{v,i}}$, where $n_{v,i}$ is the dimension of the joint motion subspace. In order to convey the information about *which* motions are allowed by a specific joints, we can define a suitable matrix $\boldsymbol{S} \in \mathbb{R}^{6\times n_{v,i}}$ mapping joint velocities $\boldsymbol{v}_i$ into allowed spatial velocities of the child link relative to the parent link $\hat{\boldsymbol{v}}_{\text{rel},i}$:

$$\hat{\boldsymbol{v}}_{\text{rel},i} = \boldsymbol{S}\boldsymbol{v}_i; \tag{2.9}$$

relative accelerations can be computed as well by differentiating (2.9), taking into account that the matrix $\boldsymbol{S}$ can vary in time as well. Proper propagation of (2.9) allows to compute the spatial velocity (or velocity twist) of all robot links, giving rise to the *differential kinematics* function, which is usually given in the form of a *Jacobian matrix* $\boldsymbol{J} \in \mathbb{R}^{6\times n_v}$ mapping the

joint velocities into velocity twists:

$$\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{\nu}, \tag{2.10}$$

with $\boldsymbol{\nu}$ being obtained by stacking all joint velocities into a column vector. It is worth to notice that the first three rows of $\boldsymbol{J}$ are indeed the analytic Jacobian of the forward kinematics function (2.8) translation part. Clearly, the same does not apply to the rotation part, since a rigid body angular velocity does not represent the derivative of any vector quantity.

Remark. *In many cases, the joint transform $\boldsymbol{T}_{Ji}(\boldsymbol{q}_i)$ can be parametrized by a Euclidean vector, i.e. $\mathscr{C}_i = \mathbb{R}^{n_{q,i}}$ (e.g. prismatic joints, revolute joints with joint limits, and others) without loss of information. In all such cases, the joint velocity is simply the derivative of the joint variable, i.e. $\boldsymbol{\nu}_i = \dot{\boldsymbol{q}}_i$. If this is the case for all robot joints, as for most industrial manipulators, then $\boldsymbol{\nu} = \dot{\boldsymbol{q}}$ and the differential kinematics equation reads the more familiar expression $\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$. Moreover, the robot configuration $\boldsymbol{q}$ is a Euclidean vector itself, and it is obtained by stacking all joint variables, and $n_q = n_\nu = n$.*

Example (1-dof revolute joint)

A 1-dof revolute joint allows for the relative rotation of its parent and child links about a single axis. Without loss of generality, such an axis can be identified with the z-axis (other choices are equivalent to rotating its tree transform $\boldsymbol{T}_{Ti}$). In this case, the following propositions hold:

- the joint variable q_i is scalar ($n_{q,i} = 1$);
- the joint transform has the following form:

$$\boldsymbol{T}_{Ji}(q_i) = \begin{bmatrix} c_i & s_i & 0 & 0 \\ -s_i & c_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.11}$$

  where $s_i = \sin q_i$ and $c_i = \cos q_i$;
- the joint velocity is $\dot{q}_i$ ($n_{\nu,i} = 1$);
- the motion subspace is defined by

$$\boldsymbol{S} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T. \tag{2.12}$$

Example (3-dof spherical joint)

A 3-dof spherical joint allows an arbitrary relative rotation between its parent and child links. Such a rotation can be defined with respect to the predecessor frame origin (different choices are equivalent to translating its tree transform $\boldsymbol{T}_{Ti}$). For such a joint, the following propositions hold:

- the joint variable $\boldsymbol{q}_i$ can be taken as a unit quaternion, i.e.

$$\mathscr{C}_i = \{\boldsymbol{q} \in \mathbb{R}^4 \text{ s.t. } \|\boldsymbol{q}\| = 1\}, \tag{2.13}$$

  and $n_{q,i} = 4$;

- the joint transform has the following form:

$$\boldsymbol{T}_{Ji}(\boldsymbol{q}_i) = \begin{bmatrix} \boldsymbol{R}(\boldsymbol{q}_i) & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}, \tag{2.14}$$

  where $\boldsymbol{R}(\boldsymbol{q}_i)$ is the rotation matrix corresponding to the quaternion $\boldsymbol{q}_i$;

- the joint velocity is $\boldsymbol{\omega}_i$, i.e. the angular velocity between the two bodies ($n_{v,i} = 3$);

- integration of a constant motion is done through the *Rodrigues' formula*;

- the motion subspace is defined by

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{0}_{3\times3} \\ \boldsymbol{I}_{3\times3} \end{bmatrix}. \tag{2.15}$$

Notice how, because the spherical joint must allow a relative motion in SO(3), its joint variable inherently lives on a manifold (unit sphere in four dimensions). Therefore, its joint velocity is not equal to the derivative of the joint variable. It is also possible to locally represent SO(3) with a triplet of Euler angles $\boldsymbol{q}_i \in \mathbb{R}^3$, describing three successive rotations about different local axes. With such a choice, the 3-dof spherical joint can be effectively replaced by a series of three 1-dof revolute joints. However, for each choice of Euler angles (e.g. ZYZ, XYZ, ...) there exist a singularity value of the joint variable for which the motion subspace matrix becomes rank deficient:

$$\text{rank}(\boldsymbol{S}) < 3, \tag{2.16}$$

and it is no longer possible to represent an arbitrary angular velocity between parent and child body through the joint velocity variable. Moreover, close to such a singular orientation, small angular velocities might require high joint velocities, likely causing numerical instabilities if no special care is taken.

2.2.1 Floating base robots

In opposition to fixed base industrial manipulators, legged robots can be categorized as *floating-base* systems, because there is no link of the robot which is rigidly fixed to the world frame. In such a case, the kinematic state of such robot must include, in addition to the position of actuated joints, the Cartesian pose of one of its links (usually referred to as the floating base) with respect to the global frame. This can be done by connecting the floating base to the world frame through prismatic joints about x, y, and z directions, as well as a spherical joint to represent the rotation part. In the present work of thesis, with the aim to simplify the notation and reduce code complexity, we decided to represent the floating base orientation as a XYZ Euler angle triplet, for which the singular orientation corresponds to the floating base x axis pointing downwards. If the robot base link is selected as the floating base, with its x-axis pointing forward, this corresponds to a pose which is physically outside the robot workspace by a considerable margin. Hence, we shall make the following simplifications:

- the robot configuration $\boldsymbol{q}$ is a Euclidean vector belonging to $\mathbb{R}^n$;
- the robot velocity vector is the derivative of $\boldsymbol{q}$, i.e. $\boldsymbol{\nu} = \dot{\boldsymbol{q}}$;
- the symbol n denotes the dimension of both the configuration space and the velocity space, i.e. $n_q = n_\nu = n$.

2.3 Inverse kinematics

The inverse kinematics (IK) problem aims at finding a joint configuration vector $\boldsymbol{q}^* \in \mathbb{R}^n$ which achieves a desired pose $\boldsymbol{T}_{\text{des}}$ for some target frame which is attached to the robot. In the non-redundant scenario where the number of degrees of freedom equals the dimensionality of the Cartesian task at hand (e.g. $n = 6$ and the considered task involves the full pose of some link), one might attempt to solve the non-linear system of equations which matches the

forward kinematics function and the desired pose, i.e.

$$\boldsymbol{T}(\boldsymbol{q}) = \boldsymbol{T}_{\text{des}}; \tag{2.17}$$

indeed, for simple manipulators the solution exists in closed form. Notice that multiple isolated solutions usually exist even if there is no redundancy in the system. However, in the general case (2.17) is not applicable because (i) there is no analytic solution, (ii) the solution must satisfy additional constraints (e.g. joint limits) and (iii) the robotic system is redundant w.r.t. the given task, i.e. a continuum of solutions might exist. In such a case, an intuitive way to formulate the IK problem is in terms of an optimization problem. To this aim, let us initially focus on a pure position task, i.e. the position of the target frame origin $\boldsymbol{p}(\boldsymbol{q})$ must coincide (or be as close as possible) to a given desired value $\boldsymbol{p}_{\text{des}}$; this can be obtained by solving the problem

$$\begin{aligned} &\min_{\boldsymbol{q}} \|\boldsymbol{p}(\boldsymbol{q}) - \boldsymbol{p}_{\text{des}}\|^2 + \varepsilon\|\boldsymbol{q} - \boldsymbol{q}_0\|^2 \\ &\text{subject to} \quad \boldsymbol{q}_{\min} \leq \boldsymbol{q} \leq \boldsymbol{q}_{\max} \end{aligned}, \tag{2.18}$$

where a small $\varepsilon > 0$ acts as *postural task*, aiming at finding a specific solution which is as close as possible to a reference posture $\boldsymbol{q}_0$. Such an approach can be easily extended to a multi-task scenario, by adding a least-squares term in the cost function for every defined task. However, in order to keep the notation as general as possible, it is beneficial to define an abstract task $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{q})$, where $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is a non-linear mapping representing any task, or aggregation of tasks, for which the desired value $\boldsymbol{x}_{\text{des}} \in \mathbb{R}^m$ is available.

Remark. *Notice how to define the task function $\boldsymbol{f}$ co-domain to be the m-dimensional Euclidean space represents an abuse of notation, since the task function (or some of its components) might actually map into a manifold (e.g. SO(3)). However, it is the author's opinion that such a choice is beneficial in simplifying the notation and improving general readability.*

2.3.1 Orientation tasks

Let us now extend the general approach of (2.18) to an orientation task, whose objective is to match a frame orientation $\boldsymbol{R}(\boldsymbol{q}) \in \text{SO}(3)$ to a given desired value $\boldsymbol{R}_{\text{des}}$. To this aim, we need to define a suitable error function on the space of rotation matrices, which replaces the position error $\boldsymbol{p}(\boldsymbol{q}) - \boldsymbol{p}_{\text{des}}$. This can be obtained in at least two different ways:

- if orientations are given in terms of rotation matrices, one can define the error as

$$\boldsymbol{e}_O = \text{skew}(\boldsymbol{R}(\boldsymbol{q})\boldsymbol{R}_{\text{des}}^T)^\vee, \tag{2.19}$$

  where (i) the *skew* operator projects a matrix into the skew symmetric subspace, i.e. $\text{skew}(R) = \frac{R-R^T}{2}$, and (ii) $S^\vee$ extracts the underlying 3-D vector from a skew symmetric matrix S.

- If orientations are given in terms of quaternions, the error can be defined as in Nakanishi et al. (2008), i.e.

$$\boldsymbol{e}_O = \boldsymbol{\varepsilon}_{\text{des}} \times \boldsymbol{\varepsilon}(\boldsymbol{q}) + \eta_{\text{des}}\boldsymbol{\varepsilon}(\boldsymbol{q}) - \eta(\boldsymbol{q})\boldsymbol{\varepsilon}_{\text{des}}, \tag{2.20}$$

  where $\boldsymbol{\varepsilon}$ and η denote the quaternion vector part (imaginary part) and scalar part (real part), respectively.

  Remark. *Because one quaternion and its opposite represent the same rigid body orientation, we need to define the orientation error in* (2.20) *using two quaternions that belong to the same half of the unit hypersphere. This requires the dot product of the desired and actual quaternion to be greater-equal than zero, i.e.*

$$\boldsymbol{\varepsilon}_{des} \cdot \boldsymbol{\varepsilon}(\boldsymbol{q}) + \eta_{des}\eta(\boldsymbol{q}) \geq 0; \tag{2.21}$$

  otherwise, the sign of desired value must be changed.

A more general form of (2.18) can be then written as

$$\begin{aligned} &\min_{\boldsymbol{q}} \|\boldsymbol{x}(\boldsymbol{q}) - \boldsymbol{x}_{\text{des}}\|^2 + \varepsilon\|\boldsymbol{q} - \boldsymbol{q}_0\|^2 \\ &\text{subject to} \quad \boldsymbol{q}_{\min} \leq \boldsymbol{q} \leq \boldsymbol{q}_{\max} \end{aligned}, \tag{2.22}$$

where $\boldsymbol{x} \in \mathbb{R}^m$ is a generic task or aggregation of tasks, and the error $\boldsymbol{x}(\boldsymbol{q}) - \boldsymbol{x}_{\text{des}}$ must be interpreted as simple difference for Euclidean tasks, or with the appropriate error metric for SO(3) tasks.

2.3.2 Online inverse kinematics

Computing the IK solution via (2.22) requires the solution of non-linear and, in general, non-convex program. For complex, multi-legged systems such as the Centauro robot, this strategy might turn out to be too computationally expensive to be applicable to trajectory

tracking problems, where we want the robot links to track some reference signal in an online fashion. Online IK is clearly of paramount importance for tele-operated systems, and also to execute on the robot trajectories that are natively expressed in Cartesian space (as e.g. for the walking pattern generation). In such cases, one can make use of the differential kinematics equation with the aim to compute, at each control loop, joint space velocities which realize a given task-space velocity. More in detail, given our task of interest $\boldsymbol{x}(\boldsymbol{q}) \in \mathbb{R}^m$, we compute its variation as

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{2.23}$$

Then, given a desired signal for the task value $\boldsymbol{x}_{\text{des}}(t)$, we compute a reference value for the task velocity such that the tracking error asymptotically converges to zero. To this aim, let $\boldsymbol{e} \in \mathbb{R}^m$ be the task error

$$\boldsymbol{e} = \boldsymbol{x} - \boldsymbol{x}_{\text{des}}; \tag{2.24}$$

then, let us define an exponentially converging error dynamics as follows:

$$\dot{\boldsymbol{e}} + K_p \boldsymbol{e} = 0, \tag{2.25}$$

where K_p is an m-by-m symmetric and positive definite matrix gain. Such a dynamics ensures convergence to zero from an arbitrary initial error. Finally, from (2.25), the following reference task velocity is obtained in the form of a proportional law, as shown the equation below:

$$\dot{\boldsymbol{x}}_{\text{ref}} = \dot{\boldsymbol{x}}_{\text{des}} + K_p \boldsymbol{e}. \tag{2.26}$$

The task velocity required by (2.26) can be obtained by finding a solution to the linear system

$$\boldsymbol{J}(\boldsymbol{q}_k)\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}_{\text{ref}}, \tag{2.27}$$

where, at the current time t, $\boldsymbol{q}_k$ is the known joint position value at the k-th control loop, leaving $\dot{\boldsymbol{q}}$ as the only unknown to be computed. When a solution $\dot{\boldsymbol{q}}^*$ is found, it is then integrated for Δt seconds (i.e. the control period) in order to find $\boldsymbol{q}_{k+1}$

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \dot{\boldsymbol{q}}^* \Delta t; \tag{2.28}$$

note that, being (2.27) a linear system, computing one of its solutions is comparatively cheap. Furthermore, powerful mathematical tools can be employed in order to deal with redundancy, i.e. the existence of infinitely many solutions to (2.27). For instance, the solution on (2.27) with minimum L2-norm is given by applying the *Moore-Penrose pseudoinverse* of $\boldsymbol{J}$, denoted

by $\boldsymbol{J}^\dagger$, as follows:

$$\dot{\boldsymbol{q}}^* = \boldsymbol{J}^\dagger \dot{\boldsymbol{x}}. \tag{2.29}$$

2.3.3 Pseudo-inverse

Given *any matrix* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, its Moore-Penrose pseudo-inverse $\boldsymbol{A}^\dagger \in \mathbb{R}^{n\times m}$ is defined by the following properties

1. $\boldsymbol{A}\boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{A}$;
2. $\boldsymbol{A}^\dagger\boldsymbol{A}\boldsymbol{A}^\dagger = \boldsymbol{A}^\dagger$;
3. $\boldsymbol{A}\boldsymbol{A}^\dagger$ and $\boldsymbol{A}\boldsymbol{A}^\dagger$ are symmetric.

Such a matrix always exists and is unique. The Moore-Penrose pseudo-inverse is especially useful in solving undetermined linear systems and linear least-squares problems; indeed, it can be proved that the set of minimizers of a least-squares function is computed as follows:

$$\boldsymbol{x}^* \in \operatorname{argmin}_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 \iff \boldsymbol{x}^* = \boldsymbol{A}^\dagger \boldsymbol{b} + (\boldsymbol{I}_n - \boldsymbol{A}^\dagger\boldsymbol{A})\boldsymbol{x}_0, \tag{2.30}$$

where $\boldsymbol{x}_0$ is a vector in $\mathbb{R}^n$. Moreover, the solution given by the right hand side of (2.30) is the one, among all minimizers, which is closest to the point $\boldsymbol{x}_0$; consequently, putting $\boldsymbol{x}_0 = 0$ gives the minimum norm element. The pseudo-inverse can be computed as

$$\boldsymbol{A}^\dagger = \boldsymbol{A}^T \left(\boldsymbol{A}\boldsymbol{A}^T\right)^{-1} \tag{2.31}$$

if $\boldsymbol{A}\boldsymbol{A}^T$ is invertible (i.e. $\boldsymbol{A}$ is a full rank fat matrix), or as

$$\boldsymbol{A}^\dagger = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1} \boldsymbol{A}^T \tag{2.32}$$

if $\boldsymbol{A}^T\boldsymbol{A}$ is invertible (i.e. $\boldsymbol{A}$ is a full rank tall matrix). If $\boldsymbol{A}$ is (close to) rank deficient, a *damped pseudo inverse* approximation can be computed as follows:

$$\boldsymbol{A}^\dagger \approx \boldsymbol{A}^T \left(\boldsymbol{A}\boldsymbol{A}^T + \lambda\, \boldsymbol{I}_m\right)^{-1} = \left(\boldsymbol{A}^T\boldsymbol{A} + \lambda\, \boldsymbol{I}_n\right)^{-1} \boldsymbol{A}^T, \tag{2.33}$$

for some small positive value of the regularization λ. This is often referred to as *Tikhonov regularization*, and it is also possible to prove that, in the limit, it provides the correct value

of the pseudo-inverse even in the rank deficient case, i.e.

$$\lim_{\lambda \to 0} \boldsymbol{A}^T \left(\boldsymbol{A}\boldsymbol{A}^T + \lambda \boldsymbol{I}_m\right)^{-1} = \lim_{\lambda \to 0} \left(\boldsymbol{A}^T \boldsymbol{A} + \lambda \boldsymbol{I}_n\right)^{-1} \boldsymbol{A}^T = \boldsymbol{A}^\dagger. \tag{2.34}$$

It is important to notice how pseudo-inversion is *not* a continuous operation, and it can have a pretty unstable behavior depending on the input matrix. For instance, let us consider the following family of matrices, parametrized by a scalar ε:

$$\boldsymbol{A}(\varepsilon) = \begin{bmatrix} 0 & 1 \\ \varepsilon & 1 \end{bmatrix}; \tag{2.35}$$

for any $\varepsilon \neq 0$, such a matrix is invertible, and its pseudo-inverse coincides with its inverse, that is

$$\boldsymbol{A}(\varepsilon)^\dagger = \begin{bmatrix} -\frac{1}{\varepsilon} & \frac{1}{\varepsilon} \\ 1 & 0 \end{bmatrix}, \quad \forall \varepsilon \neq 0. \tag{2.36}$$

Note how some entries of the matrix go to infinity as ε approaches zero. However, the pseudo-inverse of $\boldsymbol{A}(0)$ is well-behaved:

$$\boldsymbol{A}(0)^\dagger = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}. \tag{2.37}$$

Such instabilities (which are not due to numerical issues but to inherent ill-posedness of the linear system solution) must be dealt with appropriately for robot control purposes, by defining an *approximate* pseudo-inverse which has desirable stability properties. A simple way to do so is to form a damped pseudo-inverse as in (2.33), possibly having the regularization parameter λ depend on how far from singular the matrix $\boldsymbol{A}\boldsymbol{A}^T$ is. A more sophisticate approach uses the *singular value decomposition (SVD)* of the input matrix, which allows to selectively damp the direction along which it is singular or close to singular.

2.3.4 Singular value decomposition

Given an arbitrary matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, it can be decomposed into the product of three matrices (SVD) as follows:

$$\boldsymbol{A} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^T, \tag{2.38}$$

where:

- $\boldsymbol{U} \in \mathbb{R}^{m\times m}$ is an orthogonal matrix, i.e. its columns $\boldsymbol{u}_i \in \mathbb{R}^m$ form an orthonormal basis; such vector are called *left singular vectors*.
- $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$ only has non-zero elements on its diagonal. Diagonal elements

$$\sigma_i = \Sigma_{ii} \quad i = 1, \dots, \min(m,n),$$

  called *singular values*, are non-negative, and are sorted in decreasing order.
- $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ is also an orthogonal matrix, and its columns $\boldsymbol{v}_i \in \mathbb{R}^n$ are called *right singular vectors*.

By virtue of such properties, the following relation holds:

$$\boldsymbol{A}\,\boldsymbol{v}_i = \begin{cases} \sigma_i\,\boldsymbol{u}_i & \text{if } i \le m \\ \boldsymbol{0} & \text{otherwise} \end{cases}, \quad i \in \{1, \dots, n\} \tag{2.39}$$

recalling that both $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ have unit norm, (2.39) let us interpret the singular values as *gains* from input space to output space, which are in general different depending on the input direction. Moreover, a null-space orthonormal basis can be readily obtained by collecting all right singular vectors $\boldsymbol{v}_i$ for which either $\sigma_i = 0$ or $i > m$.

Let us now discuss the role of SVD-related quantities as far as our differential kinematics

$$\boldsymbol{J}\,\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}} \tag{2.40}$$

is concerned. To this aim, let $\boldsymbol{J}$ be a fat matrix, i.e. $n > m$ ($m = 6$ for a full Cartesian task), which is the common case when working with redundant robots. First of all, we observe that **right singular vectors** are elements of the input space, i.e. they represent joint velocities. For all $i \in \{m+1, \dots, n\}$, the joint velocities $\boldsymbol{v}_i$ do not cause any motion of the end-effector, but only self-motions of the kinematic tree. Such joint velocities belong to the null-space of $\boldsymbol{J}$, denoted as $N(\boldsymbol{J})$. On the opposite side, joint velocities $\boldsymbol{v}_1, \dots, \boldsymbol{v}_m$ do cause the end-effector to move with velocity given by

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\,\boldsymbol{v}_i = \sigma_i\,\boldsymbol{u}_i; \tag{2.41}$$

hence, joint velocity values directed as $\boldsymbol{v}_1$ achieve the largest Cartesian velocity, because $\sigma_1 = \sigma_{\max}$ is by definition the largest singular value.

Left singular vectors are instead elements of the output space, i.e. they are velocity twists. The velocity twist $\boldsymbol{u}_i$ can be obtained with a suitable choice of joint velocity if and only

if its corresponding singular value is greater than zero. In such a case, the minimum-norm joint velocity achieving the required twist is clearly

$$\dot{\boldsymbol{q}} = \frac{1}{\sigma_i}\boldsymbol{v}_i, \tag{2.42}$$

which is by all means the same minimum-norm solution as obtained in (2.29) via pseudo-inversion. Indeed, given the SVD of the Jacobian matrix, one can find the minimum-norm solution to the differential inverse kinematics by (i) rotating the target twist $\dot{\boldsymbol{x}}$ by the matrix $\boldsymbol{U}^T$ in order to find its components along the left singular vectors, then (ii) dividing each component by the corresponding singular value (gain), and finally (iii) rotating the obtained joint velocity (which is expressed in $\boldsymbol{v}_i$ coordinates) back to the input space multiplying by $\boldsymbol{V}$. Mathematically this reads as follows:

$$\dot{\boldsymbol{q}}^* = \boldsymbol{V}\boldsymbol{\Sigma}^\dagger \boldsymbol{U}^T \dot{\boldsymbol{x}}, \tag{2.43}$$

where the matrix appearing at the right hand side can be proved to be the Moore-Penrose pseudo-inverse of the Jacobian:

$$\boldsymbol{J}^\dagger = \boldsymbol{V}\boldsymbol{\Sigma}^\dagger \boldsymbol{U}^T. \tag{2.44}$$

It is readily noticed how Cartesian velocities with high corresponding gain (i.e. singular value) can be obtained by the robot with low effort (i.e. joint velocity), whereas Cartesian velocities with low gain might require excessive control effort. Clearly, this procedure must be regularized such that unsafe motions are never obtained; the ability to compute separate gains for each direction of the rigid body velocity space allows us to regularize the pseudo-inverse in a selective way, i.e. by only perturbing the directions with low gain, as opposed to the Tikhonov regularization strategy of (2.33). In practice, the singular value inversion step (2.42) is replaced with a smoother function ensuring that inverse singular values never exceed a given threshold, and that asymptotically behaves like $\frac{1}{\sigma}$, such as:

$$\mathrm{inv}(\sigma) = \frac{\sigma}{\sigma^2 + \left(\frac{\sigma_0}{2}\right)^2}, \tag{2.45}$$

whose maximum value is $\mathrm{inv}(\frac{\sigma_0}{2}) = \frac{1}{\sigma_0}$. As a side note, it is often a good idea to apply regularized inversion to the *normalized* singular values w.r.t. $\sigma_{\max}$, thus setting $\sigma_0 \ll 1$, such that the value of σ_0 is consistent across different robots and tasks. It is also possible to define a piece-wise continuous regularized inverse function which perform *exact* inversion beyond a given threshold, in order to effectively disable any regularization on directions with high

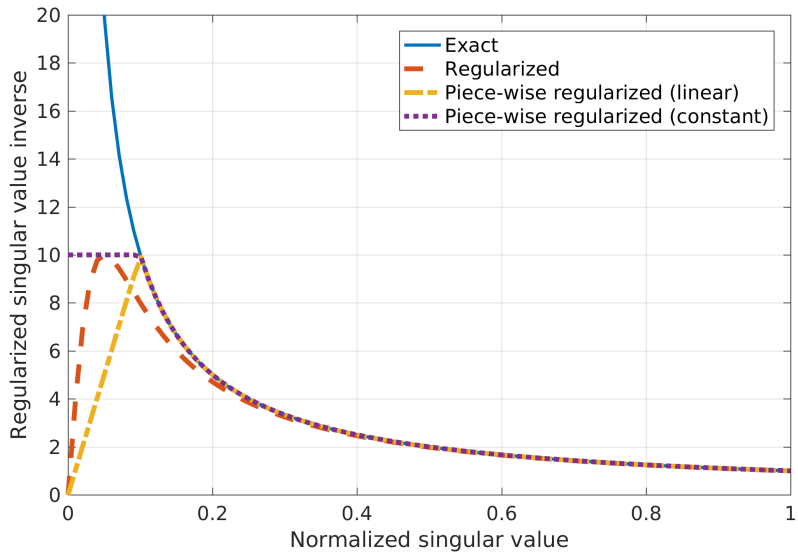


Figure 2.5 Different strategies for the regularized inversion of singular values, with threshold $\sigma_0 = 0.1$. The exact inversion (blue line) is unstable for small values of σ. Regularized inversion is always bounded, and passes through the origin (indeed, $0^\dagger = 0$); yet, its value is very conservative up to roughly $2\sigma_0$. Piece-wise regularized inversion is by definition exact above σ_0; the linear variant is to be preferred as it respects $\mathrm{inv}(0) = 0$.

gain, such as

$$\mathrm{inv}(\sigma) = \min\left\{\frac{1}{\sigma_0^2}\sigma,\ \frac{1}{\sigma}\right\}. \tag{2.46}$$

A pictorial view of these strategies is given in Figure 2.5.

2.3.5 Connection with NLP-based IK

Online IK based on the inversion of differential kinematics is strongly connected to non-linear one shot IK, as presented in Section 2.3. The non-linear least squares cost function of (2.22), that is

$$\boldsymbol{f}(\boldsymbol{q}) = \|\boldsymbol{x}(\boldsymbol{q}) - \boldsymbol{x}_{\mathrm{des}}\|^2, \tag{2.47}$$

has derivatives (dropping the explicit dependence on $\boldsymbol{q}$)

$$\begin{aligned} \nabla \boldsymbol{f} &= \boldsymbol{J}^T(\boldsymbol{x} - \boldsymbol{x}_{\mathrm{des}}) \\ \nabla^2 \boldsymbol{f} &= \boldsymbol{J}^T\boldsymbol{J} + \sum_{i=1}^{m}\left[(x_i - x_{\mathrm{des},i})\nabla^2 x_i\right]. \end{aligned} \tag{2.48}$$

Hence, the gradient descent step is equivalent to the differential inverse-kinematics via Jacobian-transpose:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \eta \nabla \boldsymbol{f} = \boldsymbol{q}_k + \eta \boldsymbol{J}^T (\boldsymbol{x} - \boldsymbol{x}_{\text{des}}), \tag{2.49}$$

where η is the step size, which is chosen as $\eta = K_p \Delta t$ for online IK purposes. Furthermore, the *Gauss-Newton* method requires to neglect second-order terms in the objective function hessian $\nabla^2 \boldsymbol{f}$, and to find a descent direction $\boldsymbol{d}_k$ by solving the linear system

$$\left[\boldsymbol{J}^T \boldsymbol{J}\right] \boldsymbol{d}_k = \boldsymbol{J}^T (\boldsymbol{x} - \boldsymbol{x}_{\text{des}}), \tag{2.50}$$

whose solution always exists, and can be (approximately) obtained as

$$\boldsymbol{d}_k = \left[\boldsymbol{J}^T \boldsymbol{J} + \lambda \boldsymbol{I}_n\right]^{-1} \boldsymbol{J}^T (\boldsymbol{x} - \boldsymbol{x}_{\text{des}}), \tag{2.51}$$

which is equivalent (see equation 2.33) to the (damped) pseudo-inverse based solution of (2.29), and becomes exact when the regularization parameter approaches zero. Overall, velocity-level inverse kinematics can be interpreted as applying a single step of a NLP solver at each control, in an online fashion, and with a fixed step length. Finally, it is worth noticing how the same reasoning, when considering a constrained NLP (e.g. including joint limits), brings to the *Sequential Quadratic Programming (SQP)* step.

2.4 Robot dynamics

Mechanical systems are *second order systems*, i.e. the control input directly affects the second-derivative of the system configuration. Indeed, for a generic articulated body, the relation between joint acceleration and joint torque is linear, and it is given by the *inverse dynamics* equation

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{2.52}$$

where the name "inverse" is due to the fact that it allows to compute the control input *given the motion*, and not vice-versa. Computation of the right hand side value of (2.52) can be done efficiently through a spatial algebra formulation of the *Recursive Newton-Euler Algorithm (RNEA)*, as detailed e.g. in Featherstone (2010). In (2.52), (i) $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ is the joint space inertia matrix, which also allows to compute the mechanism kinetic energy, i.e. $T = \frac{1}{2} \dot{\boldsymbol{q}}^T \boldsymbol{B} \dot{\boldsymbol{q}}$, and is therefore symmetric and positive definite; (ii) $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ is the matrix of centrifugal and Coriolis effects; (iii) $\boldsymbol{g} \in \mathbb{R}^n$ is the vector of gravitational torques (torques needed to compensate for gravity effects); (iv) $\boldsymbol{\tau} \in \mathbb{R}^n$ is the vector of joint torques, i.e. the

control input. Often, centrifugal, Coriolis, and gravitational effects are grouped together in a vector of non-linear terms (or bias torques) $\boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{g}$. In the more specific case of *floating base*, under the assumption that the base orientation be parametrized in a minimal way (see Section 2.2), the inverse dynamics model is modified to the following expression

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_{\mathrm{C}}^T \boldsymbol{F}_{\mathrm{C}}; \tag{2.53}$$

in the floating base case, the configuration vector $\boldsymbol{q}$ is made up by stacking the minimal parametrization of the floating base pose (under-actuated component) with the actuated joint configuration, as follows

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_u \\ \boldsymbol{q}_a \end{bmatrix}, \tag{2.54}$$

so that $n = 6 + n_a$. The matrix $\boldsymbol{S} \in \mathbb{R}^{n \times n_a}$ represents the under-actuation, mapping actuated torques $\boldsymbol{\tau} \in \mathbb{R}^{n_a}$ to under-actuated torques for the full model. Finally, floating base systems are usually subject to *contact forces* $\boldsymbol{F} \in \mathbb{R}^k$, which are responsible for any change in the system center of mass velocity and angular momentum; the *contact Jacobian* $\boldsymbol{J}_C \in \mathbb{R}^{k \times n}$ allows to compute the contact velocity, which is usually zero, i.e. $\boldsymbol{J}_C \dot{\boldsymbol{q}} = \boldsymbol{0}$.

2.4.1 Task-space dynamics

Given a generic task $\boldsymbol{x}(\boldsymbol{q}) \in \mathbb{R}^m$ as in Section 2.3.1, its dynamics equation is obtained by first computing its second derivative

$$\ddot{\boldsymbol{x}} = \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}, \tag{2.55}$$

and then plugging the forward dynamics equation (inverse of equation 2.52) so as to remove the joint acceleration variable; this yields the following *task space dynamics* equation

$$\boldsymbol{\Lambda}(\boldsymbol{q})\ddot{\boldsymbol{x}} + \boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{F}_{\tau|x}, \tag{2.56}$$

where (i) $\boldsymbol{\Lambda} = \left(\boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{J}^T\right)^{-1} \in \mathbb{R}^{m \times m}$ is the task-space inertia matrix, (ii) $\boldsymbol{\mu} \in \mathbb{R}^m$ is the vector of bias forces, and (iii) $\boldsymbol{F}_{\tau|x} = \bar{\boldsymbol{J}}^T \boldsymbol{\tau} \in \mathbb{R}^m$ is the vector of joint torques projected onto the task space. Where $\bar{\boldsymbol{J}} \in \mathbb{R}^{n \times m}$ is known as *dynamically-consistent pseudo-inverse* has the following expression:

$$\bar{\boldsymbol{J}} = \boldsymbol{B}^{-1}\boldsymbol{J}^T \left(\boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{J}^T\right)^{-1}. \tag{2.57}$$

Such a matrix is a *weighted pseudo-inverse*, i.e. $\dot{\boldsymbol{q}}^* = \bar{\boldsymbol{J}}\dot{\boldsymbol{x}}$ is the solution of $\min_{\dot{\boldsymbol{q}}} \boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}$ with minimum weighted L2-norm, as given by $\|\dot{\boldsymbol{q}}\|_{\boldsymbol{B}}^2 = \dot{\boldsymbol{q}}^T \boldsymbol{B}\dot{\boldsymbol{q}}$ (i.e. minimum kinetic energy).

This is not a pseudo-inverse, strictly speaking, since it does not respect the third property of Section 2.3.3.

2.4.2 Centroidal dynamics

Regardless of how complex a mechanical system is, the rate of change of the linear and angular momentum about the robot center of mass is only governed by external forces acting on the robot, whereas joint torques do not have any control authority on such quantities. In robotics literature, the composition of linear and angular momentum (taken w.r.t. the center of mass) is often referred to as the *centroidal momentum*, denoted by the symbol $\boldsymbol{h} \in \mathbb{R}^6$, the first three components corresponding to the linear part (denoted by $\boldsymbol{p} = m\,\boldsymbol{v}_{\text{com}}$), and the last three components corresponding to the angular part (denoted by $\boldsymbol{L}$):

$$\boldsymbol{h} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{L} \end{bmatrix}. \tag{2.58}$$

Both quantities depend *linearly* on the robot joint velocity vector, i.e.

$$\boldsymbol{h} = \boldsymbol{A}(\boldsymbol{q})\,\dot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{A}_p(\boldsymbol{q})\,\dot{\boldsymbol{q}} \\ \boldsymbol{A}_L(\boldsymbol{q})\,\dot{\boldsymbol{q}} \end{bmatrix}. \tag{2.59}$$

At first glance, the equations for the linear and angular part look similar; however, they have very different properties and implications. To see this, let us focus on the case where the centroidal momentum takes a value which is constantly equal to zero, i.e. $\boldsymbol{h} = \boldsymbol{0}$; the linear part is equivalent to $m\,\boldsymbol{v}_{\text{com}} = 0$, which clearly can be integrated as follows:

$$\boldsymbol{A}_p(\boldsymbol{q})\,\dot{\boldsymbol{q}} = \boldsymbol{0} \;\rightarrow\; \boldsymbol{x}_{\text{com}}(\boldsymbol{q}) = const. \tag{2.60}$$

The original constraint on linear momentum, which apparently involves both joint positions and joint velocities, is actually equivalent to a constraint on joint positions alone. Such a constraint is said to be *integrable*, or *holonomic*. On the contrary, it can be proved (Saccon et al., 2017) that, in general, such property does not hold for a similar angular momentum constraint. More specifically, a function of the joint configuration vector whose derivative gives the angular momentum does not in general exist:

$$\nexists \boldsymbol{f} : \mathscr{C} \rightarrow \mathbb{R}^3 \quad \text{such that} \quad \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}} = \boldsymbol{A}_L(\boldsymbol{q}). \tag{2.61}$$

The angular momentum equation is therefore *not integrable*, or *non-holonomic*. As a consequence, a constraint on angular momentum does not necessarily result in a constraint on the configurations that a robot might be able to reach. As noted in Wieber (2006a), animals exploit such a property to always touch the ground on their feet after a fall, no matter the (constant) value of their angular momentum during the motion.

The rate of change of the centroidal momentum is given by *Newton-Euler* equation, which is also known as *centroidal dynamics* equation

$$\begin{cases} \dot{\boldsymbol{p}} &= m\boldsymbol{g} + \sum_i \boldsymbol{f}_i \\ \dot{\boldsymbol{L}} &= \sum_i (\boldsymbol{x}_i - \boldsymbol{x}_{\text{com}}) \times \boldsymbol{f}_i + \boldsymbol{m}_i \end{cases}, \tag{2.62}$$

where (i) $\boldsymbol{F}_i = \left[\boldsymbol{f}_i^T \ \boldsymbol{m}_i^T\right]^T$ are external wrenches acting on the system, (ii) $m\boldsymbol{g}$ is the gravity force, and (iii) $\boldsymbol{x}_i$ are the points where forces are applied. From a kinematic point of view, the centroidal momentum derivative is also given by

$$\dot{\boldsymbol{h}} = \boldsymbol{A}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \dot{\boldsymbol{A}}(\boldsymbol{q},\dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}}; \tag{2.63}$$

It is also worth noticing how (2.62) and (2.63) are equivalent to the first six rows of the inverse dynamics equation (2.53) which, indeed, only have external forces in their right hand side, and not torques.

Remark. *Concerning notation, in this section we have used the symbol* $\boldsymbol{x}$ *to indicate point positions, as in this context the symbol* $\boldsymbol{p}$ *is reserved for momentum. In other contexts (such as Chapter 6), the letter* $\boldsymbol{p}$ *will be used for point positions, and* $\boldsymbol{x}$ *shall indicate a dynamic system state.*

2.4.3 Contact dynamics

The fact that joint torques do not directly affect the robot centroidal dynamics can be counterintuitive. Indeed, from common experience, we know that by varying our muscular effort, it is indeed possible to modulate, e.g., the center-of-mass position. This is due the dynamic constraints acting on the robot when subject to *contacts*. Mathematically speaking, a contact has the form of an holonomic constraint

$$\boldsymbol{f}_C(\boldsymbol{q}) = \boldsymbol{0}, \tag{2.64}$$

where $\boldsymbol{f}_C : \mathbb{R}^n \to \mathbb{R}^m$, and m is the constraint dimension. Contact constraints are enforced through *contact forces*, denoted as $\boldsymbol{F}_C \in \mathbb{R}^m$. For instance, a quadrupedal robot with point feet is subject to a constraint $\boldsymbol{p}_i(\boldsymbol{q}) = \bar{\boldsymbol{p}}_i$, for all feet indexes $i = 1, \ldots, 4$, which states that the position of each foot is always equal to a constant value. In such a case, $m = 12$ and the contact force vector is computed by stacking the four pure forces exchanged between the feet and the ground.

In order to analyze the effect of such a constraint on the robot dynamics, we need to consider (2.53), plus the acceleration-level constraint obtained by differentiating (2.64) twice:

$$\begin{cases} \boldsymbol{B}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) & = \boldsymbol{S}\,\boldsymbol{\tau} + \boldsymbol{J}_{\mathrm{C}}^T \boldsymbol{F}_{\mathrm{C}} \\ \boldsymbol{J}_{\mathrm{C}}\,\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_{\mathrm{C}}\,\dot{\boldsymbol{q}} & = \boldsymbol{0} \end{cases}. \tag{2.65}$$

By exploiting the invertibility of the joint-space inertia matrix, joint acceleration can be computed from the dynamics equation and substituted into the constraint, to obtain

$$\left[\boldsymbol{J}_{\mathrm{C}} \boldsymbol{B}^{-1} \boldsymbol{J}_{\mathrm{C}}^T\right] \boldsymbol{F}_{\mathrm{C}} = \boldsymbol{J}_{\mathrm{C}} \boldsymbol{B}^{-1} \left(\boldsymbol{h} - \boldsymbol{S}\,\boldsymbol{\tau}\right) - \dot{\boldsymbol{J}}_{\mathrm{C}}\,\dot{\boldsymbol{q}}. \tag{2.66}$$

Under the assumption that the contact Jacobian be always *full rank*, the matrix appearing at the right hand side is always invertible, and is actually the inverse of the task-space inertia matrix $\boldsymbol{\Lambda}_C$ corresponding to the contact task, as previously introduced in (2.56). There is therefore a contact-induced relation between contact forces and joint torques and, consequently, also between centroidal momentum and joint torques.

2.4.4 Motion feasibility

Because the robot centroidal dynamics only depend on contact forces, if such quantities must obey certain constraints, also the robot motion must satisfy appropriate constraint as well. This is especially true for floating base robots, since fixed base systems can exchange almost arbitrary wrenches with the environment through the clamp. Mathematically speaking, if $\mathscr{F}_C \subset \mathbb{R}^k$ is the set of allowed contact forces, then considering the top six rows of (2.53), it is clear that a motion $\boldsymbol{q}(t)$ is feasible if and only if

$$\forall t \in [t_0,\, t_f]\ \exists \boldsymbol{F}_C \in \mathscr{F}_C \quad \text{s.t.} \quad \boldsymbol{B}_u(\boldsymbol{q}(t))\,\ddot{\boldsymbol{q}}(t) + \boldsymbol{h}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t))_u = \boldsymbol{J}_{\mathrm{C}}(\boldsymbol{q}(t))_u^T \boldsymbol{F}_{\mathrm{C}}, \tag{2.67}$$

where the subscript "u" refers to the under-actuated part of a matrix, i.e. the top six rows. As shown in Wieber (2002), if the feasible set $\mathscr{F}_C$ is linear, such an existence condition can

be turned into a linear inequality check for the left hand side. On the other hand, it is well known that a robot standing on flat ground cannot be statically stable if the projection of its center of mass lies outside the support polygon; this is a point-in-set check which is way easier to compute, and to interpret. This intuitive property directly comes from the *unilateral* constraint which characterized contact forces, i.e.

$$\boldsymbol{n}^T \boldsymbol{f} \geq 0, \tag{2.68}$$

where $\boldsymbol{n} \in \mathbb{R}^3$ is the outward normal to the contact plane. In order to see this, assuming for the sake of simplicity (i) the ground plane to be horizontal $\boldsymbol{n} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, and (ii) passing through the origin (i.e. the contact plane is described by $\boldsymbol{n}^T \boldsymbol{x} = \boldsymbol{0}$), it is enough to apply the cross product by the contact normal $\boldsymbol{n}\times$ to the angular momentum equation (2.62). Recalling that $\boldsymbol{a} \times (\boldsymbol{b} \times \boldsymbol{c}) = (\boldsymbol{a} \cdot \boldsymbol{c})\boldsymbol{b} - (\boldsymbol{a} \cdot \boldsymbol{b})\boldsymbol{c}$, the following expression is obtained:

$$\sum_i (f_{z,i} \boldsymbol{x}_i) - mg\boldsymbol{x} + m h_{\text{com}}(\ddot{\boldsymbol{x}} - \boldsymbol{g}) = \boldsymbol{n} \times \dot{\boldsymbol{L}} \tag{2.69}$$

where (i) contact torques $\boldsymbol{m}_i$ and center of mass height variations have been neglected for simplicity, (ii) $g = -\boldsymbol{n}^T \boldsymbol{g}$, and (iii) $h_{\text{com}} = \boldsymbol{n}^T \boldsymbol{x}$. Dividing by $\sum_i f_{z,i} = mg$ then yields

$$\frac{\sum_i (f_{z,i} \boldsymbol{x}_i)}{\sum_i f_{z,i}} - \boldsymbol{x} + \frac{h_{\text{com}}}{g}(\ddot{\boldsymbol{x}} - \boldsymbol{g}) = \frac{\boldsymbol{n} \times \dot{\boldsymbol{L}}}{mg}. \tag{2.70}$$

Note now that the first term in (2.70) is a weighted average of the contact points $\boldsymbol{x}_i$, each one weighted according to the portion of robot weight that it supports. As such, this is clearly a point on the contact plane; furthermore, because of the unilateral constraint (2.68), all coefficients are positive; hence, the resulting weighted average is actually a *convex* average, and the resulting point necessarily lies inside the convex hull of the supporting points $\boldsymbol{x}_i$. Such a quantity is the *center of pressure* corresponding to the contact forces, denoted by $\boldsymbol{x}_{\text{cop}}$, i.e.

$$\boldsymbol{x}_{\text{cop}} = \frac{\sum_i (f_{z,i} \boldsymbol{x}_i)}{\sum_i f_{z,i}}. \tag{2.71}$$

Therefore, the force system corresponding to the vectors $\boldsymbol{f}_i$ applied at $\boldsymbol{x}_i$ can be replaced by a single force $\boldsymbol{f}_{\text{eq}} = \sum_i \boldsymbol{f}_i$, applied at $\boldsymbol{x}_{\text{cop}}$, up to a pure momentum about the contact plane normal $\boldsymbol{n}$. Again, this can be easily seen by (i) imposing equality of momentum as in the following equation

$$(\boldsymbol{z} - \boldsymbol{x}_{\text{com}}) \times \boldsymbol{f}_{\text{eq}} = \sum_i (\boldsymbol{x}_i - \boldsymbol{x}_{\text{com}}) \times \boldsymbol{f}_i, \tag{2.72}$$

(ii) applying the $\boldsymbol{n}\times$ operator on the left, and (iii) solving for $\boldsymbol{z}$. Subtracting the right hand side from the left hand side, equation (2.72) can be simplified to

$$\sum_i (\boldsymbol{z}-\boldsymbol{x}_i)\times \boldsymbol{f}_i = \boldsymbol{0}, \tag{2.73}$$

i.e. contact forces exert zero momentum about the center of pressure. This is the reason why, in the robotic walking literature, the center of pressure is often referred to as *Zero Moment Point (ZMP)*.

Under static conditions, and focusing on x and y coordinates, one gets $\left(\boldsymbol{x}=\boldsymbol{x}_{\text{zmp}}\right)_{xy}$, i.e. the projection of the center of mass must be a point inside the support polygon, showing how, as a consequence of the centroidal dynamics equation, constraints on contact forces actually translate into motion constraints. In the static case, this is fairly simple and intuitive; on the other hand, in dynamic conditions, the point which is restricted to the support area becomes

$$\left[\boldsymbol{x}_{\text{zmp}} = \boldsymbol{x} - \frac{h_{\text{com}}}{g}\ddot{\boldsymbol{x}} + \frac{\boldsymbol{n}\times\dot{\boldsymbol{L}}}{mg}\right]_{xy}, \tag{2.74}$$

which is a more involved expression deserving further discussion. A graphical representation of (2.74) is presented in Figure 2.6.

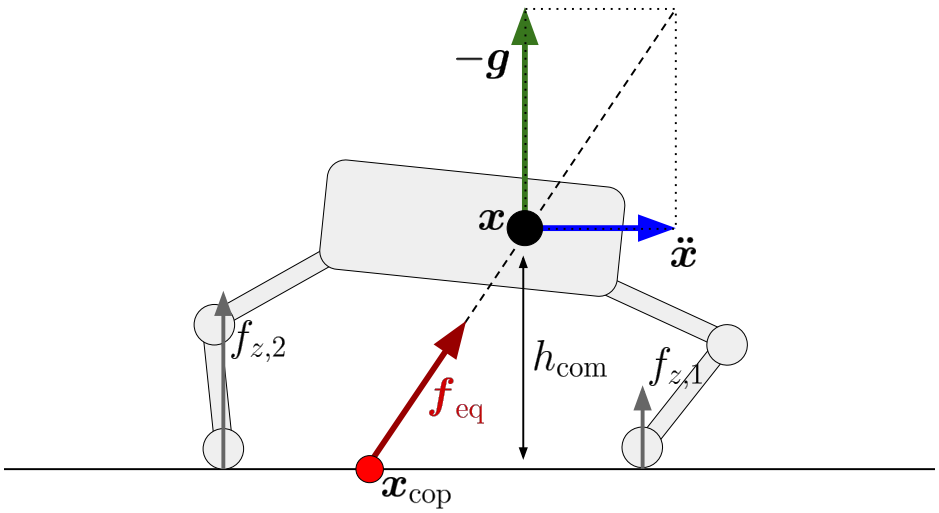
2.4.5 ZMP stability criterion

According to the previous discussion, feasibility of a robot motion under unilateral constraints can be assessed by a simple point-in-set test, where the relevant point is the ZMP (or CoP) as given in (2.74), and the corresponding set is the convex hull of contact points. While the feasibility check is straightforward, the process of generating a feasible motion, i.e. a trajectory $\boldsymbol{q}(t)$ which always satisfies $\boldsymbol{x}_{\text{zmp}}(t) \in \text{ConvHull}\{\boldsymbol{x}_i\}_{i=1}^n$ turns out to be less trivial. In order to simplify the analysis, let us make the common assumption that the robot angular momentum is constant. This yields the so called *Linear Inverted Pendulum Model (LIPM)*[1] for the ZMP, i.e.

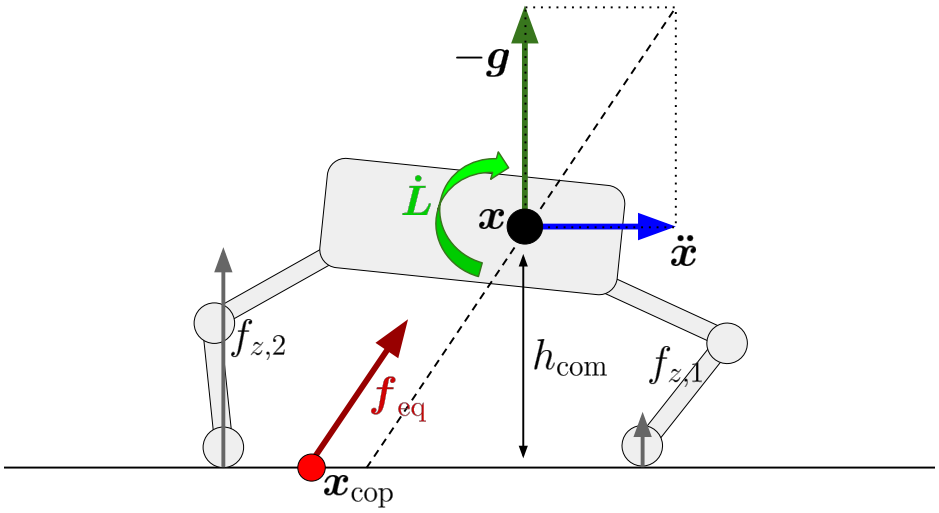
$$\boldsymbol{x}_{\text{zmp}} = \boldsymbol{x} - \frac{h_{\text{com}}}{g}\ddot{\boldsymbol{x}}, \tag{2.75}$$

which has the benefit of expressing the ZMP exclusively as a function of the center of mass dynamics. As we shall shortly verify, satisfying (2.75) indeed requires some care, as constraining the ZMP to a bounded set can easily result in an *unboudned* (i.e. unstable) CoM

[1] The LIPM name is due to the fact that the dynamics of a variable-length inverted pendulum pinned at the CoP, and whose tip is constrained to lie on a plane matches (2.75).



(a) When the rate of change of angular momentum is zero, the center of pressure is obtained by projecting the center of mass position to the contact plane, along the *apparent acceleration* direction. The equivalent force passes through the CoM, so that no momentum is generated.



(b) When the angular momentum is changing, the CoP location is shifted such that the equivalent force does not pass through the CoM anymore, so that the required momentum is generated.

Figure 2.6 Effect of linear and angular momentum rate of change on the center of pressure location, denoted with a red dot.

trajectory. Following Kajita et al. (2003); Lanari et al. (2014), let us consider a walking robot with point feet, i.e. such that its ZMP is always at the contact point. At time $t = 0$ the robot performs a step, and because a single contact point is defined, the ZMP undergoes the same

change as well. The CoM trajectory can then be computed by solving the ODE (2.75) with:

$$\boldsymbol{x}_{\text{zmp}}(t) = \begin{cases} 0 & t < 0 \\ \Delta & t \geq 0 \end{cases}. \tag{2.76}$$

Let us first introduce the quantity $\omega = \sqrt{\frac{g}{h_{\text{com}}}}$, so that our ODE becomes

$$\boldsymbol{x}_{\text{zmp}} = \boldsymbol{x} - \frac{\ddot{\boldsymbol{x}}}{\omega^2}; \tag{2.77}$$

the general solution is

$$\boldsymbol{x}(t) = \begin{cases} \alpha_1 e^{-\frac{t}{\omega}} + \alpha_2 e^{\frac{t}{\omega}} & t < 0 \\ \beta_1 e^{-\frac{t}{\omega}} + \beta_2 e^{\frac{t}{\omega}} + \Delta & t \geq 0 \end{cases}, \tag{2.78}$$

where, matching the CoM position and velocity at $t = 0$ permits to obtain the following expressions for the coefficients:

$$\begin{aligned} \alpha_1 &= \tfrac{1}{2}(\boldsymbol{x}_0 - \omega \dot{\boldsymbol{x}}_0) \\ \alpha_2 &= \tfrac{1}{2}(\boldsymbol{x}_0 + \omega \dot{\boldsymbol{x}}_0) \\ \beta_1 &= \tfrac{1}{2}(\boldsymbol{x}_0 - \omega \dot{\boldsymbol{x}}_0 - \Delta) \\ \beta_2 &= \tfrac{1}{2}(\boldsymbol{x}_0 + \omega \dot{\boldsymbol{x}}_0 - \Delta). \end{aligned} \tag{2.79}$$

A bounded center of mass trajectory is obtained if and only if the coefficients α_1 and β_2 are zero, which leads to the following boundedness conditions:

$$\begin{aligned} \boldsymbol{x}_0 &= \tfrac{\Delta}{2} \\ \boldsymbol{x}_0 + \omega \dot{\boldsymbol{x}}_0 &= \Delta \end{aligned}. \tag{2.80}$$

We therefore observe how, for a given (bounded) ZMP trajectory, the CoM behavior is in general unbounded; only if a condition on the initial state is verified a bounded trajectory is achieved, as depicted in Figure 2.7. Notice also how the second condition in (2.80) implies that the *capture point* $\boldsymbol{x}_{\text{cp}} = \boldsymbol{x} + \omega \dot{\boldsymbol{x}}$ at the stepping time be coincident with the next foothold.

This behavior can be interpreted in the light of system theory by observing that the transfer function from the CoM acceleration to the ZMP has a zero in the right hand side of

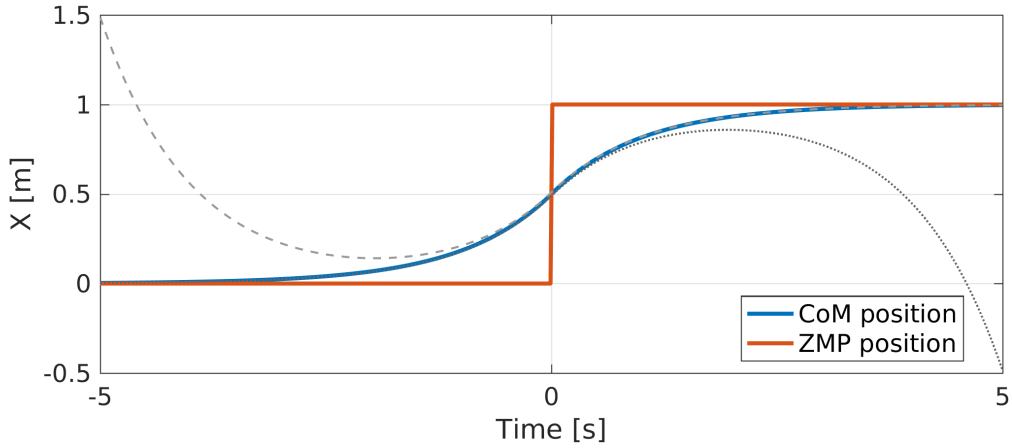


Figure 2.7 Different center of mass trajectories (blue and gray lines) for a given ZMP trajectory (red). Slightly changing the state at $t = 0$ yields unstable CoM trajectories (dashed lines).

the complex plane:

$$\frac{X_{\text{zmp}}(s)}{A_{\text{com}}(s)} = \frac{\omega^2 - s^2}{\omega^2 s^2}; \tag{2.81}$$

therefore, (i) the inverse of such a transfer function is unstable as seen in the point foot walker example, and (ii) there exist unbounded CoM trajectories which still result in a bounded ZMP trajectory. In other words, the *zero-dynamics* corresponding to the ZMP, interpreted as an output of a state-space dynamical system, is *unstable*.

Remark. *It is worth noticing how the only bounded CoM trajectory, as seen in Figure 2.7, is an anti-causal one, i.e. the center of mass starts moving before the robot takes the step at $t = 0$, in order to arrive at the correct initial condition. This observation leads to the conclusion that, in order to generate a bounded CoM trajectory, a preview on future ZMP constraints is needed, as done e.g. in Kajita et al. (2003).*

Part I

Software architecture

Chapter 3

Real-time robotic middleware

This chapter introduces *XBotCore*, a real-time robotic middleware enabling the developer to write control code which runs as close as possible to the hardware, with minimum latency and jitter. A framework with such properties must ensure that (i) critical control code is never interrupted (*preempted*) while running, and (ii) critical control code starts executing as soon as it is needed (usually in reaction of an external event), with minimum latency and jitter. Such guarantees require careful code crafting as well as adequate operating system support, since many standard functionalities of modern programming languages (e.g. memory allocation, input/output, and others) can cause the application to be preempted by the kernel for an unpredictable amount of time. With the aim to facilitate the usage of XBotCore combined with a modern distributed control environment, a lot of effort was put on the integration of basic ROS primitives (e.g. topics and services) directly into the real-time layer, taking care not to lose the aforementioned properties. The remainder of this chapter is devoted to the illustration of how this rather generic design goals have been achieved.

3.1 Motivation and related works

The widespread use of robotics in new application domains outside the industrial workplace settings requires robotic platforms that demonstrate functionality far beyond that of industrial robotic machines. The implementation of the additional capabilities increases the complexity of the robotic platforms at the hardware, control and application software level. As a result, the inevitable complexity of today's robots targeting partially unstructured environments has reached a noticeable extent; e.g., such robots typically employ of a large number of sensors, actuators, and processors executing a large number of control modules that communicate through several interfaces. These emerging applications involve complex tasks, requiring

advanced capabilities with respect to system autonomy and adaptability, which further increases the intricacy of the system software architecture creating an additional challenge on robotic the system's software; therefore, infrastructures that can be quickly and seamlessly adapt to these demands, while providing transparent and standardized interfaces to the robotics developers and users, are required. In this work, the *XBotCore* software framework is introduced; the adopted implementation mechanisms are discussed, from both the developer and user perspective. Its development was driven by the need to provide a robotic low-level middleware that abstracts the variability and complexity of the hardware, while ensuring deterministic hard real-time (RT) performance, and incorporating ready-to-use interfaces that permit it to nicely integrate with state of art robot control frameworks such as ROS. Finally, hard real time performance is made available to developers, who can customize the RT control loop with custom control plugins, which are typically dedicated to performance-critical applications as e.g. high-bandwidth feedback loops.

Several software frameworks for robotics have been developed in the past twenty years (Elkady and Sobh, 2012) targeting to provide flexible infrastructures, which not only permit to easily integrate new functionalities and interfaces in the robotic system, but also to ensure standardization, easy tracking and maintenance of the software development, despite the increased complexity. The selection among these available software middlewares is not an easy task for the research community as well as for companies interested to explore this domain, this being especially true if low-jitter real-time performance is a requirement for the specific application at hand.

Among state-of-the-art robotics middlewares, let us recall *OROCOS* (Bruyninckx, 2002) (Open Robot Control Software), an RT framework, which permits to develop robotics control applications consist of multiple interacting *components*. For strict RT applications, *OROCOS* allows to schedule *components* in a single process while it relies on the *Common Object Request Broker* [1] *(CORBA)* architecture for Inter Process Communication (IPC), implemented in C++ using ACE/TAO[2]. *Components* may run in a single or separated threads depending on their activities. Despite *OROCOS* is used in a fair number of robotics projects, the framework maintenance as well as the community is not active anymore[3].

A very similar framework to *OROCOS* is *OpenRTM-aist* (Ando et al., 2005), developed in Japan from 2002 under NEDO's (New Energy and Industrial Technology Development Organization) "Robot challenge program". It is also based on CORBA, which increases the

[1] http://www.omg.org/spec/CORBA/
[2] http://www.theaceorb.com/
[3] For instance, latest versions (> 3.0) of the *Xenomai* RT patch are not supported.

software complexity for the end-users/developers. Moreover, a part of the *OpenRTM-aist* documentation is available only in Japanese, further impeding its utilization.

PODO (Jeongsoo et al., 2014) is the framework used by KAIST (Korea Advanced Institute of Science and Technology) in the DRC-HUBO robot during the Darpa Robotics Challenge Finals (Lim et al., 2017). Its control system has RT control capabilities and its inter-process communication facilities are based on POSIX IPC; moreover it uses a shared memory system called MPC to exchange data between processes in the same machine. This heterogeneous system has the potential to cause confusion as it is unclear which architectural style must be used to communicate with a specific component (Houliston et al., 2016).

YARP (Yet Another Robot Platform) (Metta et al., 2006) and *ROS* (Robot Operating System) (Quigley et al., 2009a) are popular component-based framework for IPC that do not guarantee RT execution among modules/nodes. However it is essential to have a component responsible for the RT control of the robot, making these frameworks only suitable as external (high-level) software frameworks. It is also worth to mention here that *ROS 2* is moving towards the RT support[4] using the DDS (Data Distribution Service) middleware.

In (Smith et al., 2014) an RT architecture based on OpenJDK is introduced (used by IHMC during the DRC Finals). Nevertheless, to their own admission (Johnson et al., 2015), none of the commercially available implementations of the Java Real Time Specification had the performance required to run their controller. In other words, the existing Real-time Java Support is insufficient.

The above considerations and limitations of the existing frameworks provide the motivations to develop *XBot* framework baring in mind that the design of a software platform, which lies at the foundations of such complex and diverse robotic systems, is a crucial phase in the software development process. *XBot* was designed to be both an RT control system and a user friendly, flexible and reusable middleware for RT and non-RT control software modules. The *XBot* was developed and with following design goals and features in mind:

- **Hard RT control performance**: it must perform computation inside specific timing constraints wirh minimum timing jitter. There are several operating systems or platforms which support RT operation, such as Windows CE, INtime, RTLinux, RTAI, Xenomai, QNX, VXWorks. We selected a Linux based Real-Time Operating System (RTOS) to avoid a licensed product that does not give us the possibility to modify and adapt the source code to fit it to the specifications of our system. In particular, Xenomai satisfies the requirements for extensibility, portability and maintainability as well as ensures low latency as stated in Brown (2012).

[4] http://design.ros2.org/

- **High control frequency**: robotics applications may often require high frequency control loops, e.g. reactive walking, torque-based controllers, or force feedback modules.
- **Cross-Robot compatibility**: it should be possible to use it with any robot, without unnecessary code modification. It is crucial to be able to reuse the software platform with different robots, or subsystems of the same robotic platform. Moreover, standard robotic description files such as the URDF and SRDF formats, should be preferred for parametrization of the architecture.
- **External Framework integration**: it should be possible to use *XBot* as a low-level middleware, to be paired with external software frameworks (real-time or non real-time) without any additional bridge layer tailored for every different case.
- **Plug-in Architecture**: users and third parties should be able to develop and integrate their own modules. In a robotic system platform there is the need for an highly expandable software structure.
- **Simplicity**: avoiding unnecessary (or over-engineered) features simplifies the software maintainability.
- **Flexibility**: *XBotCore* has to be easily modified or extended to be used in systems and applications other than those for which it was specifically designed

As presented in Figure 3.2, the *XBotCore* software architecture is composed by different components, described in detail within the following sections. Each of them has a dedicated role and functionality, contributing to the realization of one or more of the design goals described in this section. Figure 3.3 presents a detailed view of the threads which are spawned in the main components of the framework. It is worth noticing how, aiming to avoid scheduling-related issues and to keep down the complexity of the software infrastructure, XBotCore currently employs only two RT threads and one non-RT thread in the framework.

3.2 Hardware abstraction

The Cross-Robot compatibility feature is achieved through the development of a suitable hardware abstraction layer (Rigano et al., 2018), which enables *XBotCore* to efficiently port and run the same control software modules on different robots, both in simulation and on the real hardware platforms.

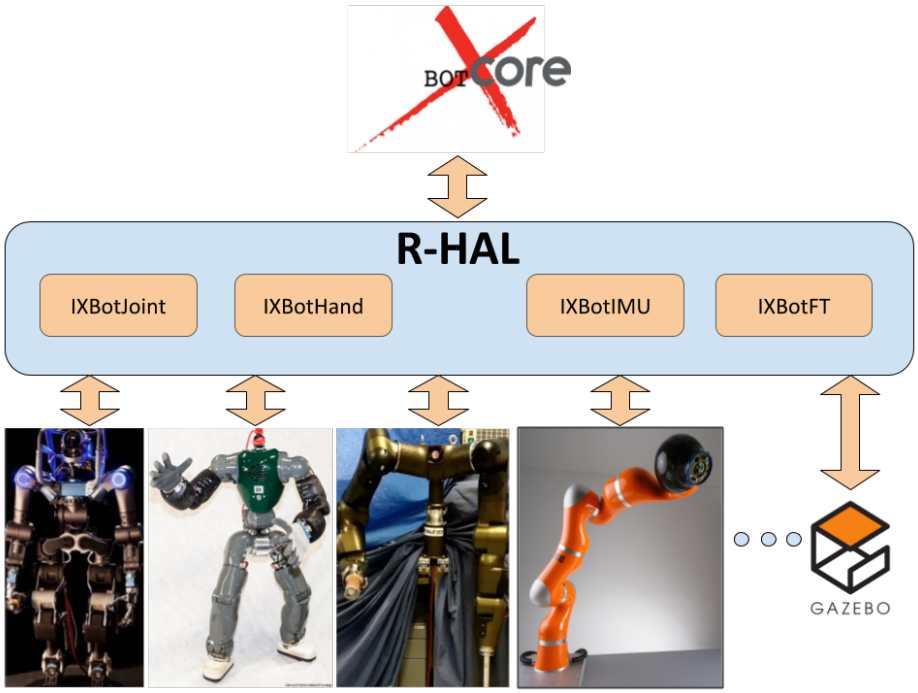


Figure 3.1 The robot Hardware Abstraction Layer introduced for the *XBotCore* software architecture. The *R-HAL* assures high flexibility towards any type of robotic platform or simulation environment.

The main idea is to provide an independent layer with respect to the robot hardware and high level software enabling the integration of new actuators, sensors or other hardware components. The core component, for achieving this, is the *R-HAL* interface, which provide three abstract methods for each of which concrete implementations have to be provided:

- `init()`, used in the initialization phase (open the connection, initialization of the data structures)
- `recvFromSlave()`, serving the reading of all the data coming from the different type of slaves (joints, imus and force torque sensors) and to fill the internal data structures
- `sendToSlave()`, used to send the reference signals to the slave joints

The integration of a new robotic platform requires the implementation of the aforementioned methods in addition to the interfaces that deals with joints, end-effectors, inertial measurement unit sensors and force torque sensors, as shown in orange in Figure 3.1. All the *R-HAL* implementations are built as a shared library object (.so) loaded at runtime according to what specified in a configuration file. In particular, the factory design pattern has been adopted to load/unload several implementations.

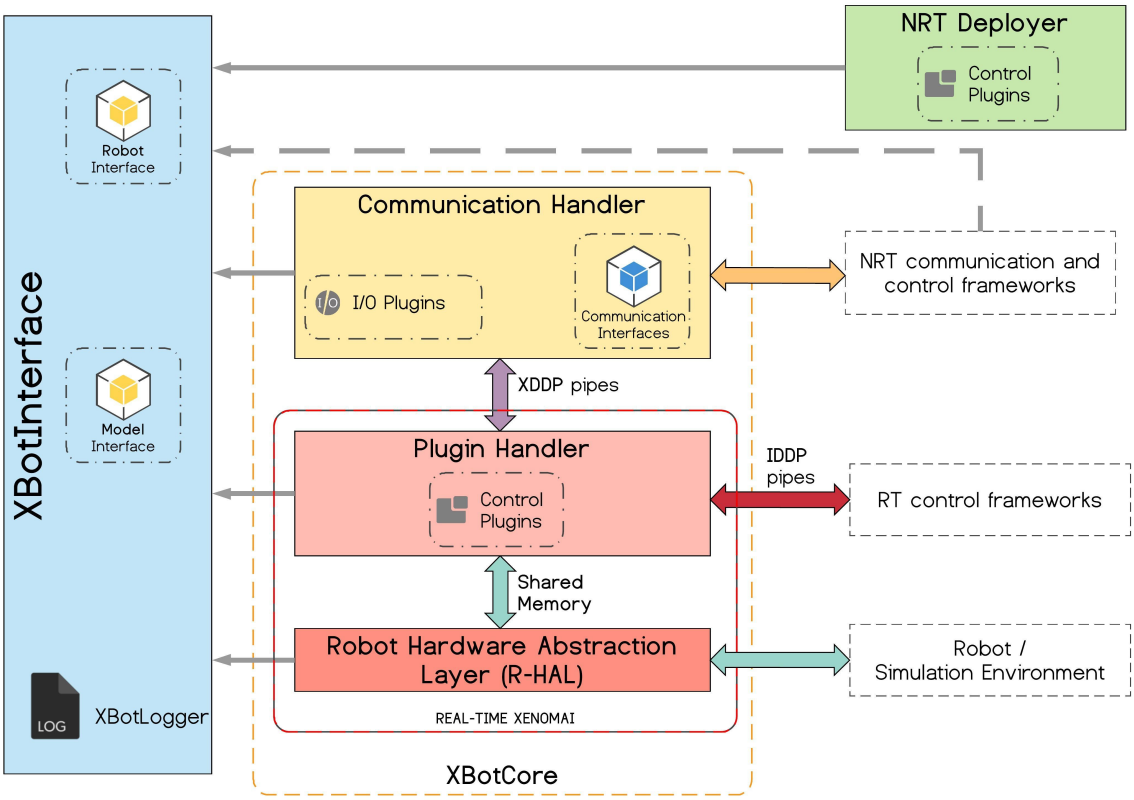


Figure 3.2 *XBotCore* software architecture: components overview and interaction.

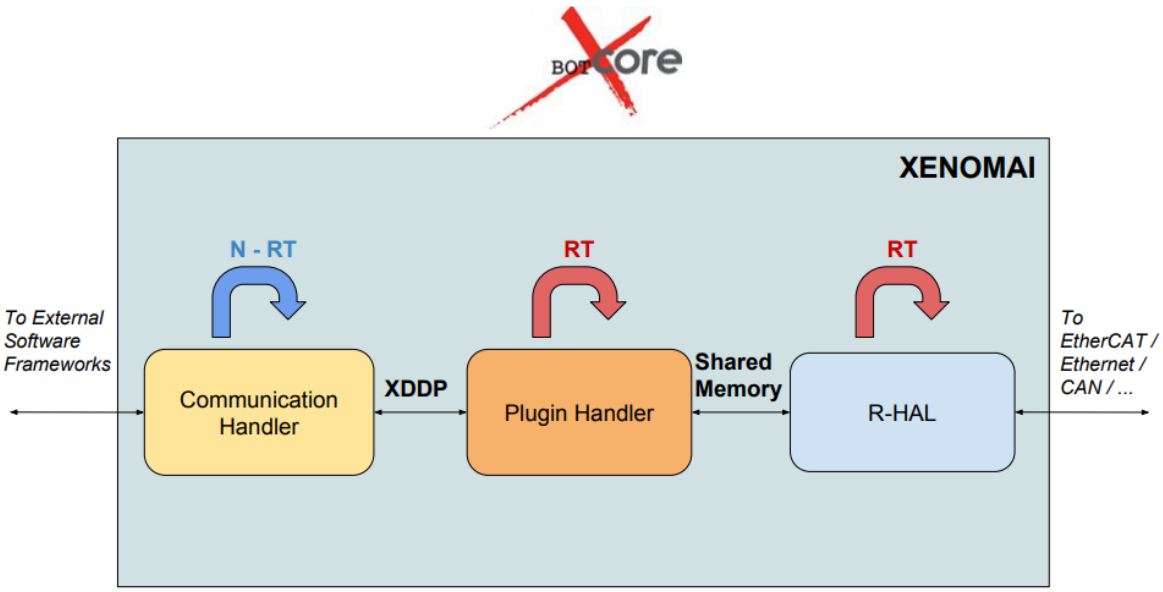


Figure 3.3 *XBotCore* threads structure and communication mechanisms.

Concerning the threading configuration, *XBotCore* employs a separate thread to execute the low level robot control loop and permits to realize controllers with different frequency. The synchronization between the *XBotCore* thread and the *R-HAL* thread is needed to access safely the shared data structures. Dealing with concurrent programming is tedious and error prone since advanced expertise is needed to implement the right synchronization pattern. To address this issue the built-in `HALThread` class is provided to mask all the details about the threads synchronization: the user has only to implement an initialization function and two methods to get and set the robot joint states (position, velocity, torque, stiffness and damping).

XBotCore currently supports EtherCAT (for robot like WALK-MAN, Centauro and COMAN+), Ethernet (for COMAN), and KUKA LWR 4 / KUKA IIWA arm based robots (Baccelliere et al., 2017a; Bischoff et al., 2010; Tsagarakis et al., 2017b, 2013). The possibility to simulate the robot and its controllers behaviours prior to testing on the real hardware is essential, especially when dealing with complex robotic systems. To achieve this, the framework provides an *R-HAL* implementation for the well-known *Gazebo*[5] simulator environment. In particular, simulator support is implemented in terms of a *Gazebo* `ModelPlugin` class, so that the XBotCore control loop executes synchronously inside the physics thread. This way, at the cost of a reduced real time factor, an ideal real time system can be simulated also on machines with limited computational resources.

3.3 Real-time control

The primary objective of this work is, as previously stated, to enable control engineers to have critical modules executed with real-time guarantees. To this aim, a plugin-based architecture has been designed, enabling dynamic loading and unloading of control modules on request. Such plugins implement a basic state machine managing their life cycle, with appropriate virtual methods allowing to customize their behavior. Plugins are executed sequentially by the *PluginHandler*, which is the main component of the *XBotCore* architecture, represented in Figure 3.2 with a dark pink color. The *PluginHandler* is implemented using a single RT thread running at high frequency (e.g. 1 kHz) and is responsible for the actions that are listed below:

1. load the set of plugins requested by the user from a configuration file;
2. initialize all the loaded plugins, and start them upon user request;

[5] http://gazebosim.org/

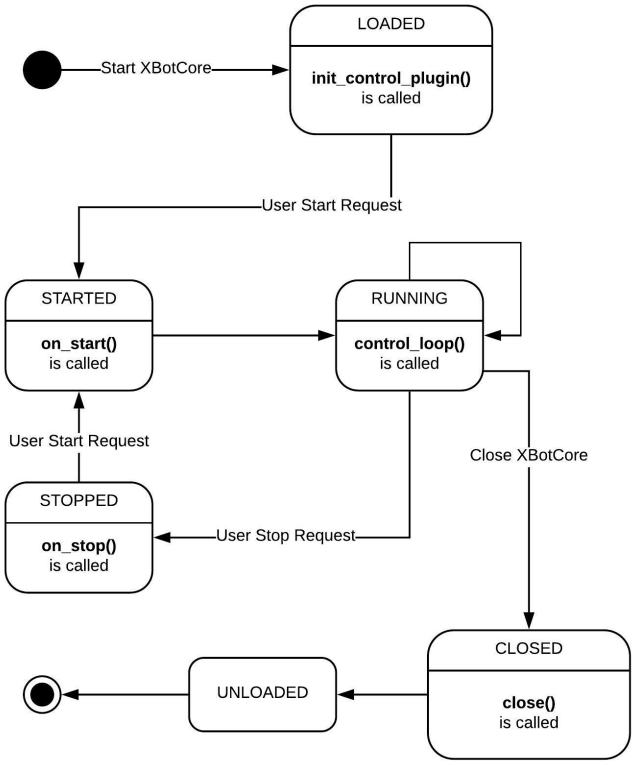


Figure 3.4 UML state diagram showing a *XBotCore* plugin life-cycle.

3. execute the started plugins sequentially;
4. reload and reinitialize a plugin upon user request;
5. close and unload all the loaded plugins.

All plugins follow a simple lifecycle, which is depicted in Figure 3.4. A plugin is created by inheriting from the abstract class *XBotControlPlugin*, and then compiling it into a shared library; this means that writing a Plugin is straightforward for the user, as he or she just needs to implement three basic functions:

- an `init_control_plugin()` function, which is called by the *PluginHandler* after the plugin is loaded/reloaded and is useful to initialize the variables of the Plugin;
- a `control_loop()` function, which is called in the run loop of the *PluginHandler* after the plugin is started;
- a `close()` function, which called in the *PluginHandler* closing phase.

Optional `on_start()` and `on_stop()` methods allow to execute specific action during the first and last control iteration, respectively.

3.4 Non real-time control

Broadly speaking, network capabilities require to perform operations (e.g. system calls) that are not deterministic in their execution time. For this reason, a purely real-time middleware does not give the possibility to communicate with external modules executing outside the robot: for this purpose, the software framework of a robotic system should incorporate a set of non real-time threads that permit the communication of the system with a remote pilot stations or cloud services. To this aim, *XBotCore* implements the `CommunicationHandler` component as a non-RT thread with the primary goal of servicing all operations which would break determinism. Data exchange between such a thread and the PluginHandler is done either via shared memory (relying on lock-free synchronization patterns on the RT side), or by exploiting a Xenomai-specific datagram protocol named XDDP (Cross Domain Datagram Protocol) which achieves asynchronous communication between RT and non-RT threads, without any *mode switches*[6]. Similar to the PluginHandler, a dynamic loading mechanism is employed in order to achieve easy expandability of the systems in terms of the non-RT components that can be loaded. Such components belong to two categories: (i) `CommunicationInterfaces`, which implement the framework-specific robot API (e.g. broadcasting robot TFs and joint states to ROS topics, accepting commands from the framework participants, etc), and (ii) `IOPlugins`, which provide access to the shared memory and XDDP pipes. The execution loop of the `Communication Handler` thread is responsible for (i) updating an internal robot state that is received from the PluginHandler through XDDP pipes with the non-RT robot API, and sending the robot state to all the communication frameworks implemented as `CommunicationInterfaces`; (ii) receiving commands from one specific `CommunicationInterface` (called "master"); (iii) sending the received commands to the robot using the XDDP pipes; (iv) executing all specified `IOPluginss`; (v) servicing real-time ROS publishers and subscribers (more on this in the next section).

[6]In a Xenomai system, threads can operate either in "primary" mode (managed by the real-time scheduler), on in "secondary" mode (managed by the standard Linux kernel). The term *mode switch* then refers to a RT thread involuntarily switching to secondary mode due to a non-RT system call.

3.4.1 ROS integration

During an initial phase of the *XBotCore* development , the target for the real-time part of the proposed architecture was to execute mostly low level control algorithms that would not need too much flexibility for their I/O from and towards the higher level NRT domain. The reader could think, for instance, of a closed-loop inverse-dynamics plugin, which provides feed-forward torques to the actuators, or of a center-of-mass stabilizer that is used for balancing and locomotion. Modules of this kind do not need to exchange much information with external pieces of software. On the contrary, it turned out later that the user base wanted to take advantage of the low latency guarantees offered by the RT layer to implement complex controllers that *do* need flexible I/O in terms of receiving *reference set-points*, online *parameter tuning*, and sending information on the *controller state*. Even though it was possible to build such an infrastructure leveraging the flexibility of the communication handler loop by means of *I/O plugins* (as discussed in Section 3.4) in combination with XDDP communication, the choice has been made to reuse as much as possible standard tools that are well established in the robotics community such as *ROS subscribers*, *publishers*, *service servers*, and *dynamic reconfigure*. While it is not immediately possible to use these tools from the RT domain, the provided threading structure permits to adapt them with moderate effort, and with no required modification to the ROS source code.

More in detail, subscribers, service servers, and dynamic reconfigure are based on callbacks. Since the data reception part involves system calls to the Linux kernel, such callbacks should be processed by a NRT thread, which in the case at hand is the communication handler. Once that data is received, callbacks are packaged as `std::function`-like objects and sent to the RT thread for execution using lock-free queues.

Publishers are slightly more complex to adapt; the standard workflow when using publishers from a normal process would be to produce a message, and then to call a *publish()* function that serializes it and sends the corresponding bytes via e.g. UDP sockets. The data transmission part needs to be performed from a NRT thread, since it involves a system call to the Linux kernel. On the other hand, the serialization part should be done where the message is produced, since it is the only place where its *type* is known. Consequently, it is necessary to "split" the *publish()* function in a way such that the serialization is performed on the RT plugin, while the actual publishing is done on the communication handler. Indeed, ROS allows the power user to take over control of such details by using its advanced API. Summarizing, XBotCore enables the developer of a RT plugin to:

- subscribe to arbitrary ROS topics of any message type (including custom messages) without the need for any adaptation step;
- implement a service server inside a XBotCore RT plugin;
- publish arbitrary messages to a topic;
- tune online the module parameters with the popular ROS dynamic reconfigure tool.

3.5 High level interfaces

After the design and the implementation of the low-latency, hard real-time layer the next significant feature is given by the implementation of flexible interfaces, which permit the XBotCore framework to integrate with state-of-art, widely spread robot control frameworks like ROS, YARP, and OROCOS. During the initial phase of the framework design, the importance of providing the user with a standard way of communicating with the robot was recognized, regardless of its specific structure (humanoid, quadruped, manipulator, etc), and also independently of the particular software layer that the user wanted to operate within. To satisfy this, an API was developed that could be used to send commands to a robot, and receive its current state, from a ROS node, an OROCOS component or a XBotcore RT plugin, in a uniform way, providing the main design goal for the *XBotInterface* library.

An object of the XBotInterface class is essentially a big, organized, container for the robot state, including its on-board controllers. As such, it includes quantities that describe the measurements coming from the robot sensors (e.g. joint position and torque, motor current, IMU states, force/torque sensing etc), and control references (joint position, torque and impedance, etc) as well. As complex robot, e.g. a humanoid can have as many as forty joints, it is important to organize such a state rationally. To this aim, the *XBot::Joint* si defined to be the most atomic component of the library. Then, the *XBot::KinematicChain* object is also defined as a collection of joints belonging to the same chain. This enables the user to specify a joint by the name of the chain it belongs to, and its position inside the chain itself, rather than remembering its literal name or its position inside a possibly huge vector of joints. However, this semantic approach to the robot description comes with disadvantages as well, mainly because of the fact that it is inconvenient to use it for the development of control algorithms, that often rely on the manipulation of the joint states according to the rules of linear algebra. Hence, the framework also provides a full-robot interface that relies on *Eigen3*, a state-of-art linear algebra library. Furthermore, interfaces to two families of

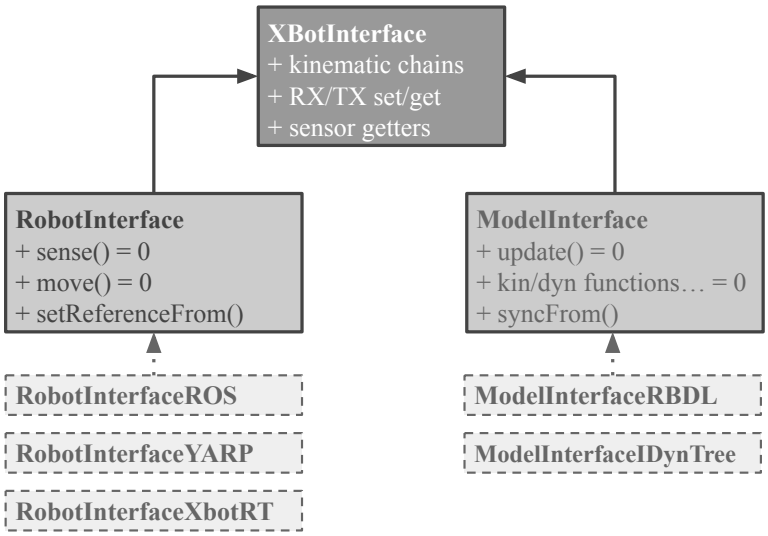


Figure 3.5 UML class hierarchy diagram for the the XBotInterface library.

sensors that are crucially important for real-time control, i.e. force-torque (FT) sensors and inertial measurement units (IMU) were implemented and incorporated in the *XBotInterface* library.

While the XBotInterface class organizes the robot state, to enable actual communication with a robot, the *RobotInterface* class has been defined as a subclass of XBotInterface, introducing a *sense()* method for collecting sensory feedback from the robot, and a *move()* method for sending reference commands to the robot. Both functions are defined as pure virtual, since their implementation depends on whether the RobotInterface is being used from a ROS node, an OROCOS component, or a Xbot RT plugin.

Besides communicating with the robot, it is often that a piece of control code may involve kinematic and dynamic computations, which are performed by some external library. Such library must take a model state (e.g. joint positions, velocities, and acceleration) as input, in to return the joint references (e.g. joint positions or torques) as output. Both states are usually in the form of arrays, which are arranged according to an order that is specific to the library itself. Again, this is an inconvenient and error-prone format for the human user, especially in the case of complex multi-chained robots. In an effort to ease the user's work, we reuse our robot state description, which is the XBotInterface class, with the following three main goals in mind:

- provide a uniform interface not only to the state of actual robots, but also of the corresponding model counterparts;

- standardize the API for retrieving the outcome of the most common kinematic and dynamic computations;
- ease the data exchange between a robot and a corresponding model, and vice-versa.

These objectives are achieved by defining the *ModelInterface* as a subclass of XBotInterface, which mainly adds an *update()* method where the underlying kinematic/dynamic library is updated with the current model state. Similarly, it is a pure virtual function whose implementation depends on the specific back-end that is being used. In addition, a minimal set of pure virtual methods are introduced for every fundamental algorithm from the domains of kinematics (e.g. forward kinematics, differential kinematics, ...) and dynamics (e.g. inverse dynamics, bias terms, mass matrix computation, ...) . As a third step, we also provide one-line functions for synchronizing robots and models, i.e. for setting a model state from the sensory feedback from the robot, and for turning a model state into a reference for the robot controllers. In the author's experience, these two one-liners have turned out to be useful especially when robot and model do not have the same structure, as it is often the case when dealing with complex robots. As a simple example, a manipulation module may use a model for just the upper-body of a legged robot, while the robot object always takes into account the system as a whole. The resulting class hierarchy for the XBotInterface library is summarized in Figure 3.5.

3.6 Conclusions

In this chapter, the *XBotCore* RT software architecture was presented. *XBotCore* provides to the users a software platform capable to work with multiple different robotic system, possibly under different middlewares, with special care to avoid code changes in order to deliver a high degree of flexibility and reusability. The design of the framework assures easy interoperability and built-in integration with other existing software tools for robotics, such as ROS, YARP or OROCOS. The component-based development of the *XBot* includes a Robotic Hardware Abstraction Layer (R-HAL) interface and a set of ready-to-use tools to control robots either inside a simulation environment or in the real hardware.

The framework has been successfully used an validated as a main software infrastructure for humanoid robots such as WALK-MAN (result of WALK-MAN EU FP7 project[7], notably *XBotCore* received the EU innovation radar award in this context[8].) and COMAN+ (result of

[7]https://www.walk-man.eu/
[8]https://www.innoradar.eu/innovation/30632

COGIMON EU H2020 project[9]) or for quadruped centaur-like robots as CENTAURO (result of the CENTAURO EU H2020 project[10]).

Moreover the cross-robot functionality has been exploited to develop both RT and non-RT control modules inside not only the above mentioned robots, but also in industrial arms like *KUKA LBR IIWA*, *KUKA LWR 4+* or *Franka Emika Panda*, or humanoid robots like *COMAN* or *iCub*.

Regarding the simulation part, *XBot* assures the porting of the control modules from the simulator to the real hardware with the same interfaces and requiring no code changes. The built-in simulator supported in the framework is Gazebo, but there is the possibility to support other simulation environments (as it happened inside the CENTAURO H2020 project with the VEROSIM simulator[11]).

XBotCore has become a mature and stable software and control middleware for robotics in the past few years, and it is the official architecture powering all robots in the Humanoid and Human Centered Mechatronics (HHCM).

[9]https://cogimon.eu/
[10]https://www.centauro-project.eu/
[11]https://www.verosim-solutions.com/en/

Chapter 4

Cartesian control

Last chapter introduced XBotCore, a new real-time middleware for robotics systems, enabling the user to implement its control loop inside a low-latency RT control loop. The interfaces offered by XBotCore are at joint level, serving as a building block for the implementation of more complex controllers. This chapter expands our robot control stack with a framework for the Cartesian control of multi-legged, highly redundant robots. The proposed framework allows the untrained user to perform complex motion tasks with robotics platforms by leveraging a simple, auto-generated *ROS*-based interface. Contrary to other motion control frameworks (e.g. ROS *MoveIt!*), the focus of this work is on the execution of Cartesian trajectories that are specified *online*, rather than planned in advance, as it is the case, for instance, in tele-operation and locomotion tasks. Moreover, the problem of generating such motions within a *hard real-time (RT)* control loop is addressed, thanks to tight integration with XBotCore. Finally, the capabilities of the designed framework are demonstrated both on the *COMAN+* humanoid robot, and on the hybrid wheeled-legged quadruped *Centauro*.

This chapter is based on the following article:
Laurenzi, A., Mingo Hoffman, E., Muratore, L., and Tsagarakis, N. G. (2019c). CartesI/O: A ROS Based Real-Time Capable Cartesian Control Framework. In *IEEE International Conference on Robotics and Automation (ICRA)*.

4.1 Motivation and related works

In the past few decades, advancements in robotics have vastly extended its domain of application, moving from the traditional industrial focus to eventually applying robots to human and more unstructured environments. Consequently, robotic platforms have grown in complexity, in an attempt to satisfy the requirements that the new applications demand

in terms of hardware, motion generation, control and human machine interfaces. Indeed, operating effectively in complex environments requires enhanced mobility in order to move on rough surfaces, overcome obstacles, and to carry out complex manipulation tasks that take place in an extended workspace, ranging from the ground-level to above the eye-level.

Such robots exhibit a high degree of redundancy, which however complicates their control from several perspectives: first, tasks have usually *different priorities*; then, when infinitely many solutions exist, some criterion is needed to select one. Finally, the complexity of the code required to control the robot grows considerably: a simple-yet-effective scheme for solving the manipulator inverse kinematics may be realized with few lines of code, whereas the control of highly redundant robots is best done inside a suitable *framework* that hides most of the involved complexity in the motion coordination of redundant robots.

An aspect that is worth highlighting is the dichotomy between tasks that are *pre-planned* offline, and tasks that are specified *online*, in a continuous fashion. As an example from the first family, the reader could consider a reach-to-grasp task taking place in a cluttered environment; such an action can be completely pre-planned before the actual execution, thus opening the possibility of performing a time-consuming search over the robot configuration space in order to find obstacle-free trajectories. Conversely, examples from the second category include tele-operation scenarios, reactive locomotion and physical interaction, and others. Online tasks greatly differ from their pre-planned counterparts, since they require *low computation times* and, especially when any kind of feedback is involved, small delay and jitter in the execution.

Existing works have recognized the need for a Cartesian control framework; yet, it is hard to find a solution that satisfies the following requirements:

- flexible task specification, in terms of both type and number of tasks;
- ability to enforce *soft* priorities as well as *hard* priorities between tasks;
- ability to specify constraints in the task execution;
- small computation time (suitable for online execution);
- possibility to execute inside a *real-time (RT)* thread in order to reduce delays and jitter;
- ease of configuration and use, quick setup time and ready to use control tools;
- parametrized with standard description formats (e.g. *URDF*) in order to support multiple platforms;

- handling of floating base robots.

Previous work from the authors (Hoffman et al., 2018, 2017) goes in the direction outlined by the aforementioned points, resulting in the C++ library *OpenSoT*, which provides tools for writing Cartesian solvers while taking into account priorities and constraints, in a real-time safe way. In the present chapter, additional layers are introduces with a twofold objective: first, to enable the *user* to customize and run Cartesian controllers without writing and compiling C++ code; second, to relieve the *developer* from the need to write custom code to interface a solver with its clients. To achieve this, we design an auto-generated uniform interface to send commands to Cartesian controllers inside the popular *ROS* framework. Then, this interface is exploited to implement an OpenSoT-based, *run-time configurable* controller. Finally, all additional components are designed to be *real-time safe*, and we assess this feature by integrating them into the real-time robot control framework *XBotCore* (see Chapter 3).

	Environment-aware planning capabilities	Real time capable	Generic robot support	Flexible task specification	Floating-base support
OpenRAVE (Diankov, 2010)	✓	✓	✓	✗	✗
MoveIt! (Chitta et al., 2012)	✓	✗	✓	✗	✗
Chorenoid (Nakaoka et al., 2010)	✗	✓	✗	✗	✓
CartesI/O	✗	✓	✓	✓	✓

Table 4.1 Comparison between different frameworks for robot Cartesian control.

The presented framework draws inspiration from previous work on the same topic; the most relevant previous works are summarized in Table 4.1, and discuss them hereafter.

OpenRAVE (Diankov, 2010) is one of the first environments for testing, developing, and deploying motion planning algorithms in real-world robotics applications. From the Cartesian control point of view, OpenRAVE relies on *ikfast* which is an analytic IK engine; it is one of the fastest IK engines available at the moment, however it does not handle redundancy (extra DoFs have to be *locked* manually in order to match the required kinematic structure).

The most widely-spread framework for Cartesian control is probably *MoveIt!* (Chitta et al., 2012), which is part of the open source *ROS* framework. MoveIt! provides a rather complete system of motion planning and execution tools that also take into account the perceived environment. The main core of MoveIt! is based on state-of-the-art sampling-based motion planning algorithms, mostly deployed inside the *OMPL* project (Şucan et al., 2012). Despite MoveIt! has been used on many different types of robot, it lacks explicit support for multi-chains and floating-base (e.g. legged) robots. Furthermore its non-linear planning nature does not make it suitable for on-line (tele-operation) or real-time control.

Another notable work is the software suite *Chorenoid* (Nakaoka et al., 2010), which permits to synthesize whole body motions for bipedal humanoid robots. The user can control different end-effectors as well as perform dynamic motions which are filtered through a *Zero-Moment-Point (ZMP)* based stabilizer. Such a tool has been used successfully on many HRP-series robots, yet it does not permit to setup customized IK problems. This indeed limits the usability of Chorenoid to manipulators and bipeds.

A lot of effort was spent on creating GUI systems for the *DARPA Robotics Challenge (DRC)* (e.g. (Caron and Nakamura, 2015; Fallon et al., 2015; Schwarz et al., 2016)). Yet, none of these works was designed to be general enough to handle different types of robots with considerably diverse structure, or to provide flexibility in terms of tasks or constraints declaration.

4.2 Architecture and implementation

Following a bottom-up description, we can distinguish:

- the *solver*; this is the component that solves a single instance of the mathematical problem that describes the specific Cartesian control algorithm ath hand. In the Cartesian control case, this could be either a *matrix pseudo-inverse* solver, or a *Quadratic Program (QP)* solver. Other possibilities exist, like more general *Non-Linear Program (NLP)* solvers.

- A *modeling language* that allows to construct the aforementioned mathematical problems in a natural and more high-level way, which is less error-prone and time consuming.

- A *base class* for Cartesian controllers, that allows for uniform programmatic usage of any specific implementation. In order to avoid code duplication, it should also perform all the *common work* which would otherwise be replicated by all implementations, as for instance enforcing velocity/acceleration limits, or transforming waypoints into properly interpolated references. In this way, the developer can focus on the controller implementation alone.

- A *middleware interface*, which enables all other processes that compose the control system to send their references in a uniform way that does not depend on the specific implementation running.

This work is mostly concerned with the last two parts, and it is completely decoupled from any specific choice of a solver and modeling language. However, integration with the *OpenSoT* library, which implements a modeling language tailored to Cartesian control, also represents a major goal; the *OpenSoT* library is briefly described in the following section.

4.3 The OpenSoT library

The *OpenSoT* library is a C++ framework which has been initially developed to participate to the DARPA Robotics Challenge as a whole-body inverse kinematics engine. During the years, inverse dynamics and force optimization have also been integrated (Hoffman et al., 2018, 2017) for different robotic platforms. The focus of *OpenSoT* is to ease the formalization of prioritized controllers through dedicated interfaces for *tasks*, *constraints* and *solvers*. Each of these entities are atomic elements which can be combined using a simple syntax, named *Math Of Task*. An example of a simple controller described using the *Math Of Task* is the following one:

$$\begin{pmatrix} \left({}^{\text{Waist}}\mathscr{T}_{\text{RWrist}} + {}^{\text{Waist}}\mathscr{T}_{\text{LWrist}}\right) / \\ \\ \mathscr{T}_{\text{Posture}} \end{pmatrix} << \left(\mathscr{C}_{\substack{\text{Joint}\\ \text{Limits}}} + \mathscr{C}_{\substack{\text{Joint Velocity}\\ \text{Limits}}}\right). \tag{4.1}$$

Two *hard* priorities (slash "/" operator) are specified: the first one is constituted by two tasks (in relative *soft* priority, plus "+" operator) which control the arms end-effectors w.r.t. the *Waist* frame, and the second one is a *Joint Postural* task. All priority levels are subject to *Joint Limits* and *Joint Velocity Limits* constraints ("<<" operator).

The control problem (4.1) is then solved in two steps: first, a *solver front-end* computes matrices and vectors that describe the QP problems for all priority levels, according to one of the prioritized QP (i.e. constrained) formulations introduced in Chapter 2.3, as e.g. the following *optimality constraint*-based approach:

$$\begin{aligned} \min_{\boldsymbol{x}_i} \; & \|A_i \boldsymbol{x}_i - b_i\|^2 + \varepsilon \|\boldsymbol{x}_i\|^2 \\ \text{s.t. } & b_{\text{l}} \leq D\boldsymbol{x}_i \leq b_{\text{u}} \\ & u_{\text{l}} \leq \boldsymbol{x}_i \leq u_{\text{u}} \\ & A_{i-1}\boldsymbol{x}^*_{i-1} = A_{i-1}\boldsymbol{x}_i \\ & \vdots \\ & A_0 \boldsymbol{x}^*_0 = A_0 \boldsymbol{x}_i, \end{aligned} \tag{4.2}$$

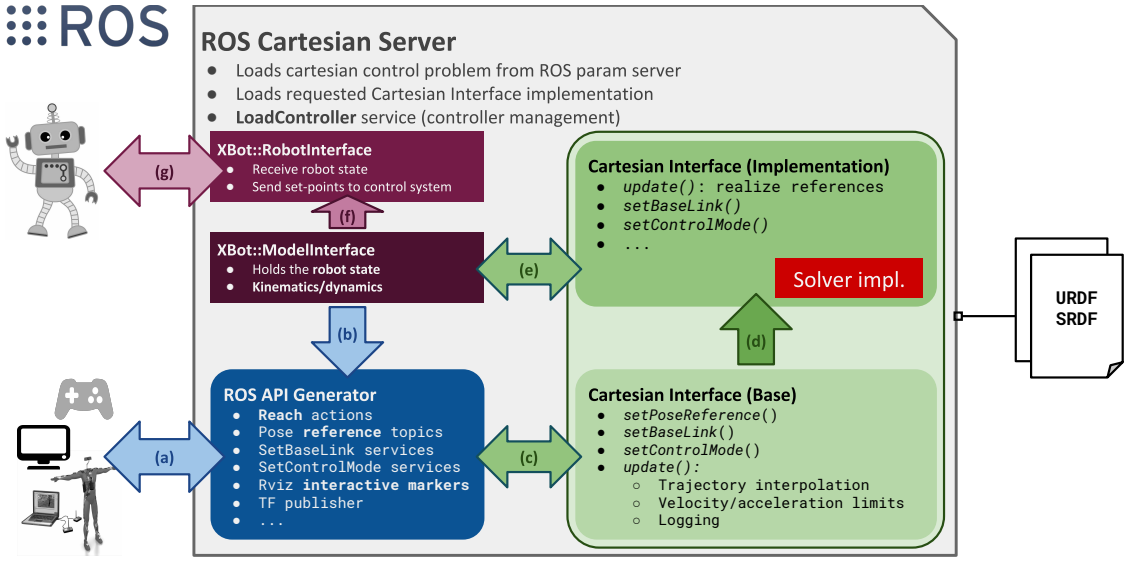


Figure 4.1 Components of the Cartesian Interface and signal flow when the Cartesian Interface is executed inside a ROS node. External processes exchange information with the CartesI/O framework via ROS topics (a), thanks to the *ROS API Generator* component. It gets the current solver state through the *ModelInterface* object (b), in order to broadcast it to the ROS environment. It also forwards the received commands to the *Cartesian Interface base class* (c), where trajectory interpolation and filtering take place. The *Cartesian Interface implementation* component gets these pre-processed signals in order to track them (d). This results in an updated model state (e), which is then sent to the robot actuators through the *RobotInterface* object (f, g).

where $\boldsymbol{x}_i$ represents the vector of decision variables (e.g. joint velocities for IK problems, or joint torques for dynamic formulations) at the i-th priority level.

Then, a *solver back-end*[1] actually computes the corresponding solutions. Notice that the *Math Of Task* formulation, as well as the chosen *solver*, are completely decoupled from the type of controller (velocity or acceleration-level IK, torque control, ...) which depends only on the specific *tasks* and *constraints* implementations.

4.4 Programmatic interface

A modeling language eases the job of formulating and solving a complex optimization problem; yet, the need to write C++ code that is customized for the specific robotic platform and task to be solved remains. This should be avoided, both with a view to promote code reuse, and also considering that, for complex platforms and according to the author's experience, *writing a hierarchy of tasks/constraints that makes the robot show the desired behavior can*

[1] At the present moment, back-ends are available for the *qpOASES* (Ferreau et al., 2014a) and *OSQP* (Stellato et al., 2017) solvers.

be "an art". Therefore, it should be left to "experts in the field", while users should simply customize the problem to better fit their needs. Furthermore, this code would be of little use without an I/O infrastructure that allows a control module to communicate with the external world. Again, the developer should be relieved from writing its own from scratch. With this motivation, this work contributes with an interface layer that:

- provides a uniform way to programmatically interact with a Cartesian controller;
- automatically generates a complete ROS API for sending references to the controller;
- allows to use the ROS-based API also in the case that the *solver* is running inside a real-time thread.

We call the base class that specifies such an interface *CartesianInterface*. It defines simple methods that can be used to change the behavior of all defined tasks, as for instance:

- a *setPoseReference()* method sets the Cartesian set-point corresponding to a task, given its controlled link name;
- the corresponding *setWayPoints()* method is used to specify a point-to-point motion passing through custom waypoints.
- A *setBaseLink()* method can be used to change online the base-link of a task.
- A *setControlMode()* method is used for selecting whether a specific task should be position-controlled, velocity-controlled, or disabled at all. Tasks that are running in velocity mode will discard any position reference, while tasks that are disabled will disappear completely from the Cartesian control problem.
- An *update()* method is used to evaluate all point-to-point trajectories given the current time, and to enforce velocity/acceleration limits by means of the *Reflexxes* library (Kröger, 2011). This method is overridden by the specific subclasses in order to implement their own control loop.

The developer willing to implement its own Cartesian controller can override all these methods in order to take appropriate actions whenever a reference, a control mode or a base-link has changed.

To store the state of the robot, and to perform kinematic/dynamic computation as well, we make use of the *XBot::ModelInterface* class from the *XBotCore* framework (see Chapter 3.5), which acts as a wrapper for rigid body dynamics libraries. Specific implementations of itcan be chosen at runtime, using a dynamic loading mechanism.

4.5 ROS interface

Once that a Cartesian controller has been implemented, a communication layer towards external modules is needed, which is our middleware interface. This is implemented in the form of a C++ class called *ROS API Generator*; for each of the defined tasks, it provides the following functionalities:

- a TF publisher for the model state;
- point-to-point motions with custom waypoints through a custom ROS action;
- commanding continuous pose and velocity references by publishing to ROS topics;
- run-time activation/deactivation of tasks, as well as change of control mode and base-link by calling custom ROS services;
- Rviz *interactive markers* manager (Gossow et al., 2011) that allows for intuitive, GUI-based reference generation;
- *joystick*-based reference generation;
- *joint sliders*, which permit to work at joint space level.

Furthermore, a ROS node that acts as a *Cartesian Server* is provided as well. It loads a user-specified implementation of the *CartesianInterface* class using a dynamic loading mechanism, and initializes the *ROS API Generator*. Finally, a ROS service is provided to dynamically change the controller that is under execution during runtime, while guaranteeing consistency of the model state. The resulting architecture is shown in Figure 4.1: note that, in order to send actual references to the robot, the *XBot::RobotInterface* class from the XBotCore framework is employed, which serves as the robot abstraction layer.

As the final step towards a complete Cartesian control framework as described in Section 4.2, this work provides a generic implementation of the *CartesianInterface* that relies on *OpenSoT* as the modeling language, and on its supported solvers for carrying out the actual optimization procedure. Such a module allows the user to formulate a *hierarchical, whole-body inverse kinematics* problem at the velocity level (as described in Section 4.3); it is written so as to be completely configurable, meaning that the stack of tasks can be specified either on a YAML file or via the ROS parameter server. In this way, starting from a standard description of the robot in terms of its URDF/SRDF, the user can directly run a ROS Cartesian Server to perform rather complex whole-body control tasks with no code

compilation involved; moreover, the auto-generated ROS API provides a convenient way to interact with such a controller from all processes that compose a distributed control system.

4.6 Real-time integration

Whenever a Cartesian controller is based on continuous feedback from the robot, its precise and jitter-free execution at the specified control frequency becomes critical; indeed, delays contribute to destabilize feedback loops, and should therefore be avoided. Typical examples are torque-based controllers (e.g. Cartesian impedance control); however, it is worth noticing that position-based controllers can involve feedback, as in the case of admittance controllers, *stabilizers* for legged robots (e.g. (Zhou et al., 2016)), and tele-manipulation with force feedback.

Such characteristics can be achieved by calling the CartesianInterface *update()* method from within a *real-time (RT)* thread that runs inside a suitable *real-time operating system (RTOS)*; in addition, the communication between this thread and the robot control PC needs to be fast introducing minimum latency. As already noted in Chapter 3, *real-time safeness* of all components of this architecture must be ensured, which broadly speaking means that *all non-deterministic operations should be avoided*, most notably memory allocations and, e.g., network communication.

By careful code development and profiling, the satisfaction of these constraints in ensured both by the *OpenSoT* library (including its solver back-ends), and by the *CartesianInterface* layer. However, it is not possible to run the *ROS API Generator* on the RT layer directly, mainly because of ROS's usage of networking primitives[2]. To address this issue, a *dual-thread* architecture is needed, where a *non-RT thread* runs the ROS API component, while a *RT thread* runs the *CartesianInterface* implementation. In practice, this is done by leveraging on the *XBotCore* framework, that provides us both a RT "control thread" and a non-RT "communication thread", which can be customized via IOPlugins (see Section 3.4). The synchronization between the two is designed to be *non-blocking* for the RT thread, by employing a lock-free callback queue mechanism. This avoids priority inversion problems, and results in a deterministic execution time for our controller, as we experimentally demonstrate in the next sections.

[2]TCP/UDP usage from a real time thread is one of the main long-term goals of the *ROS2* project.

4.7 Validation

To validate the presented *CartesI/O* framework and demonstrate its flexibility in different robot platforms, we set up two manipulation tasks to be carried out by our legged robots *Centauro* (Kashiri et al., 2019), a 39-DoF wheeled-legged quadruped with a humanoid torso, and the 28-DoF humanoid *COMAN+*. In both cases, a box-picking task is selected, where the box must be picked from a low height. In such a case, not only the arms but all the robot chains must coordinate to accomplish the task, which highlights the advantage of using a floating-base whole-body formulation. The outcome of the experiments is summarized in the attached video, which can be found at https://youtu.be/eVmDBVL83WY.

Case study: stabilized box-picking task

As the first experiment, we present an application of the CartesI/O framework to a scenario where the humanoid *COMAN+* has to pick a 3 kg box and pass it to a human. The task is defined such that the robot must reject external disturbances without falling, and it must also show compliance in the reaction in order not to hurt people around it. To achieve compliant rejection of external forces, the work of (Zhou et al., 2016) was integrated in the designed controller where a compliant admittance-base stabilizer is introduced, which essentially computes a *modified CoM reference* from a reference on the *center of pressure (CoP)* (which was kept fixed during the experiment) and from force-torque measurements at the feet as well. Since this experiment involves a feedback controller, the Cartesian solver plus the stabilizer were executed from within the *XBotCore* RT control thread as explained in Section 3.3. To execute the motions, a *ROS SMACH*[3]-based state machine written in Python is run from a different low-priority process that sends target poses to the end-effectors via our auto-generated ROS API as follows:

- the box is grasped and brought to the chest level by sending suitable references to the hands w.r.t. the world frame;
- the box is then passed to the human operator; in order to do so, the base link for the hands tasks is dynamically changed from the world frame to the torso frame; it is therefore enough to command it to rotate about the z-axis of a specified angle.

During the whole demo, the compliant stabilizer is continuously adjusting the CoM reference to track the desired CoP.

[3]http://wiki.ros.org/smach

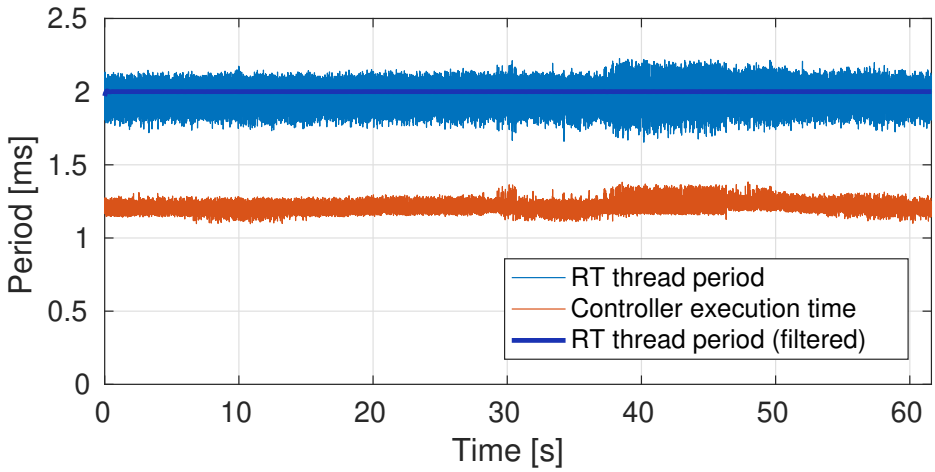


Figure 4.2 Timing statistics for the box picking experiment with *COMAN+*. The thin blue line represents the XBotCore RT thread period. The cartesian controller computation time (including the stabilizer) is depicted in red.

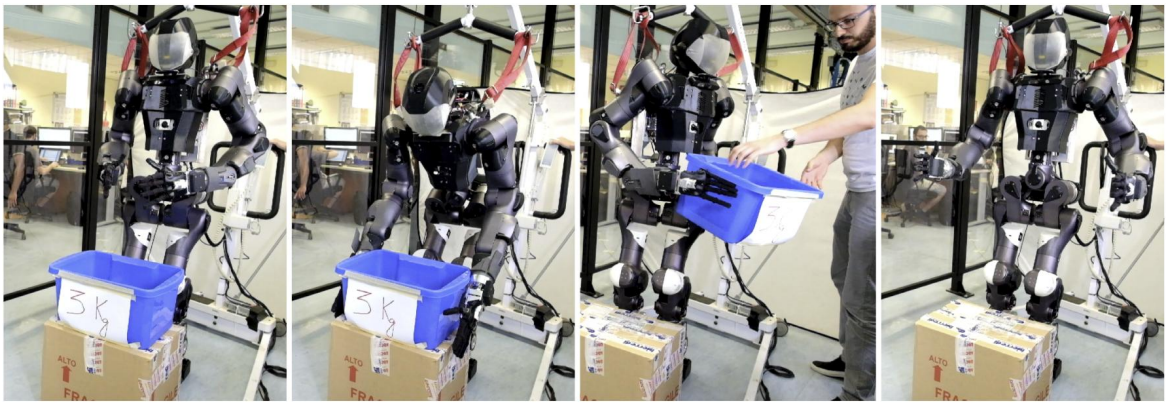


Figure 4.3 Snapshots from a box-picking task with RT stabilization using humanoid COMAN+.

Time statistics regarding the experiment are shown in Figure 4.2; it can be noticed that the RT control thread does indeed meet its deadline in a deterministic way, with a root-mean-square deviation of $T_{\text{jitter}} \approx 90\mu s$, over more than one minute of experiment. Snapshots from the experiment are visible in Figure 4.3.

Case study: ground-level bimanual manipulation

For the second experiment, we choose to pick up a 6 kg brick from the ground using the CENTAURO robot, and then pass it to a human operator. Since the operator is standing on one side of the robot, the robot must perform a brief wheeled-locomotion phase in order to turn ninety-degrees. This is done by switching at runtime between two different

implementation of the *Cartesian Interface*: the first, which is used to pick up the box, is the dynamically-configurable IK controller described in Section 4.4; the second one, called *Centauro Wheeled Motion*, is an IK controller that is tailored to the mixed wheeled-legged locomotion of the *CENTAURO* robot[4]. It is implemented by means of the *OpenSoT* library, and it permits to:

- control the waist pose w.r.t. the world through appropriate steering/rolling of the wheels;
- control the wheels position w.r.t. the waist frame in order to adjust the support polygon shape;
- perform basic control of the end-effectors w.r.t. waist frame.

Since this demonstration does not require to meet hard deadlines, Cartesian controllers are executed from the *ROS Cartesian Server* at a frequency of 100 Hz. Again the robot behavior is scripted via a *ROS SMACH* state machine as follows:

- first, the postural for the front knees is changed in reference, in order to avoid later collisions with the arms;
- then, the end-effectors are commanded to surround the box, then grasp it and lift it up;
- after this, the *Centauro Wheeled Motion* controller is loaded, and commanded to perform a ninety-degrees rotation of the whole robot w.r.t. the world frame;
- finally, we command the arms to open in order to release the box.

The outcome of this experiment is summarized in Figure 4.4, and in the attached video as well.

4.8 Discussion

This chapter has introduced *CartesI/O*, a framework for Cartesian control of floating/fixed-base robots which is focused on *online* execution, possibly under real-time constraints. As the main contribution, generic Cartesian solvers have been integrated inside an architecture that permits to interact with them in a uniform way. This is achieved at two different levels:

- programmatically, through the provided *CartesianInterface* base class;

[4]This simple controller will be then extended in Chapter 5.

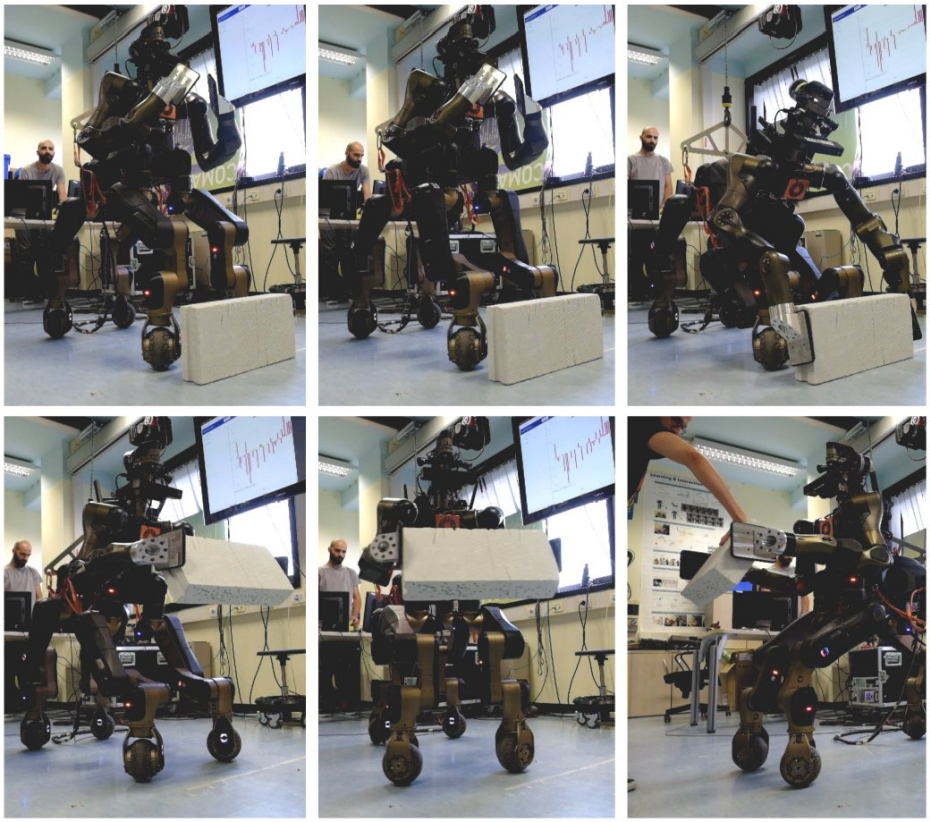


Figure 4.4 Snapshots from the experiment with CENTAURO robot described in Section 4.7.

- from the ROS middleware, via an auto-generated set of topics, services, actions and tools that can be used to monitor the solver state, send references, and customize the solver behavior.

The designed API aims at providing highly flexible task customization during runtime: the user can activate/deactivate tasks, change their base-link, switch between position and velocity control, and even dynamically load different controllers. Furthermore, the framework integrates common tools, such as trajectory interpolation with way-points, as well as enforcement of user-specified velocity/acceleration limits.

Moreover, we provide tools to connect a *real-time* Cartesian controller to our auto-generated ROS API, in order to enable mixed RT/non-RT robot control.

Future work will address the development of solvers working at the dynamics level, as well as the enhancement of the markers capabilities (e.g. Cartesian impedance markers). Moreover, we aim to add previewing capabilities, and environment-aware trajectory planning, e.g. via integration with *MoveIt!*.

Part II

Locomotion control

Chapter 5

Wheeled-legged motion control

This chapter deals with the kinematic control the highly redundant, hybrid wheeled-legged robot Centauro. Given its full wheeled mobility as allowed by its four independently steerable wheels, the choice of some local frame (in addition to the global world) is required in order to express tasks that are naturally defined in a robot-centric fashion. In this chapter, it is shown how trivially selecting such a frame as the robot trunk leads to sub-optimal results in terms of motion capabilities; as main contribution, this work therefore proposes a comparative analysis among three different choices of local frame, and demonstrate that in order to retain all advantages from the whole-body control domain, the kinematic model of the robot must be augmented with an additional virtual frame, which proves to be a useful choice of local frame, enabling e.g. the automatic adaptation of the trunk posture under constraint activation. Our implementation is based on the CartesI/O framework for the inverse kinematics and ROS-based user interfaces. The resulting Cartesian controller is finally validated by means of an extensive experimental session on the real hardware.

This chapter is based on the following article:
Laurenzi, A., Mingo Hoffman, E., Parigi Polverini, M., and Tsagarakis, N. G. (2020). An Augmented Kinematic Model for the Cartesian Control of the Robot CENTAURO. *IEEE Robotics and Automation Letters*. Accepted, to appear.

5.1 Introduction

At the time of writing, the robotic technology is mature enough for performing useful work in a reliable way inside simple environments. On the other hand, operation inside unstructured scenarios is essentially dominated by overly-simplified machines, such as small tracked vehicles, which trade simplicity of their control (and reliability) for a lack of flexibility in

the tasks that they are able to accomplish. The *Centauro* robot has been designed with the aim to take a step forward in the direction of versatility, combining powerful manipulation capabilities with a more reliable quadrupedal hybrid wheeled-legged locomotion concept. Because of its peculiar kinematic structure, Centauro provides significant flexibility in terms of motion control: thanks to the actively steerable wheels, in-place manipulation and stepping motions can be combined with modulations of the support polygon, and driving motions of the whole robot. Broadly speaking, a motion controller which completely exploits the aforementioned kinematic capabilities should provide full control of any task of interest (e.g. gaze, pose control of the trunk, end-effectors, wheels, ...) both with respect to a global world frame, and with respect to a *local frame*, depending on the nature of the specific task at hand. Notice that up to this point, the notion of *local frame* is left rather unspecified, and is just to be regarded as some frame which moves rigidly with the robot support polygon; it will however be a key concept in this work.

Existing works on similar platforms (see e.g. Schwarz et al. (2016)) circumvent this aspect by carefully crafting of all end-effectors desired poses w.r.t. the robot trunk frame such that the robot performs a desired (local or global) task; then, such poses are tracked by a chain-based inverse-kinematics solver. Despite the ability of such a strategy to generate the required motions, it is the author's belief that significant advantages can be obtained by employing a floating-base formulation for the robot model, and by exploiting techniques from the whole-body control domain. The most prominent advantage lies in the structural enforcement of the required relationships between frames directly through the task matrices (relative *Jacobians*), which makes the controller less reliant on feedback gains, more robust in the presence of constraint, and permits to directly specify reference values for each task in their natural coordinate frame.

When applying such concepts to the motion control problem of the Centauro robot, the key question naturally arises of which frame should play the role of *local* frame. Intuition suggests that such a frame should "travel with the robot", and a natural choice would be to select the trunk frame. However, it will be shown in this work how this choice falls short of exploiting the versatility of our target platform. The key observation in the present work is that, if a standard floating base model is employed, such a frame does not actually exist: in order for the robot to fully exploit its kinematic potential in a whole-body manner, *additional degrees-of-freedom must be added to the robot kinematic model*.

Beside the work of Schwarz et al. (2016), other works have previously addressed the motion control problem of articulated wheeled robots. In the work of Kamedula et al. (2018), which targets the same robotic platform of this work, the authors analyze a wheel-ground

contact model under toroidal wheel shape assumptions, and then present a controller for driving motion and support polygon regulation; Bjelonic et al. (2019) integrates wheeled contacts into the torque-based controller for the *ANYmal* robot, yielding effective compliant adaptation to terrain roughness. However, both approaches are not concerned with manipulation capabilities. Finally, Geilinger et al. (2018) introduce a trajectory optimization based approach to the design and control of wheeled-legged robots; even though impressive, highly-dynamic motions are generated, the proposed strategy is too costly for online (e.g. receding horizon) applications.

This work proposes a Cartesian control framework that permits the full control of the Centauro robot in an *online* fashion as specified in terms of a list of *motion requirements* to be defined in Section 5.3. These requirements specify desired relationships *between frames*, which we manage to enforce by writing suitable relative Cartesian tasks inside a floating-base formulation. As it will be illustrated in detail in the forthcoming sections, to achieve this, a novel kinematic model of the robot must be introduced, allowing to define a local reference frame which is *completely decoupled* from the waist frame. As main advantage, the additional degrees of freedom can be exploited by the whole-body solver with the benefit of enabling automatic adaptation of the trunk pose when a constraint on some chain is activated.

The rest of the chapter is organized as follows: Section 5.2 introduces the notation and provides background knowledge about Cartesian control; then, Section 5.3 discusses the details of this contribution. Afterwards, possible strategies to ensure that a realistic contact condition between wheel and ground actually holds are discussed (Section 5.4). Finally, Sections 5.5 and 5.6 conclude the work with an extensive experimental validation of the presented method, followed by a discussion on possible future directions.

5.2 Background on Cartesian control

This section discusses the basic theory at the heart of prioritized, online Cartesian control for highly redundant robots. The presented material extends the background knowledge provided in Chapter 2 with state-of-art techniques to take into account *priorities between tasks*, while ensuring that the computed solutions remain robust even under pathological conditions such as kinematic singularities or incompatible task hierarchy specification.

5.2.1 Kinematic modeling

Being Centauro a legged robot, its configuration space is given not only by its actuated joint positions, but also by the pose of one of its links, which is called the *floating base*. Albeit such a link can be chosen arbitrarily, it is a natural choice to select the trunk as the floating base. Consequently, let $\boldsymbol{q} \in \mathbb{R}^n$ denote the full configuration vector for the robot. Such a vector is obtained by joining the actuated joint configuration vector $\boldsymbol{q}_a \in \mathbb{R}^{n_a}$ with a minimal representation[1] of the floating base pose $\boldsymbol{q}_{\text{fb}} \in \mathbb{R}^6$, as follows:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_{\text{fb}} \\ \boldsymbol{q}_a \end{bmatrix}. \tag{5.1}$$

Then, let $\boldsymbol{x}$ denote a Cartesian task of interest (e.g. the pose of one of the robot links), which is dependent on the robot configuration though some non-linear mapping $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$:

$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{q}); \tag{5.2}$$

by differentiating (5.2), the task velocity can be computed as

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{5.3}$$

where $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ is the task *Jacobian matrix*. Notice that, with an abuse of notation, we identify the quantity $\dot{\boldsymbol{x}}$ with the task velocity twist

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}, \tag{5.4}$$

so that $\boldsymbol{J}$ is the *geometric* Jacobian.

5.2.2 QP-based inverse kinematics

If constraints need to be taken into account while solving the online IK, different tools than the Moore-Penrose pseudo-inverse are needed[2]. Recalling the least-squares, minimum norm interpretation of (2.30), one can reformulate the differential kinematics inversion problem as

[1] For the sake of simplicity, a minimal representation of a rigid body orientation (e.g. Euler angles) is used. A quaternion-based formulation can be employed whenever a singularity free representation is required.

[2] There has also been some study on pseudo-inverse based IK under constraints, such as in Flacco et al. (2015).

a *Quadratic Program (QP)* as follows:

$$
\begin{aligned}
&\min_{\dot{\boldsymbol{q}}} \|\boldsymbol{J}\,\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}\|^2 + \lambda\,\|\dot{\boldsymbol{q}}\|^2 \\
&\text{subject to} \\
&\quad \boldsymbol{q}_{\min} \leq \boldsymbol{q}_k + \dot{\boldsymbol{q}}\,\Delta t \leq \boldsymbol{q}_{\max} \quad \text{(joint limits)} \cdot \\
&\quad -\dot{\boldsymbol{q}}_{\max} \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}_{\max} \quad \text{(velocity limits)} \\
&\quad \boldsymbol{c}_{\min} \leq \boldsymbol{C}\,\dot{\boldsymbol{q}} \leq \boldsymbol{c}_{\max} \quad \text{(other bounds)}
\end{aligned}
\tag{5.5}
$$

Such a problem can then be solved efficiently with state-of-the art off the shelf libraries such as *qpOASES* (Ferreau et al., 2014a), *OSQP* (Stellato et al., 2017), and others, usually in less than one millisecond depending on the robotic system complexity and target hardware. Notice also how the unconstrained minimizer of (5.5) is given by $\dot{\boldsymbol{q}}^* = \left(\boldsymbol{J}^T\,\boldsymbol{J} + \lambda\,\boldsymbol{I}_n\right)^{-1}\boldsymbol{J}^T\,\dot{\boldsymbol{x}}$, which coincides with the (damped) pseudo-inverse solution.

5.2.3 Hierarchical inverse kinematics

When dealing with complex system such as the Centauro robot, it is common that the number of tasks that are involved in the Cartesian control problem grows quite large, needing to accommodate balancing control, contacts, manipulation, gazing, and other functionalities. Multiple tasks might interact in ways that are difficult to predict, whereas it is often desirable for some tasks to have higher priority than others (e.g. balancing or contacts). This is commonly dealt with through the definitions of *weighted L2 norms* in the objective functions (*soft* priorities), so that whenever the solver can not achieve all tasks with zero error (e.g. due to constraint activation, workspace limitations, or singularities) it will redistribute the total task error in a way such that high priority tasks are close to being satisfied. However, this comes with two drawbacks, namely (i) weights are now tunable parameters which must be found via trial and error, and (ii) large values for the weights can cause poor numerical conditioning of the resulting QP problem. A different approach is to define *strict* (or *hard*) priorities between tasks, so that there is a mathematical guarantee that lower priority tasks will not affect the higher priority task performance. This is usually achieved through *cascaded optimization*, i.e. a series of optimization problems is carried out for each priority level, each of which ensures (either implicitly or explicitly) that the optimal cost function value obtained at higher priority levels is never increased. Before providing more details on hierarchical algorithms, let us define $\boldsymbol{J}_k$ to be the task Jacobian at the k-th priority level, $\dot{\boldsymbol{x}}_k$ the desired

task velocity at the k-th priority level, and $\dot{\boldsymbol{q}}_k^*$ be the joint velocity vector resulting from the first k priority levels.

5.2.4 Projector-based unconstrained solution

At the first priority level, we need to solve the following unconstrained quadratic program:

$$\min_{\dot{\boldsymbol{q}}} \|\boldsymbol{J}_1 \dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_1\|^2, \tag{5.6}$$

whose least-squares solution is given by

$$\dot{\boldsymbol{q}}_1^* = \boldsymbol{J}_1^\dagger \dot{\boldsymbol{x}}_1; \tag{5.7}$$

driven by the work of Siciliano and Slotine (1991), recall from (2.30) that the general solution set is parametrized as

$$\dot{\boldsymbol{q}}_1(\boldsymbol{z}) = \dot{\boldsymbol{q}}_1^* + \boldsymbol{P}_1 \boldsymbol{z}, \tag{5.8}$$

where the projector $\boldsymbol{P}_1 \in \mathbb{R}^n$ is given by $\boldsymbol{P}_1 = \boldsymbol{I} - \boldsymbol{J}_1^\dagger \boldsymbol{J}_1$, and $\boldsymbol{z} \in \mathbb{R}^n$ is used to explore the solution set. Then, at the second priority level, the corresponding program

$$\min_{\dot{\boldsymbol{q}}} \|\boldsymbol{J}_2 \dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_2\|^2 \tag{5.9}$$

can be written so as to enforce satisfaction of the $k = 1$ task, by substituting (5.8) inside the cost function, yielding

$$\min_{\boldsymbol{z}} \|\boldsymbol{J}_2 \boldsymbol{P}_1 \boldsymbol{z} + \boldsymbol{J}_2 \dot{\boldsymbol{q}}_1^* - \dot{\boldsymbol{x}}_2\|^2. \tag{5.10}$$

Again, the solution set in terms of the $\boldsymbol{z}$ vector, and the least-squares solution, can be found by applying (5.8), giving rise to the following expressions

$$\begin{aligned} \dot{\boldsymbol{q}}_2(\boldsymbol{z}) &= \dot{\boldsymbol{q}}_1^* + \boldsymbol{P}_1 (\boldsymbol{J}_2 \boldsymbol{P}_1)^\dagger (\dot{\boldsymbol{x}}_2 - \boldsymbol{J}_2 \dot{\boldsymbol{q}}_1^*) + \boldsymbol{P}_2 \boldsymbol{z} \\ \boldsymbol{P}_2 &= \boldsymbol{P}_1 - \boldsymbol{P}_1 (\boldsymbol{J}_2 \boldsymbol{P}_1)^\dagger (\boldsymbol{J}_2 \boldsymbol{P}_1) \\ \dot{\boldsymbol{q}}_2^* &= \dot{\boldsymbol{q}}_2(0). \end{aligned} \tag{5.11}$$

By exploiting appropriate pseudo-inverse properties (Baerlocher and Boulic, 1998; Maciejewski and Klein, 1985)[3], the following equivalent recursion is obtained:

$$\begin{aligned}
\boldsymbol{z}_k^* &= (\boldsymbol{J}_k \boldsymbol{P}_{k-1})^\dagger \left(\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1}^*\right) \\
\dot{\boldsymbol{q}}_k^* &= \dot{\boldsymbol{q}}_{k-1}^* + \boldsymbol{z}_k^* \\
\boldsymbol{P}_k &= \boldsymbol{P}_{k-1} - (\boldsymbol{J}_k \boldsymbol{P}_{k-1})^\dagger \boldsymbol{J}_k \boldsymbol{P}_{k-1} \\
\boldsymbol{P}_0 &= \boldsymbol{I}_n \\
\dot{\boldsymbol{q}}_0 &= 0.
\end{aligned} \tag{5.12}$$

5.2.5 Nullspace-based unconstrained solution

The projector-based hierarchical IK algorithm of previous section requires to solve, via pseudo-inversion, a number k of quadratic programs of the form

$$\min_{\boldsymbol{z}} \| (\boldsymbol{J}_k \boldsymbol{P}_{k-1})\, \boldsymbol{z} - \boldsymbol{b}_k \|^2,$$

where, for all $k \geq 1$, the projected Jacobian matrix is *structurally* rank deficient. Indeed, the optimization variable $\boldsymbol{z}$ dimension is equal to n for all priorities, whereas the optimization problem at k must have solution lying inside the null-space of all higher priority tasks. This property can be exploited in order to reduce the number of optimization variable along the stack, as it is detailed hereafter. Starting again from the quadratic program of (5.6), let us parametrize its solution set in a different way, as shown by the following equation:

$$\begin{aligned}
\dot{\boldsymbol{q}}_1^* &= \boldsymbol{J}_1^\dagger \dot{\boldsymbol{x}}_1 \\
\dot{\boldsymbol{q}}_1(\boldsymbol{z}) &= \dot{\boldsymbol{q}}_1^* + \boldsymbol{N}_1 \boldsymbol{z},
\end{aligned} \tag{5.13}$$

where the columns of $\boldsymbol{N}_1$ form an orthonormal basis of the null-space of $\boldsymbol{J}_1$, denoted as $N(\boldsymbol{J}_1)$, i.e.

$$\boldsymbol{N}_1 \in \mathbb{R}^{n \times n_1} \quad \text{where } n_1 = \dim N(\boldsymbol{J}_1) \tag{5.14}$$

$$\boldsymbol{N}_1^T \boldsymbol{N}_1 = \boldsymbol{I}_{n_1} \tag{5.15}$$

$$\boldsymbol{J}_1 \boldsymbol{N}_1 = \boldsymbol{0}, \tag{5.16}$$

and $\boldsymbol{z} \in \mathbb{R}^{n_1}$ is a coordinate vector of the corresponding null-space. Note also that the null-space basis can be obtained by means of an SVD, as noted in Section 2.3.4. The quadratic

[3] In particular, the reader may notice that $\boldsymbol{P}_{k-1} (\boldsymbol{J}_k \boldsymbol{P}_{k-1})^\dagger = (\boldsymbol{J}_k \boldsymbol{P}_{k-1})^\dagger$ for any projector matrix $\boldsymbol{P}_{k-1}$.

program at $k = 2$ then becomes

$$\min_{\boldsymbol{z}} \|\boldsymbol{J}_2 \boldsymbol{N}_1 \boldsymbol{z} + \boldsymbol{J}_2 \dot{\boldsymbol{q}}_1^* - \dot{\boldsymbol{x}}_2\|^2. \tag{5.17}$$

Notice how the task *reduced Jacobian* is "slimmer" than the original Jacobian matrix, its column number being equal to the dimension of the preceding task null-space n_1, and it is potentially full rank. Analogue reasoning as in the previous section allows to obtain the following recursion:

$$\begin{aligned}
\boldsymbol{z}_k^* &= (\boldsymbol{J}_k \boldsymbol{N}_{k-1})^\dagger \left(\dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1}^*\right) \\
\dot{\boldsymbol{q}}_k^* &= \dot{\boldsymbol{q}}_{k-1}^* + \boldsymbol{N}_{k-1} \boldsymbol{z}_k^* \\
\boldsymbol{N}_k &= N(\boldsymbol{J}_k \boldsymbol{N}_{k-1}), \quad \text{e.g. from SVD} \\
\boldsymbol{N}_0 &= \boldsymbol{I}_n \\
\dot{\boldsymbol{q}}_0 &= 0.
\end{aligned} \tag{5.18}$$

Although very similar to the projector-based formulation, the null-space based saves computations by shrinking the optimization vector dimension as the k index increases. With this strategy, and even for very complex robots such as Centauro, the last priority level is typically very small, and therefore cheap to solve. Computational complexity can be further reduced using the *Complete Orthogonal Decomposition (COD)* in place of the SVD as shown in Escande et al. (2014b), at the cost of inferior robustness when computing inverses.

5.2.6 Constrained solutions

In the constrained case, the quadratic program to be solved at the k-the priority level is in the form

$$\begin{aligned}
&\min_{\dot{\boldsymbol{q}}} \|\boldsymbol{J}_k \dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_k\|^2 + \lambda \|\dot{\boldsymbol{q}}\|^2 \\
&\text{subject to} \\
&\quad \boldsymbol{c}_{\min} \leq \boldsymbol{C} \dot{\boldsymbol{q}} \leq \boldsymbol{c}_{\max} \quad \text{(joint limits, other bounds, ...)}
\end{aligned}, \tag{5.19}$$

with the additional constraint that the task functions of higher priority tasks cannot be altered. This can be done in at least three ways: (i) equality constraints can be explicitly added to the quadratic program (so called *optimality constraints*); (ii) either the projector-based approach or the null-space based approach of the previous sections can be employed, such that priorities are implicitly taken into account inside the cost function definition. If the first

strategy is chosen, the quadratic program (5.19) is changed to the one given below:

$$\begin{aligned}
&\min_{\dot{\boldsymbol{q}}} \|\boldsymbol{J}_k \dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_k\|^2 + \lambda \|\dot{\boldsymbol{q}}\|^2 \\
&\text{subject to} \\
&\quad \boldsymbol{c}_{\min} \leq \boldsymbol{C} \dot{\boldsymbol{q}} \leq \boldsymbol{c}_{\max} \\
&\quad \boldsymbol{J}_1 \dot{\boldsymbol{q}} = \boldsymbol{J}_1 \dot{\boldsymbol{q}}_1^* \\
&\quad \vdots \\
&\quad \boldsymbol{J}_{k-1} \dot{\boldsymbol{q}} = \boldsymbol{J}_{k-1} \dot{\boldsymbol{q}}_{k-1}^*
\end{aligned} \quad ; \tag{5.20}$$

this strategy relies on the QP solver to actually carry out the optimization inside the null-space of all preceding tasks, with no pre-processing step (e.g. SVD) involved. Therefore, good performance is obtained. The drawback is that the only form of regularization being applied to (5.20) is the Tikhonov term in the objective function. However, when non-trivial control problem are solved, it is quite common for optimality constraints to become *linearly dependent*; more specifically, this happens whenever a higher priority task is rank-deficient (e.g. 6-D Cartesian control of a kinematic chain with less than six dofs) or when multiple tasks are in conflict. Under such circumstances, the QP solver might experience instabilities and return unsafe joint velocities.

In order to mitigate this issue, priority enforcement must be carried out outside the solver, by employing one of the hierarchical IK strategies of Sections 5.2.4 and 5.2.5. In order to do so, one simply applies the recursive formulas of (5.12) and (5.18), replacing the pseudo-inverse based optimal solution for the $\boldsymbol{z}$ parameter with the appropriate QP, i.e.

$$\begin{aligned}
&\min_{\boldsymbol{z}} \|\boldsymbol{J}_{k|k-1} \boldsymbol{z} - \dot{\boldsymbol{x}}_{k|k-1}\|^2 + \lambda \|\dot{\boldsymbol{q}}\|^2 \\
&\text{subject to} \\
&\quad \boldsymbol{c}_{\min} \leq \boldsymbol{C} \dot{\boldsymbol{q}} \leq \boldsymbol{c}_{\max}
\end{aligned} \quad , \tag{5.21}$$

where the priority consistent Jacobian is either $\boldsymbol{J}_{k|k-1} = \boldsymbol{J}_k \boldsymbol{P}_{k-1}$ or $\boldsymbol{J}_{k|k-1} = \boldsymbol{J}_k \boldsymbol{N}_{k-1}$ depending on the chosen strategy, and $\dot{\boldsymbol{x}}_{k|k-1} = \dot{\boldsymbol{x}}_k - \boldsymbol{J}_k \dot{\boldsymbol{q}}_{k-1}^*$. In either cases, Tikhonov regularization can be replaced by a selective SVD-based regularization such as (2.45) or (2.46).

Table 5.1 Comparison between different proposed strategies for the motion control of the Centauro robot (R = roll axis, P = pitch axis).

Local frame \ Motion	M1	M2	M3	M4
Trunk frame	✓	✗	✓	✓
Horizontal trunk frame	✓	R,P	✓	✓
Virtual frame	✓	✓	✓	✓

5.3 Controller design

As mentioned in the introductory section, a complete motion controller for a hybrid wheeled-legged robot such as Centauro must provide the freedom to combine local tasks with global tasks. This rather generic statement is now detailed into a list of *motion requirements*, which different controller designs will then be compared against. According to such a list, the Centauro robot should be able to:

M1: move as a whole w.r.t. a global world frame by appropriately steering and rolling the wheels (*driving* motion);

M2: while driving, adjust the trunk pose w.r.t. a local frame (e.g. to lift one foot);

M3: while driving, reshape the support polygon by shifting the wheels position w.r.t. a local frame (e.g. to pass through a narrow passage);

M4: perform a manipulation task w.r.t. to either a local or global world frame, while shifting the support polygon using the wheels (e.g. to improve stability);

Over the course of this section, we incrementally build the proposed approach to the motion controller design, that is presented in Section 5.3.3, starting from the naive strategy of Section 5.3.1. Results of each iteration, as compared to the defined motion requirements, are summarized in Table 5.1, and a thorough discussion on the advantages given by this contribution is presented in Section 5.3.4.

For the sake of this presentation, it is assumed that the robot wheels can be steered and rolled in a way such that their slippage (i.e. the relative velocity between the contact point and the ground) is zero, whenever this is physically feasible. An algorithmic way to ensure this behavior will be presented in Section 5.4.

5.3.1 Trunk-based control

A first approach to the wheeled motion control of Centauro is to combine *relative* Cartesian tasks between the four wheels and the trunk, with another one controlling the trunk pose w.r.t. a global world frame.

Relative control between frames can be enforced at the Jacobian level as follows. First, let the label *"d"* denote the distal frame, and *"b"* the base frame; then, the relative velocity twist between the two is given by the following expression:

$$\begin{cases} \boldsymbol{v}_{\text{rel}} = \boldsymbol{v}_d - \left(\boldsymbol{v}_b + \boldsymbol{\omega}_b \times \boldsymbol{p}_d^b\right) \\ \boldsymbol{\omega}_{\text{rel}} = \boldsymbol{\omega}_d - \boldsymbol{\omega}_b \end{cases}, \tag{5.22}$$

where the cross-product term in (5.22) takes into account the motion of the distal frame as seen from the base frame. In matrix form, equation (5.22) reads

$$\dot{\boldsymbol{x}}_{\text{rel}} = \begin{bmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} & -\boldsymbol{I}_{3\times3} & \boldsymbol{S}(\boldsymbol{p}_d^b) \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} & -\boldsymbol{I}_{3\times3} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{x}}_d \\ \dot{\boldsymbol{x}}_b \end{bmatrix}, \tag{5.23}$$

where $\boldsymbol{S} \in \mathbb{R}^{3\times3}$ is the skew-symmetric matrix such that $\boldsymbol{S}(\boldsymbol{a})\boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{b} \quad \forall \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$. Introducing the distal and base Jacobians w.r.t. the world frame $\boldsymbol{J}_d$ and $\boldsymbol{J}_b$, such that $\dot{\boldsymbol{x}}_d = \boldsymbol{J}_d \dot{\boldsymbol{q}}$ and $\dot{\boldsymbol{x}}_b = \boldsymbol{J}_b \dot{\boldsymbol{q}}$, the relative Jacobian is given by

$$\boldsymbol{J}_{\text{rel}} = \begin{bmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} & -\boldsymbol{I}_{3\times3} & \boldsymbol{S}(\boldsymbol{p}_d^b) \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} & -\boldsymbol{I}_{3\times3} \end{bmatrix} \begin{bmatrix} \boldsymbol{J}_d \\ \boldsymbol{J}_b \end{bmatrix}. \tag{5.24}$$

According to the *math-of-tasks* formalism defined in (Mingo Hoffman et al., 2017), a possible stack-of-task implementing this solution is the following:

$$\begin{pmatrix} \left(\sum_i {}^{\text{Trunk}}\mathscr{T}^{[\text{XYZ}]}_{\text{Wheel}_i} + {}^{\text{World}}\mathscr{T}_{\text{Trunk}}\right) / \\ \left({}^{[-]}\mathscr{T}_{\text{Hands}} + \sum_i {}^{\text{World}}\mathscr{T}^{[\text{RPY}]}_{\text{Ankle}_i}\right) / \\ \left(\mathscr{T}_{\text{Posture}}\right) \end{pmatrix} << \left(\mathscr{C}_{\substack{\text{Pos.}\\\text{Lims}}} + \mathscr{C}_{\substack{\text{Vel.}\\\text{Lims}}}\right); \tag{5.25}$$

in the previous equation, ${}^{\text{A}}\mathscr{T}_{\text{B}}$ denotes a Cartesian task of the frame B relative to the frame A. The base frame for end-effectors is left unspecified, as it is dynamically changed from *trunk* to *world* depending on the task. Finally, the plus operator "+" indicates *aggregation* between tasks, whereas the slash "/" sets the left hand side task at a *higher priority* w.r.t. the right hand side task, while the $<<$ symbol is used to specify constraints.

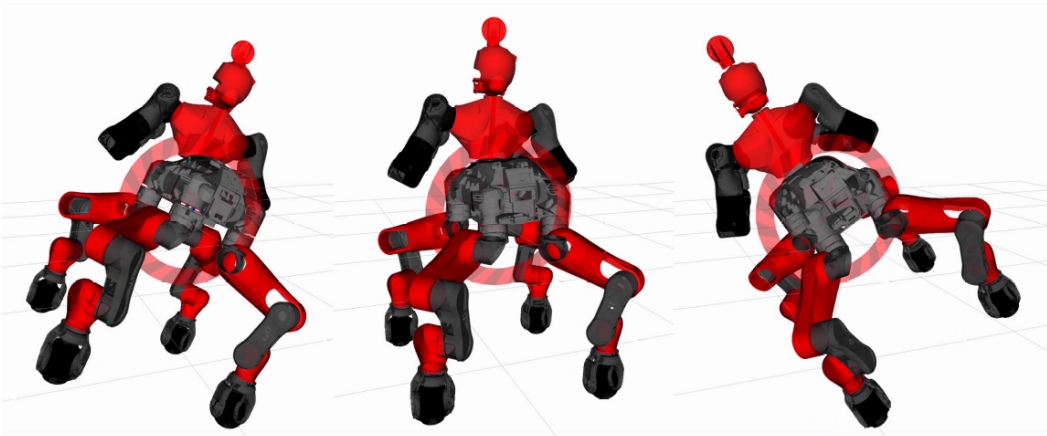


Figure 5.1 Kinematic behavior of the Centauro robot while a rolling motion is commanded to it base. The Cartesian pose of the wheels is controlled w.r.t. to the trunk frame (*trunk-based control)*. The obtained motion is not physically feasible.

With such a stack, and also assuming that some controller continuously steers and spins each wheel so that it does not slip, the robot base can be conveniently moved around the world frame by just setting its desired Cartesian velocity or pose. Moreover, the robot support polygon can be reshaped by simply setting appropriate desired poses of the wheels w.r.t. the base. However, this scheme cannot handle any roll or pitch motion commanded to the base; indeed, when the base is commanded e.g. to roll as depicted in Figure 5.1, the imposed relative task causes the wheels to follow such a motion, leading to wheel-ground contact break. Under such circumstance, we obtain a physically unfeasible robot motion.

5.3.2 Horizontal frame-based control

In order to adapt the scheme of Section 5.3.1 so that it can handle rolling and pitching of the base without breaking contact with the ground, there is the need to introduce a special frame of reference, which we call *horizontal trunk frame (HF)*, with the following three properties:

1. the horizontal trunk frame origin coincides with the trunk frame origin;
2. the projection on the horizontal plane of the trunk frame forward axis (x-axis) coincides with the projection of the horizontal trunk frame forward axis;
3. the horizontal trunk frame vertical axis (z-axis) coincides with the world frame vertical axis.

This idea is inspired by (Barasuol et al., 2013), in which a similar frame was adopted in the context of quadrupedal trotting. The reason why such a frame is a good candidate to act as the robot *local frame* lies in the fact that its pose matches the trunk pose for what concerns its position and heading, while being unaffected by rolling or pitching of the base. Thanks to this property, using the horizontal trunk frame as base frame for the wheels tasks can solve the contact-braking issue of the trunk-based approach; as it is shown in Figure 5.2, when the base is commanded to follow the same rolling motion as in Section 5.3.1, the horizontal frame does not move, and the wheel-ground contact is retained.

In order to implement the required Cartesian task w.r.t. the horizontal trunk frame, it is enough to compute its Jacobian, and then apply (5.24). In order to do so, let us observe that such a frame coincides with the trunk frame, except that it does not move about its roll and pitch axes. Consequently, its jacobian $\boldsymbol{J}_h$ is given by the following expression:

$$\boldsymbol{J}_h = \mathrm{diag}(1,1,0,0,0,1)\,\boldsymbol{J}_{\text{trunk}}. \tag{5.26}$$

A possible stack of task leveraging the horizontal frame is given in the following equation:

$$\begin{pmatrix} \left(\sum_i {}^{\text{HF}}\mathscr{T}^{[\text{XYZ}]}_{\text{Wheel}_i} + {}^{\text{World}}\mathscr{T}^{[\text{XY,Yaw}]}_{\text{HF}}\right) / \\ \left({}^{\text{HF}}\mathscr{T}^{[\text{Z,RP}]}_{\text{Trunk}} + {}^{[\text{-}]}\mathscr{T}_{\text{Hands}} + \sum_i {}^{\text{World}}\mathscr{T}^{[\text{RPY}]}_{\text{Ankle}_i}\right) / \\ \left(\mathscr{T}_{\text{Posture}}\right) \end{pmatrix} << \left(\mathscr{C}_{\substack{\text{Pos.}\\ \text{Lims}}} + \mathscr{C}_{\substack{\text{Vel.}\\ \text{Lims}}}\right). \tag{5.27}$$

5.3.3 Virtual local frame

As observed in Section 5.3.2, the Cartesian controller based on the horizontal trunk frame manages to improve the motion capabilities, by introducing a base frame for the wheels that is *partially decoupled* from the trunk link. However, it is unable to achieve local control of the trunk along the coupled directions, namely x, y, and yaw axes. Such commands result in the whole robot rolling in the given direction w.r.t. the world frame, whereas our desired outcome is a local adjustment of the trunk.

In order to achieve the complete decoupling between the local frame and the trunk frame, this work proposes to inject additional degrees of freedom in the robot model, between the robot trunk and a newly introduced *virtual frame (VF)* which, intuitively speaking, can be interpreted as an additional world frame which *travels with the robot*. The configuration

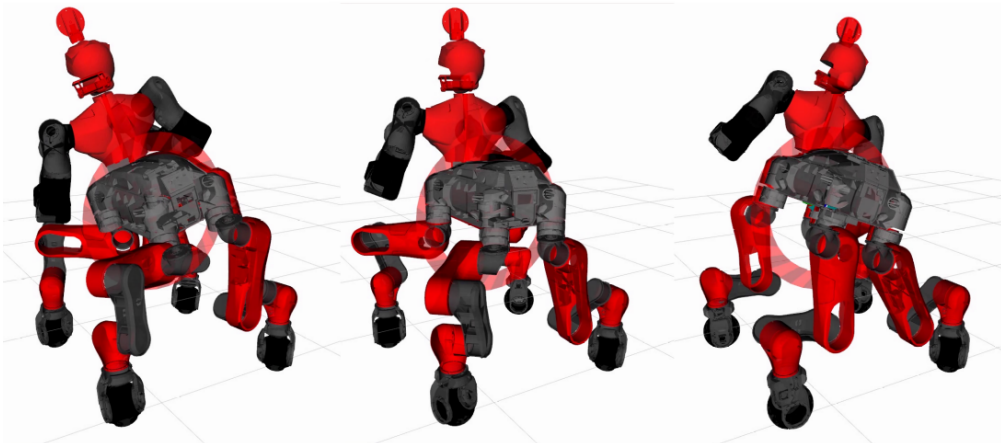


Figure 5.2 Kinematic behavior of the Centauro robot while a rolling motion is commanded to it base. The Cartesian pose of the wheels is controlled w.r.t. to a horizontal frame that is attached to the trunk (*horizontal frame-based control).*

vector for the Centauro robot is thus changed w.r.t. (5.1), as follows:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_{\text{fb}} \\ \boldsymbol{q}_{\text{v}} \\ \boldsymbol{q}_{a} \end{bmatrix}, \tag{5.28}$$

with $\boldsymbol{q}_{\text{v}} \in \mathbb{R}^6$ being a minimal representation of the virtual frame pose w.r.t. the trunk frame. A pictorial representation of the resulting kinematic model is given in Figure 5.3.

This virtual frame of reference can then be used as base frame for both the wheels and the trunk tasks. Indeed, having introduced six additional degrees-of-freedom between the trunk and the virtual frame (red chain in Figure 5.3), full local control of the robot waist can be achieved, whereas the global motion of the robot is obtained by means of a Cartesian task for the virtual frame w.r.t. the world, as described by the following stack of tasks:

$$\begin{pmatrix} \left(\sum_i {}^{\text{VF}}\mathscr{T}^{[\text{XYZ}]}_{\text{Wheel}_i} + {}^{\text{World}}\mathscr{T}_{\text{VF}}\right) / \\ \left({}^{\text{VF}}\mathscr{T}_{\text{Trunk}} +^{[-]} \mathscr{T}_{\text{Hands}} + \sum_i {}^{\text{World}}\mathscr{T}^{[\text{RPY}]}_{\text{Ankle}_i}\right) / \\ \left(\mathscr{T}_{\text{Posture}}\right) \end{pmatrix} << \left(\mathscr{C}_{\substack{\text{Pos.}\\ \text{Lims}}} + \mathscr{C}_{\substack{\text{Vel.}\\ \text{Lims}}}\right). \tag{5.29}$$

Interestingly, virtual frames have been exploited by the robotic grasping community in the context of object manipulation, as e.g. in (Dehio et al., 2018; Tahara et al., 2010; Wang et al., 2015). Similarly to our case, such a description conveniently provides an additional link which is not rigidly attached to any of the robot frames.

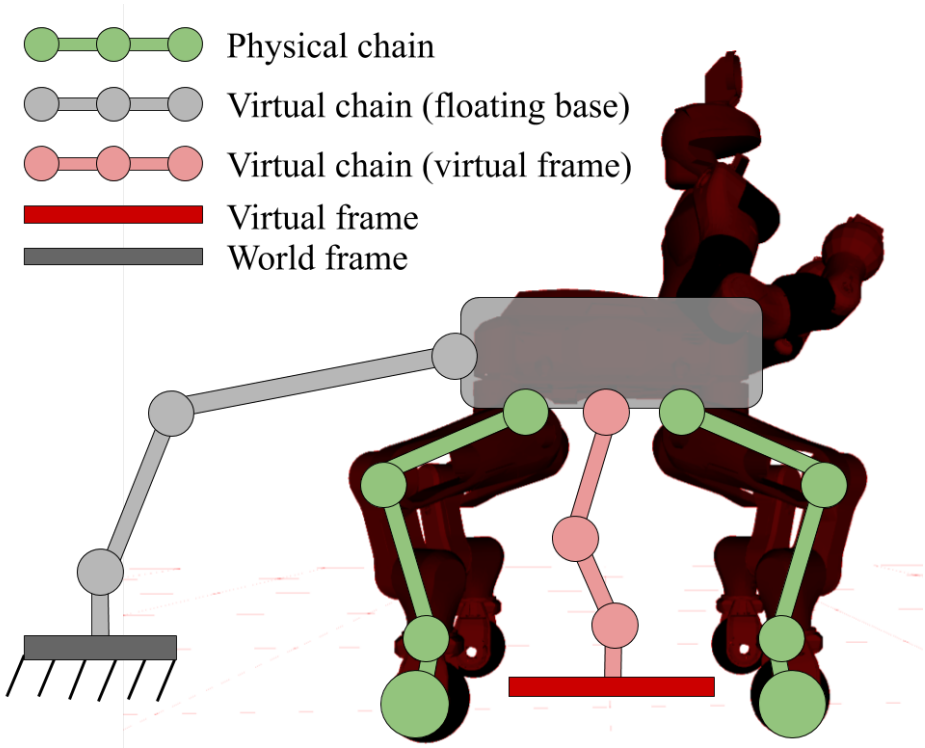


Figure 5.3 Augmented kinematic model for a wheeled-legged quadruped. A virtual local frame (dark red) is introduced in order to achieve full local control of the trunk frame. An additional virtual chain (light red) is used to connect such a frame to the robot, so that full decoupling can be obtained.

5.3.4 Discussion

As it has been shown in Section 5.3.3, full local control of the trunk can be achieved by augmenting the kinematic model with an additional virtual chain. A controller based on such formulation can fully exploit the platform flexibility, by adding the ability to shift the trunk w.r.t. the support polygon about all axes. Furthermore, having defined a local frame which is decoupled from the robot trunk enables taking full advantage of our whole-body prioritized control framework. By relaxing the local trunk task ${}^{\mathrm{VF}}\mathscr{T}_{\mathrm{Trunk}}$ (i.e. putting it at low priority, or removing some of its degrees of freedom from the IK problem), the solver can adapt its posture to accommodate for more extreme desired poses of the wheels or of the end-effectors; in other words, local trunk adjustments are managed automatically by the solver. For instance, whenever a wheel (or a hand) is commanded to a local pose which exceeds its own workspace, an adaptation of the trunk pose will be required. Because the first two strategies employ the trunk itself as local frame, they are unable to provide such an adaptation; on the other hand, the virtual-frame based approach has enough degrees of freedom between the local frame and the trunk frame to provide the required trunk adaptation and accomplish the task, as it will be exemplified in Section 5.5 (see Figure 5.8). Having adopted a whole-body floating base formulation, we gained the ability to freely mix local adjustments with global control, by setting the base frame of each task to be the virtual frame or the global world, respectively. Notice how chain based formulations, as e.g. the IK used in Schwarz et al. (2016), need to explicitly reason about desired poses for wheels and end-effectors w.r.t. the trunk frame in order to achieve some task that is natively defined w.r.t. the world frame; for instance, a pitching motion for the base can be obtained by changing the length of front and rear legs. Such poses must be tracked accurately (high λ as in (**??**)), and activation of a constraint on one chain is not taken into account on other chains, thus hindering automatic adaptations as in Figure 5.8. On the contrary, the designed controller allows the user to directly specify the desired task in its native coordinate system, while the solver takes care of the correct relationships between all relevant frames.

5.4 Pure rolling condition

For the sake of simplicity, previous discussion has neglected the problem of guaranteeing the pure rolling of each wheel on the ground surface. However, such matter is of paramount importance, since the planned motion can be accurately transferred to the hardware only if the relevant *contact conditions* are not violated. In the case of a wheeled robot, a zero-slippage

condition needs to be ensured, which means that the contact point of each wheel must have zero velocity w.r.t. the ground:

$$\boldsymbol{v}_C = \boldsymbol{J}_C \dot{\boldsymbol{q}} = 0, \tag{5.30}$$

where $\boldsymbol{J}_C \in \mathbb{R}^{3\times n}$ is the *contact Jacobian*, i.e. the Jacobian of a point $\boldsymbol{p}_C$ which instantaneously moves with the wheel, but is always located at the contact point. Letting $\boldsymbol{p}_w$ denote the center of the wheel, R the wheel radius, and $\boldsymbol{n}_C$ the outward normal to the contact surface, then such a point is given by the following equation:

$$\boldsymbol{p}_C = \boldsymbol{p}_w - R\boldsymbol{n}_C. \tag{5.31}$$

5.4.1 Steering control

To gain further insight on how to effectively enforce the pure rolling constraint, we single out the contribution of the wheel joints, as in the equation below:

$$\boldsymbol{v}_C = \boldsymbol{v}_w + R\dot{q}_w \boldsymbol{i}_a, \tag{5.32}$$

where all quantities are conveniently expressed w.r.t. the local frame, such that they are time-invariant for a constant motion in local coordinates. The meaning of the symbols in (5.32) is as follows: $\dot{q}_w$ is the angular velocity of the wheel about its spinning axis, $\boldsymbol{v}_w$ represents the absolute velocity of the wheel's center, and $\boldsymbol{i}_a$ is the direction of the ankle frame x-axis. Such a frame is defined as depicted in Figure 5.4: the x-axis points along the wheel forward direction, the z-axis coincides with the steering joint axis, and the y-axis completes the right-handed frame. Furthermore, $\boldsymbol{v}_w$ is assumed to be constant.

With the aim to control the contact velocity to zero, we first compute its variation as follows:

$$\dot{\boldsymbol{v}}_C = R\ddot{q}_w \boldsymbol{i}_a + R\dot{q}_w(\boldsymbol{\omega}_a \times \boldsymbol{i}_a) + \cancelto{0}{\dot{\boldsymbol{v}}_w}; \tag{5.33}$$

moreover, since the ankle frame angular velocity $\boldsymbol{\omega}_a$ is given by the steering joint rotation, we set $\boldsymbol{\omega}_a = \dot{q}_s \boldsymbol{k}_a$, with $\dot{q}_s$ denoting the steering joint velocity, and $\boldsymbol{k}_a$ being the direction of the steering axis; this yields

$$\dot{\boldsymbol{v}}_C = R\ddot{q}_w \boldsymbol{i}_a. + R\dot{q}_w \dot{q}_s \boldsymbol{j}_a. \tag{5.34}$$

It can be noticed that the rate of change of contact velocity is given by two terms: a forward acceleration component in the direction $\boldsymbol{i}_a$, which can be controlled to zero by appropriately spinning the wheel joint, and a lateral component in the direction $\boldsymbol{j}_a$ which is also influenced by the steering joint speed. In principle, the lateral contact velocity can be made to vanish by

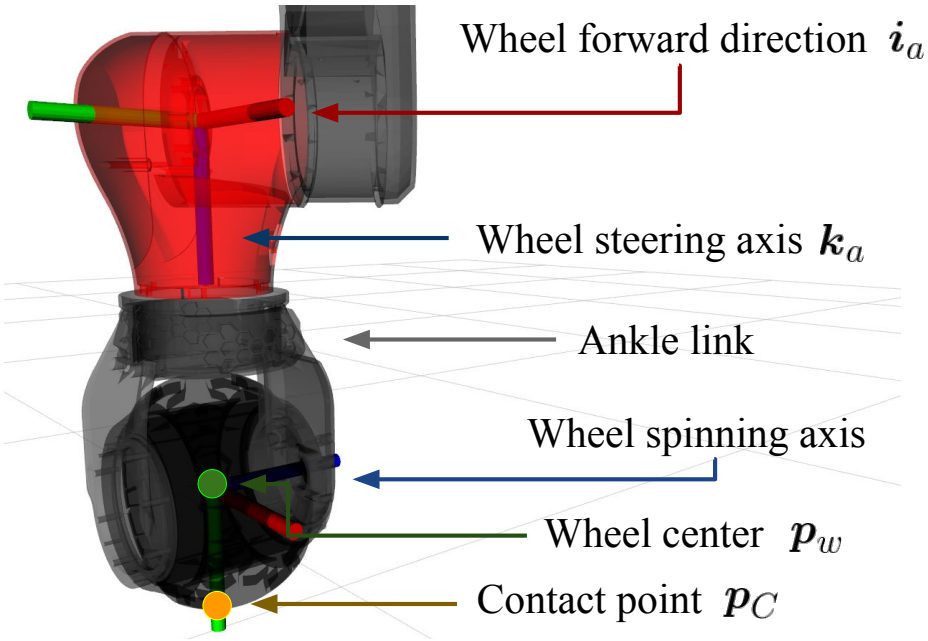


Figure 5.4 Structure and reference frames for the Centauro wheel complex.

enforcing the following first order dynamics

$$\boldsymbol{j}_a \cdot [\dot{\boldsymbol{v}}_C + k_C \boldsymbol{v}_C] = 0, \quad k_C > 0, \tag{5.35}$$

which can be done with a suitable choice of the steering joint velocity. Plugging (5.34) in (5.35) and solving for $\dot{q}_s$ gives rise to the steering control law given by the following expression[4]:

$$\dot{q}_s = k_C \frac{\boldsymbol{j}_a \cdot \boldsymbol{v}_C}{R\,\dot{q}_w}, \tag{5.36}$$

where the scalar gain k_C controls the speed of convergence. Finally, note that the forward component is continuously canceled by the contact task (5.30).

5.4.2 Dealing with joint limits

Under the assumption that the steering joint can spin continuously, (5.36) provides a feasible solution to the wheel steering problem. However, this is not the case for our Centauro robot: its steering motors are indeed characterized by hard stops, that prevent the cables connecting the wheel to the robot from excessive twisting. Equation (5.36) is a *local* law, in the sense

[4]Equation (5.36) can be regularized around $\dot{q}_w = 0$ by replacing $\frac{1}{\dot{q}_w}$ with a term such as $\frac{\dot{q}_w}{\dot{q}_w^2+\varepsilon}$ for some $\varepsilon > 0$.

that it has no knowledge about whether the commanded motion will eventually lead to a constraint violation. For this reason, the presence of joint limits demands a different approach to be followed.

More specifically, let us note that in order for $\boldsymbol{v}_C$ to be zero, a necessary condition is that velocity of the wheel center be parallel to the wheel forward direction, so that the two terms in (5.32) can cancel out. Clearly, such a condition admits two solutions, which can be obtained by adding or subtracting 180 degrees. Steering angle candidates can be computed by the following argument: we first consider the angle ϑ between $\boldsymbol{v}_C$ expressed in the ankle frame and the direction $\boldsymbol{i}_a$; such an angle represents a *steering error* and, as such, can be added to the current steering angle in order to obtain the correct ones:

$$\begin{cases} q_s^{(1)} = q_s + \vartheta \\ q_s^{(2)} = \mathrm{wrap}_{[-\pi,\pi]}\left(q_s^{(1)} + \pi\right) \end{cases}, \tag{5.37}$$

If the steering joint range spans more than 180 degrees, then at least one between $q_s^{(1)}$ and $q_s^{(2)}$ will not violate the limits. Finally, in order to obtain an as smooth trajectory as possible for the steering joint, whenever both solutions are valid, the one that is the closest to the current value is eventually selected.

5.4.3 Integration into a stack of tasks

To enforce the pure rolling condition into a stack of tasks, two tasks are defined, $\mathscr{T}_{\text{Steering}}$ and $\mathscr{T}_{\text{Rolling}}$, to separately handle steering (5.37) and slippage control (5.30). Such tasks must be integrated into the Cartesian control problems (5.25), (5.27), and (5.29) by placing them at the appropriate priority level. To this aim, we observe that the rolling task must have lower priority than the wheel position task ${}^{\text{VF}}\mathscr{T}_{\text{Wheel}}^{[\text{XYZ}]}$, so that the wheel can move despite the steering angle not being numerically equal to its optimal value; hence, one possibility is to place it at the second priority level. Conversely, steering tasks should be placed at highest priority, so that the steering angle is not affected by the ankle orientation task ${}^{\text{World}}\mathscr{T}_{\text{Ankle}}^{[\text{RPY}]}$.

5.5 Experiments

The proposed control algorithm was implemented inside the *CartesI/O* framework (Laurenzi et al., 2019c), that is a ROS-based library for online Cartesian control with real-time (RT) support. Under the hood of CartesI/O, the *OpenSoT* library (Mingo Hoffman et al.,

2017) implements the math of tasks in the C++ language through operator overloading; the corresponding stack-of-tasks is then setup and solved by OpenSoT in a RT-safe way, leveraging on off-the-shelf QP solvers. More specifically, a nullspace-based constrained IK scheme has been selected for this implementation, since it provides a good balance between computational requirements and stability of the solution. The Centauro robot is powered by the *XBotCore* middleware (Muratore et al., 2017b), which allows for mixed RT (through the development of *real-time plugins*) and non-RT control (via ROS integration). Because the whole framework is parametrized in terms of a standard *URDF*-based description of the robot, implementation of the virtual-frame concept only required to provide an augmented URDF model with appropriate additional joints and links.

To validate the proposed control scheme against the given motion requirements (M1) – (M4), and most notably in terms of transferability to the hardware, an extensive set of experiments has been designed and performed. The stack of tasks employed for the experimental sessions is based on (5.29), adding further tasks for steering and slippage control, as discussed in Section 5.4.3. Solving the cascaded QPs of (5.21), plus the computation of nullspace bases, required on average $\bar{t}_{\text{cpu}} = 3.4$ ms, with a standard deviation of $\sigma_{\text{cpu}} = 0.17$ ms. The reader is encouraged to check out the accompanying video, which is also available at https://youtu.be/uIaAGrhMbuY.

A first experiment consists of a single run where the robot performs a driving motion (M1) while simultaneously adjusting the support polygon (see Figure 5.5) from a starting squared shape, to a narrower configuration, and then to a wider one (M2). Afterwards, local

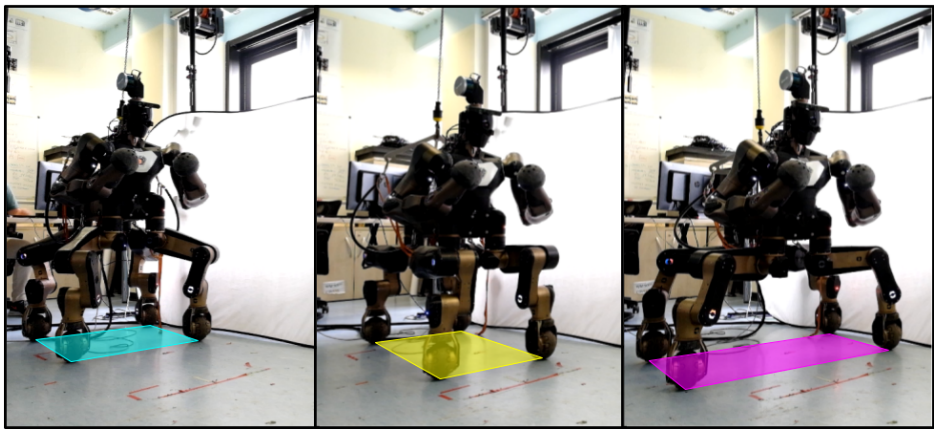


Figure 5.5 Support polygon shape modulation while performing a driving motion with forward speed of $v = 0.05$ ms^{-1}.

adjustments while driving (Figure 5.6) are validated with a sequence of lateral, sagittal, and rotation adjustments (M3). Finally, the robot stops to perform a simulated manipulation task

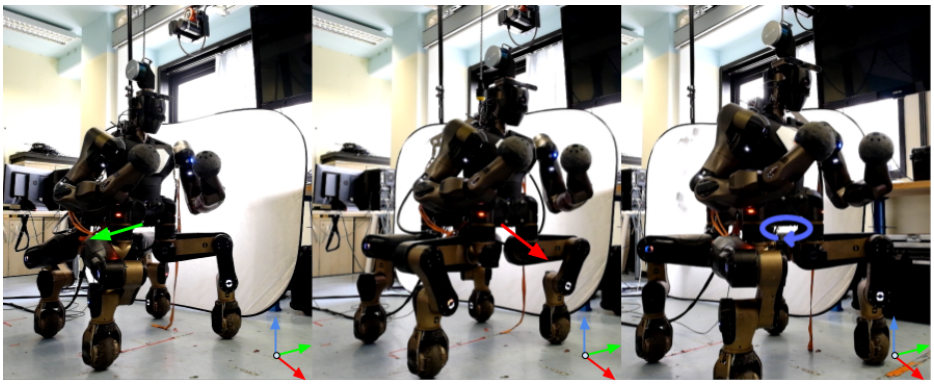


Figure 5.6 Trunk motion w.r.t. the virtual frame in right, forward, and clockwise direction.

in world coordinates (Figure 5.7), which consists in reaching a high position; the trunk task ${}^{\mathrm{VF}}\mathscr{T}_{\mathrm{Trunk}}$ is deactivated in order for the trunk to adapt to the end-effectors desired pose. Then, while the end-effectors are kept fixed w.r.t. the global world frame, the support polygon is adjusted by sending suitable references to the virtual frame (M4).

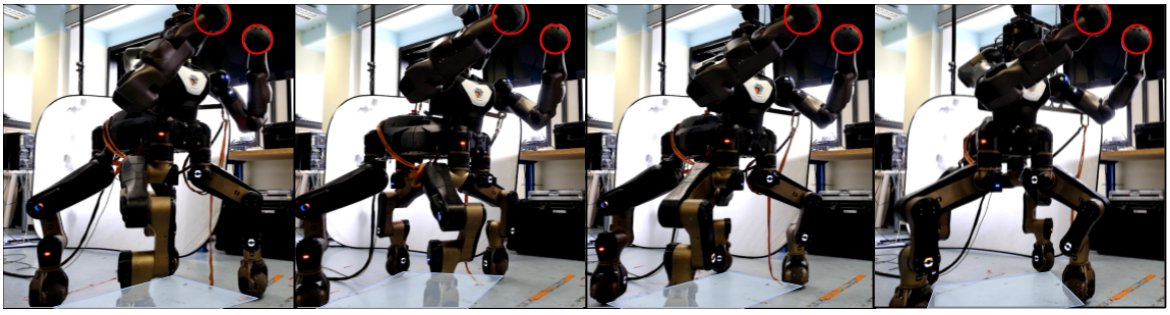


Figure 5.7 Manipulation in world frame coordinates with simultaneous support polygon (SP) adjustment. First, the SP is shifted backwards; then, it is rotated clockwise; finally, it is rotated counterclockwise. The trunk posture is automatically adapted by the solver.

In order to validate the ability of the proposed controller to deal with constraint activation in a whole-body fashion, we command the rear-right wheel to lift from the ground (Figure 5.8), after having adjusted the trunk position in order to avoid the robot to fall. The commanded position cannot be reached by only moving the rear-right leg, due to the hip pitch joint reaching its hard stop. In such a scenario, where chain-based solvers would be bound to fail, our strategy is able to automatically adapt the trunk position (which is set to have lower priority than the wheel) in order to complete the task. A quantitative assessment is given in Figure 5.9, where the virtual frame based scheme achieves close to zero wheel position error at the apex of the commanded trajectory (mid point of the red shaded area). Notice how, because end-effectors are being controlled w.r.t. the local frame, the arms compensate for

the trunk motion in order to keep the end-effectors in position. This further highlights the benefits of a whole-body approach.

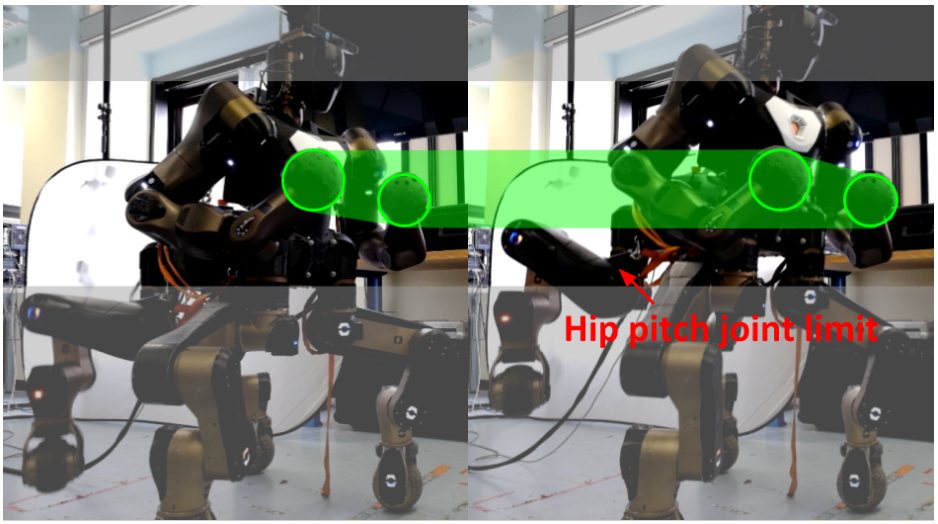


Figure 5.8 Trunk adaptation in the presence of constraints. Gray shaded areas highlight the upward motion of the trunk, whereas end-effectors keep their position as emphasized by the green shaded areas.

The effectiveness of the proposed steering strategy is assessed in Figure 5.10, which shows a time history of each wheel slippage velocity, i.e. the relative velocity between each wheel contact point the the ground. It can be noticed how such values are moderately close to zero, resulting in a good fulfillment of the desired contact condition. Short spikes in the slippage profile indicate fast steering maneuvers to align each wheel to the correct direction given by (5.37). Longer spikes are instead due to the switching between the two solutions in (5.37), that is necessary due to hard stops in the steering joints. Nevertheless, our simple steering strategy is sufficient for a smooth transition from a kinematic model to the real hardware, as it can be assessed in the accompanying video.

5.6 Discussion

The present chapter has introduced a novel methodology for the Cartesian control of hybrid wheeled-legged robots. A comparison with other approaches has been discussed, leading to two conclusions; first, a local frame of reference for the robot is to be selected with care; indeed, the trivial solution does not permit to achieve the complete control of the platform, as given by the set of motion requirements (M1) – (M4). Second, only by adding further virtual degrees of freedom to the kinematic model it is possible to achieve a fully decoupled

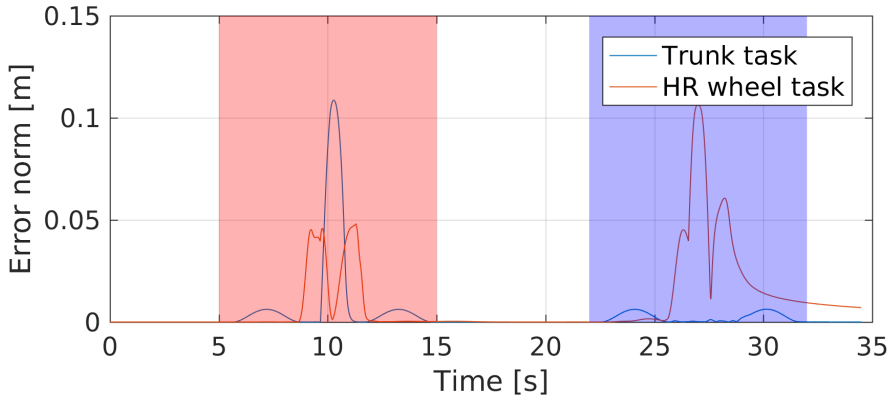


Figure 5.9 Wheel lifting under the proposed stack of tasks (red area), compared to the same task when the local frame coincides with the trunk frame (blue area).

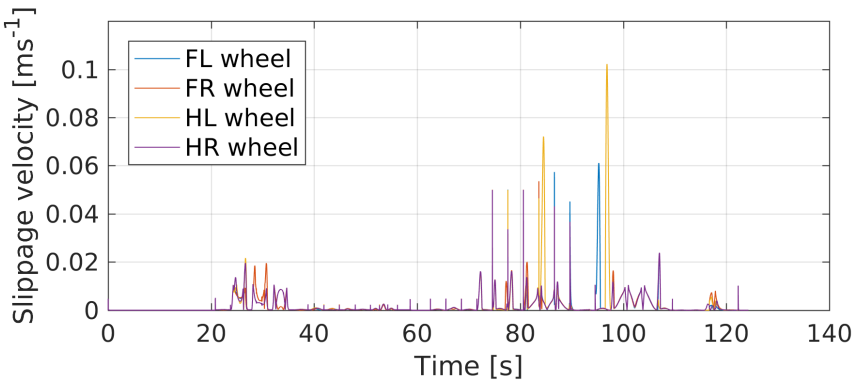


Figure 5.10 Time history of the absolute velocity of the wheels contact points w.r.t. the ground, computed along the commanded motion sequence.

control of the trunk frame w.r.t. to the local frame. Such a choice allows to mix tasks that are naturally defined in local coordinated with tasks that are defined in global coordinates, while taking full advantage of a whole-body floating base formulation. Finally, steering strategies have been proposed and analyzed, and a thorough experimental validation has been carried out.

Future work will address the application of the proposed method to a trunk stabilizer with terrain adaptation capabilities, which will enable the Centauro robot to locomote on non-flat surfaces. Moreover, the application of the proposed augmented model to dynamics-based controllers will also be subject of future research.

Chapter 6

Legged locomotion

The present chapter addresses the generation of a walking gait with automatic footstep placement for a quadrupedal robot, within a *Linear Model Predictive Control* framework. Existing work has shown how this is only possible within a non-convex programming framework, finding a solution of which is well-known to be very hard. The chapter contributes with a way to formulate the joint optimization problem as an approximate QP with linear constraints, whose global optimum can be quickly found with off-the-shelf solvers. More specifically, this is done by introducing auxiliary states and control inputs, each of which is subject to linear constraints that are inspired from the literature on bipedal locomotion. Finally, the introduced method is validated on the real Centauro robot.

This chapter is based on the following article:
Laurenzi, A., Hoffman, E. M., and Tsagarakis, N. G. (2018b). Quadrupedal walking motion and footstep placement through linear model predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2267–2273. IEEE.

6.1 Introduction

Mobile robots are nowadays expected to have an increasingly important role in many domains, from industrial automation and logistics to maintenance and search and rescue applications leaving the flat factory floors to enter less structured and controlled environment. To operate efficiently in these more challenging environments they should be able to safely move around regardless of how cluttered, or inaccurately known is the terrain. Walking robots have the potential to perform locomotion through arbitrarily complex terrains, at the cost of an increased mechanical *and* control complexity related to stability and body coordination issues.

This chapter addresses the problem of generating an *omni-directional walking gait* for a quadrupedal robot, i.e. a coordinated motion of the robot legs satisfying the property that at least three legs must always be on the ground. When tackling such a problem, it is important to notice that legged robots are *floating base* systems, whose global motion can only be obtained by means of *contact forces* exchanged with the environment. In turn, contact forces must fulfill *physical constraints*, and consequently there exist motions that *cannot* be executed by a floating base robot, as also discussed in Chapter 2.4.4. Simplified models have been proposed in the literature to describe the set of feasible motions in a way that is more suitable for the development of simple and fast planning algorithms, such as the *linear inverted pendulum model (LIPM)* (Kajita et al., 2003); this simple model forms the basis of many popular walking controllers.

The generation of a walking gait can be decomposed as the series of a footstep planning stage followed by a center-of-mass (CoM) motion planning. This strategy is common among the earliest approaches to legged locomotion, but it was shown (Diedam et al., 2008) that, for the case of bipeds, it is also possible to *jointly* generate both footsteps *and* CoM motion inside a QP framework, gaining improved robustness and disturbance rejection capabilities (Herdt et al., 2010a). On the contrary, in the case of quadrupedal walking, joint optimization over both CoM motion and footsteps gives rise to *non-linear constraints*, which make the optimization problem more difficult and less efficient to solve.

The main contribution of this work is a decomposition of such a joint optimization problem that does not introduce non-linear constraints, by introducing *auxiliary states and control inputs* that are subject to linear constraints. In this way, we formulate an *approximate* optimization problem that can be *exactly* solved. The proposed approach is validated on our Centauro robot, both in simulation and on real hardware.

The remainder of this chapter is organized as follows:

- Section 6.2 presents a selection of relevant works in the field;
- Section 6.3 introduces the mathematical formulation that the presented algorithm is based on, starting from existing models, and then presenting the decomposition that is the core contribution of the present work;
- Section 6.4 contains the details of the proposed implementation;
- Section 6.5 shows real hardware results;
- in Section 6.6 we summarize the outcome of this work and present possible future directions.

6.2 Related works

Legged locomotion is a rather mature field of research; the solutions that have been proposed can be roughly split into three categories: algorithms that are based on optimal control, *hybrid zero dynamics (HZD)* formulations (Sreenath et al., 2011), and algorithms that are based on bio-inspired oscillators (Wu et al., 2009). Optimal control methods have demonstrated particularly impressive results, allowing for automatic gait discovery in complex environments (Neunert et al., 2017; Winkler et al., 2018), albeit with such computational requirements that often make them unsuitable for online applications. The rest of this section focuses on relevant works that employ a similar strategy as ours, which is based on optimal control, and with sufficiently low computational burden to allow for online operation.

Such a family of walking gait generators is mainly due to the work of Kajita et al. (2003) for bipeds, who tackled the CoM motion generation problem *within feasibility constraints* as a servo tracking problem, where the system dynamics is given as an integrator, and the tracked output is the *Zero Moment Point (ZMP)*, that corresponds to the center of pressure of the ground reaction forces. Inspired by the observation that humans start moving their CoM *before* taking a step, Kajita formulated an LQR problem *with preview* of the future ZMP reference. It was later recognized (Wieber, 2006c) that to construct a hand-tuned preview on the ZMP trajectory is actually not needed, provided that the ZMP is *constrained* inside future support polygons, which is possible inside a *linear MPC (LMPC)* framework. As final steps in the development of LMPC-based bipedal walking, in the works of Diedam et al. (2008); Herdt et al. (2010a), the footsteps, CoM motion, and waist orientation (Herdt et al., 2010b) were jointly optimized in a single QP problem. Stability analyses of the LMPC *with ZMP constraints* can be found in the articles of Wieber (2006c, 2008), showing that, for the limited control horizon case, stability crucially depends on the control horizon time span, with the requirement that some CoM derivative is minimized inside the cost function. The concept of *capture point* was also introduced to identify states that *can be stabilized* without taking a step (Koolen et al., 2012). Lanari et al. (2014); Scianca et al. (2016) contributed to this topic as well, by relating the capture point idea to a *boundedness constraint*, which describes the initial CoM state and footsteps that permit to generate a bounded CoM motion. Moreover, approaches to bipedal walking have been proposed that control the ZMP according to a feedback law on the capture point (Englsberger et al., 2011).

Concerning quadrupedal walking, McGhee and Frank (1968) studied the stability properties of the different *walking gait patterns*, i.e. the order according to which the four legs are lifted, while Ma et al. (2005) suggested a simple way of selecting a specific pattern according

to the desired speed. In the works of Buchli et al. (2009); Pongas et al. (2007), heuristic approaches that rely on geometric reasoning to generate *static* walking are proposed. An optimal control approach with pre-set footholds can be found in (Winkler et al., 2015), while in (Winkler et al., 2017) footsteps are optimized as well inside a *non-linear programming* framework, whose solution is, in general, extremely hard to find.

This work addresses this specific point by reformulating the joint CoM-footsteps optimization problem as an approximate QP whose solution can be readily found, as explained in the next section.

6.3 Simplified models

As it was mentioned in the introductory section, and in Chapter 1 as well, the main difficulty that the walking gait designer must face derives from the fact that legged robots are *underactuated*: global motion cannot be directly achieved by their actuated degrees of freedom; instead, it must be generated by contact forces exchanged with the environment. This intuition is beautifully summarized by the following *centroidal dynamics* equation [1]

$$\begin{aligned} m\ddot{\boldsymbol{p}}_{\text{com}} &= \sum_{i=1}^{N} \boldsymbol{f}_i + m\boldsymbol{g} \\ \dot{\boldsymbol{L}} &= \sum_{i=1}^{N} (\boldsymbol{p}_i - \boldsymbol{p}_{\text{com}}) \times \boldsymbol{f}_i, \end{aligned} \tag{6.1}$$

where m is the system mass, $\boldsymbol{p}_{\text{com}} \in \mathbb{R}^3$ is the robot CoM position, $\boldsymbol{f}_i \in \mathbb{R}^3$ is the i-th contact force, N is the number of contacts, $\boldsymbol{g} \in \mathbb{R}^3$ is the gravity acceleration, $\boldsymbol{L} \in \mathbb{R}^3$ is the robot angular momentum, and $\boldsymbol{p}_i \in \mathbb{R}^3$ is the i-th contact point. It is remarkably important to notice that these contact forces are *constrained*, and consequently there exist CoM trajectories that cannot be executed by a legged robot. The most important constraint is commonly recognized (Wieber, 2008) as the *unilateral constraint*, which takes the following form:

$$\boldsymbol{n}_i^T \boldsymbol{f}_i \geq 0 \quad \forall i \in \{1, \ldots, N\} \tag{6.2}$$

where $\boldsymbol{n}_i \in \mathbb{R}^3$ is the outward normal of the i-th contact surface. Broadly speaking, this means that the robot can only *push* on the ground. Assuming coplanar contacts (and, for simplicity, $\boldsymbol{n} = [0\,0\,1]^T$) and rearranging equations (6.1) and (6.2) as in (Kajita et al., 2014),

[1] Notice that we neglect any *torque* exchanged with the ground, which is equivalent to assuming point contacts.

the equivalent centroidal momentum constraint can be obtained as follows:

$$\begin{aligned} &\boldsymbol{z} \in \text{ConvHull}\{\boldsymbol{p}_i\}_{i=1}^N \\ &\boldsymbol{z} = \left[\boldsymbol{p}_{\text{com}} - \frac{h}{g+\ddot{h}}\ddot{\boldsymbol{p}}_{\text{com}} + \frac{\boldsymbol{n}\times\dot{\boldsymbol{L}}}{mg+m\ddot{h}}\right]_{x,y}, \end{aligned} \tag{6.3}$$

where $\boldsymbol{z} \in \mathbb{R}^2$ is commonly referred to as the *Zero Moment Point (ZMP)*. Neglecting variations in the robot CoM height h and angular momentum, (6.3) gives rise to the popular *cart-table* model (Kajita et al., 2003):

$$\begin{aligned} &\boldsymbol{z} \in \text{ConvHull}\{\boldsymbol{p}_i\}_{i=1}^N \\ &\boldsymbol{z} = \left[\boldsymbol{p}_{\text{com}} - \frac{\ddot{\boldsymbol{p}}_{\text{com}}}{\omega^2}\right]_{x,y}, \end{aligned} \tag{6.4}$$

with $\omega = \sqrt{\frac{g}{h}}$ representing a parameter that characterizes the influence of the CoM acceleration on the ZMP position. Notice how, according to such a simplified model, the feasibility of a CoM trajectory only depends on whether a *linear combination* of the CoM derivatives belongs to some *convex* set.

6.4 LMPC-based gait generation

6.4.1 Classical approach

If we can assume the set of contact points to be given in advance (e.g. by some footstep planning stage), then we can follow Wieber (2006c); Winkler et al. (2015) and cast the walking gait generation problem into a linear MPC problem, as it is briefly summarized hereafter.

We first specify our process dynamics as a triple integrator of the CoM jerk, as follows:

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}, \tag{6.5}$$

where $\boldsymbol{x} \in \mathbb{R}^6$ is the state vector defined by the aggregation of the planar CoM position, velocity and acceleration, and $\boldsymbol{u} \in \mathbb{R}^2$ is the control input (which corresponds to the CoM

jerk). Consequently, $A \in \mathbb{R}^{6\times 6}$ and $B \in \mathbb{R}^{6\times 2}$ take the following form:

$$\begin{aligned} A &= \begin{bmatrix} 0_{2\times 2} & I_{2\times 2} & 0_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & I_{2\times 2} \\ 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 2} \end{bmatrix} \\ B &= \begin{bmatrix} 0_{2\times 2} \\ 0_{2\times 2} \\ I_{2\times 2} \end{bmatrix}. \end{aligned} \tag{6.6}$$

The ZMP can be defined as an output $\boldsymbol{z} \in \mathbb{R}^2$ of (6.5):

$$\boldsymbol{z} = C_{\text{zmp}}\boldsymbol{x}; \tag{6.7}$$

the definition of C_{zmp} follows from (6.4):

$$C_{\text{zmp}} = \begin{bmatrix} I_{2\times 2} & 0_{2\times 2} & -\frac{1}{\omega}I_{2\times 2} \end{bmatrix}. \tag{6.8}$$

Finally, piece-wise constant control input over some *control horizon* is assumed

$$\boldsymbol{u}(t) = \boldsymbol{u}_k \quad \forall t \in [t_k, t_{k+1}], \quad k \in \{0, \dots, M-1\}, \tag{6.9}$$

where t_k is the k-th discretization knot, and M denotes the control horizon length (in this work, a fixed discretization step Δt has been used). From standard theory of linear systems we know that the ZMP (as well as any other output) at time t_k depends linearly on both the initial state $\boldsymbol{x}_0 = \boldsymbol{x}(t_0)$ and the sequence of controls $\boldsymbol{U} \in \mathbb{R}^{2M}$, as specified below:

$$\boldsymbol{z}_k = \tilde{C}^k_{\text{zmp}}\boldsymbol{x}_0 + \tilde{D}^k_{\text{zmp}}\boldsymbol{U} \tag{6.10}$$

with $\boldsymbol{U} = \begin{bmatrix} \boldsymbol{u}_0^T & \dots & \boldsymbol{u}_{M-1}^T \end{bmatrix}^T$; the matrices $\tilde{C}^k_{\text{zmp}}$ and $\tilde{D}^k_{\text{zmp}}$ are obtained from integration of (6.5) over the knots (6.9).

The ZMP can then be constrained to the convex hull of the contact points over the whole control horizon. Indeed, the feasibility constraint (6.4) can be written as a *linear* inequality of the following form

$$\Big[(\boldsymbol{p}_{j(i),k} - \boldsymbol{p}_{i,k}) \times (\boldsymbol{z}_k - \boldsymbol{p}_{i,k})\Big]_z \leq 0 \tag{6.11}$$

for each time step k over the control horizon, and for each support polygon side $(i, j(i))$, where $j(i)$ denotes the subsequent of the i-th foot, according to a clockwise ordering[2]. The resulting optimization problem takes the form

$$\begin{aligned} \min_{\boldsymbol{U}} \; & \tfrac{1}{2} \sum_{k=1}^{M} \boldsymbol{x}_k^T Q_k \boldsymbol{x}_k + \boldsymbol{u}_k^T R_k \boldsymbol{u}_k \\ \text{s.t.} \quad & A_{\text{zmp}}(\boldsymbol{P})\boldsymbol{U} \leq \boldsymbol{b}_{\text{zmp}}(\boldsymbol{P}, \boldsymbol{x}_0), \end{aligned} \tag{6.12}$$

where A_{zmp} and $\boldsymbol{b}_{\text{zmp}}$ account for (6.11) when evaluated over all support polygons sides and over the control horizon as well. Such matrices depend on the current *and* future footsteps, which are collected in the vector $\boldsymbol{P} \in \mathbb{R}^{2\cdot(1+M_P)\cdot 4}$, with M_P representing the number of *predicted footsteps*.

Notice that, if the footsteps $\boldsymbol{p}_i$ are not optimized, then the constraint (6.11) is linear; on the contrary, if footsteps need to be included inside the optimization process, non-linearities arise in the form of *quadratic constraints*. Moreover, such a constraint becomes non-convex (see the appendix for a simple proof), resulting in an *NP-hard* problem. Even though several algorithms exist that allow to find a (local) minimizer of such a problem, it is the author's belief that finding a linearly constrained QP approximation of the full problem would be beneficial for at least two reasons:

- QPs are a standard class of optimization problems that are well-known in the scientific community; global minimizers can be quickly computed by means of off-the-shelf solvers (e.g. *qpOASES* (Ferreau et al., 2014a)). General-purpose NLP solvers, on the other hand, can be expected to be significantly slower.

- NLP solvers can only provide local minima of non-convex problems. It can be argued that the risk of converging to a "bad" local minimum may ruin the planner performance.

The remainder of this section is devoted to the development of such a QP approximation, that is the main contribution of the present work.

6.4.2 Proposed decomposition

As it was mentioned in the previous subsection, this chapter goal is to derive a QP approximation of problem (6.12) when optimizing for both ZMP and footsteps. More specifically,

[2] As it is customary in the literature, we assign integer labels to the four legs according to a clock-wise ordering and starting from the front-left leg.

the approximated feasible set should be a *linear subset* of the complete set (6.11), so that a solution to the approximated QP will also be a feasible point for the original problem.

To this aim, it is observed that the nonlinearity in (6.11) originates from the coupling between stance feet *pairs*. Indeed, also in the case of bipedal walking, Herdt et al. (2010a) noticed how nonlinearities arise whenever more than one stance foot is considered. With this in mind, this work proposes to split the set of the feet indices $I = \{1,2,3,4\}$ into two partitions of two indices each, I_A and I_B. Correspondingly, two auxiliary states $\boldsymbol{x}_A \in \mathbb{R}^6$ and $\boldsymbol{x}_B \in \mathbb{R}^6$ are introduced, such that the full robot state $\boldsymbol{x}$ is given by a convex combination of the two auxiliary states:

$$\boldsymbol{x} = \alpha\,\boldsymbol{x}_A + (1-\alpha)\,\boldsymbol{x}_B \tag{6.13}$$

for some parameter $\alpha \in (0\ 1)$, that we call *distribution factor*. In addition, auxiliary control inputs $\boldsymbol{u}_A \in \mathbb{R}^2$ and $\boldsymbol{u}_B \in \mathbb{R}^2$ are also defined such that an analogous relation as (6.13) holds for the same value of α:

$$\boldsymbol{u} = \alpha\,\boldsymbol{u}_A + (1-\alpha)\,\boldsymbol{u}_B. \tag{6.14}$$

Following these definitions, an auxiliary system whose can be defined with state $\tilde{\boldsymbol{x}} \in \mathbb{R}^{12}$ and input $\tilde{\boldsymbol{u}} \in \mathbb{R}^4$ are given by the concatenation of the two auxiliary states and inputs:

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x}_A \\ \boldsymbol{x}_B \end{bmatrix}, \qquad \tilde{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{u}_A \\ \boldsymbol{u}_B \end{bmatrix}. \tag{6.15}$$

Clearly, the auxiliary dynamics

$$\dot{\tilde{\boldsymbol{x}}} = \tilde{A}\,\tilde{\boldsymbol{x}} + \tilde{B}\,\tilde{\boldsymbol{u}} \tag{6.16}$$

is described by the following matrices:

$$\tilde{A} = \begin{bmatrix} A & 0_{6\times 6} \\ 0_{6\times 6} & A \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} B \\ B \end{bmatrix}. \tag{6.17}$$

The robot state $\boldsymbol{x}$ can then be recovered as an output for system (6.16), as it is shown below:

$$\boldsymbol{x} = C_{\text{state}}\,\tilde{\boldsymbol{x}} \tag{6.18}$$

$$C_{\text{state}} = \begin{bmatrix} \alpha\, I_{6\times 6} & (1-\alpha)\, I_{6\times 6} \end{bmatrix}. \tag{6.19}$$

Likewise, we can define outputs corresponding to the *auxiliary ZMPs* $\boldsymbol{z}_A$ and $\boldsymbol{z}_B$ by considering (6.4) for the auxiliary states $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$, respectively.

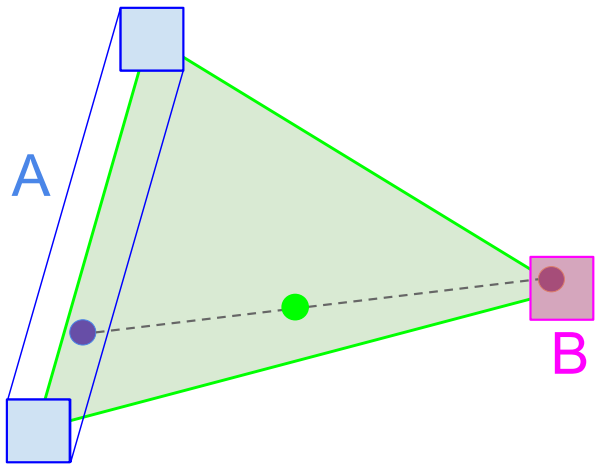


Figure 6.1 Decomposed feasibility constraint as described in Section 6.4.3. Auxiliary ZMPs are shown as colored circles for both system A (blue) and B (purple). The resulting global ZMP (green) is inside the support polygon.

6.4.3 Feasibility constraint

To generate linear constraints, it can be noticed that the two auxiliary states, together with the corresponding footsteps, define two *equivalent bipeds*. Drawing from (Herdt et al., 2010a), biped-like feasibility constraints can be defined for both auxiliary systems, enforcing the two auxiliary ZMPs to lie inside the corresponding biped supports. Finally, we notice that the full quadruped support is given by the convex hull of the two equivalent bipeds supports, according to the following expression:

$$z = \alpha z_A + (1-\alpha) z_B; \tag{6.20}$$

consequently, as the global ZMP is given by a convex combination of the auxiliary ZMPs, it will lie inside the full polygon. An illustration of this is given by Figure 6.1.

To obtain a numerically stable QP, equivalent bipeds feet size have been set to a small (but not zero) $\delta \boldsymbol{p} \in \mathbb{R}^2$.

6.4.4 Auxiliary state initialization

It is worth noticing that, having introduced new auxiliary states in our dynamics, we do not have an *observable* system anymore; broadly speaking, this means that the full state (6.15) cannot be reconstructed from the measured output, which is assumed to be the robot

state $\boldsymbol{x}$ defined by (6.18). As a consequence, it is impossible to compute (or estimate) in a meaningful way the initial value of the auxiliary state $\tilde{\boldsymbol{x}}$, which is needed at each control time by the MPC algorithm. However, since auxiliary sub-states do not carry any physical meaning, we are free to choose the corresponding value arbitrarily, as long as the following equality holds true:

$$\boldsymbol{x}_0 = C_{\text{state}} \tilde{\boldsymbol{x}}_0, \tag{6.21}$$

i.e. the initial robot state matches the measured one. Finally, let us notice how the initial auxiliary state appears *linearly* in both the cost function and the constraints of the LMPC problem; hence, the solver can determine an optimal value for $\tilde{\boldsymbol{x}}_0$ by introducing it as decision variable, and enforcing (6.21) as a constraint.

6.4.5 Parameters choice

To implement the proposed decomposition, we first need to choose a partitioning I_A, I_B. To this aim, let us notice that the quality of velocity tracking along different directions will differ, depending on the specific choice. More specifically, a front-back partitioning ($I_A = \{1,2\}$, $I_B = \{3,4\}$) will privilege forward walking, while a left-right partitioning ($I_A = \{1,4\}$, $I_B = \{2,3\}$) will favour lateral walking. This is explained as follows: in the first scenario, the supports of the two systems have the possibility to overlap along the forward direction, whereas they are always disjointed along the lateral direction. Consequently, the ZMP trajectory can be continuous along the forward axis, while it is always discontinuous along the vertical axis, causing oscillations that are well known in the literature. For a left-right partitioning, the vice-versa happens instead.

Concerning the role of the distribution factor α, it intuitively controls how much of the robot weight is supported by the auxiliary systems A and B, i.e. their relative load distribution.

Throughout the rest of this work, a front-back partitioning is employed, while the distribution factor is fixed at $\alpha = \frac{1}{2}$. A more detailed discussion on the role of the distribution factor is left for future work.

6.5 Implementation and experiments

The proposed algorithm was implemented in C++ inside the *OpenSoT* framework (Hoffman et al., 2017), that mainly targets hierarchical QP optimization problems with constraints, decoupling the concepts of *front-end*, i.e. the interface that allows to formulate the optimization problem, from the *back-end*, i.e. the tool that is actually used to solve it. More

specifically, the front end allows to combine *tasks* and *constraints* in a natural way by overloading suitable operators. The back-end implementation that was used in this work was powered by the *qpOASES* (Ferreau et al., 2014a) solver. The following tasks and constraints were implemented:

- tracking of a CoM velocity reference $\boldsymbol{v}_{\text{ref}}$:

$$J_{\text{vel}}(\boldsymbol{U}, \tilde{\boldsymbol{x}}_0) = \sum_{k=1}^{M} \left\| \dot{\boldsymbol{p}}_{\text{com},k} - \boldsymbol{v}_{\text{ref}} \right\|^2; \tag{6.22}$$

- a footstep regularization task, which tries to bias the feet positions to the center of the respective workspaces $\bar{\boldsymbol{p}}_j$, for each foot j belonging to the set of stance feet at time k, denoted by S_k:

$$J_{\text{footstep}}(\boldsymbol{U}, \boldsymbol{P}, \tilde{\boldsymbol{x}}_0) = \sum_{k=1}^{M} \sum_{j \in S_k} \left\| \boldsymbol{p}_{j,k} - \boldsymbol{p}_{\text{com},k} - \bar{\boldsymbol{p}}_j \right\|^2; \tag{6.23}$$

- minimum CoM acceleration and jerk tasks, as follows:

$$\begin{aligned} J_{\text{acc}}(\boldsymbol{U}, \tilde{\boldsymbol{x}}_0) &= \sum_{k=1}^{M} \left\| \ddot{\boldsymbol{p}}_{\text{com},k} \right\|^2 \\ J_{\text{jerk}}(\boldsymbol{U}, \tilde{\boldsymbol{x}}_0) &= \sum_{k=1}^{M} \left\| \boldsymbol{u}_k \right\|^2; \end{aligned} \tag{6.24}$$

- feasibility constraint (for the single auxiliary states), as described in Section 6.4.3;
- footspan constraint, whose aim is to ensure that the relative position of the feet lies between some lower and upper bound:

$$\Delta p_{\min}^{i,j} \leq \boldsymbol{p}_{i,k} - \boldsymbol{p}_{j(i),k} \leq \Delta p_{\max}^{i,j} \quad \forall i \in S_k; \tag{6.25}$$

  in (6.25) $j(i) \in S_k$ denotes the index of the leg adjacent to leg i, according to a clockwise ordering.

- Initial state consistency constraint (6.21).

The final objective function was obtained as a weighted sum of the atomic tasks that were listed above.

Table 6.1 Parameters used for the experiment.

Parameter	Value	Parameter	Value
M	20	w_{acc}	1
M_P	2	$w_{\text{vel},\,x}$	100
Δt	0.05 s	$w_{\text{vel},\,y}$	1000
$\delta p_{x,y}$	0.05 m	$w_{\text{footsteps}}$	1000
$\Delta p^{1,2}_{\min}$, $\Delta p^{4,3}_{\min}$	$[-0.3, 0.3]$ m	$\Delta p^{2,3}_{\min}$, $\Delta p^{1,4}_{\min}$	$[0.6, -0.2]$ m
$\Delta p^{1,2}_{\max}$, $\Delta p^{4,3}_{\max}$	$[0.3, 0.7]$ m	$\Delta p^{2,3}_{\max}$, $\Delta p^{1,4}_{\max}$	$[1.2, 0.2]$ m
T	3.0 s	β	0.8

In order to validate the proposed approach, we test it on our Centauro robot, which is powered by our control framework *XBotCore* (Muratore et al., 2017c), allowing us to control the robot under hard real-time (RT) constraints, while offering at the same time a complete interface to non-RT (NRT) external processes. A piece-wise constant velocity reference is commanded to the CoM, both in the forward and lateral direction. The gait pattern is dynamically computed as a function of the velocity reference according to the strategy of Ma et al. (2005), in order to maximize the static stability margin, using a fixed stride time T and duty cycle β. We tune the parameters as in Table 6.1, trying to balance tracking performance while avoiding excessive stretching of the legs. The resulting optimization problem has $n_V = 108$ decision variables and $n_C = 208$ constraints, which leads to roughly 50 Hz average execution frequency (see Fig. 6.3), which is more than three times faster when compared to the results of Winkler et al. (2017).

However, it should be noted that the proposed implementation does not take advantage of the sparsity pattern, as it does not exploit the fact that the hessian of the objective function is actually constant and does not need to be recomputed and re-factorized at each iteration. As a remark, notice that, by choosing $\beta < 0.75$, the proposed algorithm can also generate trotting motions.

To transfer the planned motion to the robot, a simple inverse kinematics (IK) scheme is adopted. Once again, we leverage on the OpenSoT framework to write a hierarchical IK problem with the following priorities:

1. CoM task + Feet position task
2. Knee task + Waist orientation task + Postural task,

where the aim of the *knee task* is to avoid the collision of the robot knees. Moreover, joint position and velocity limits are enforced as constraints. A low weight is assigned to the waist

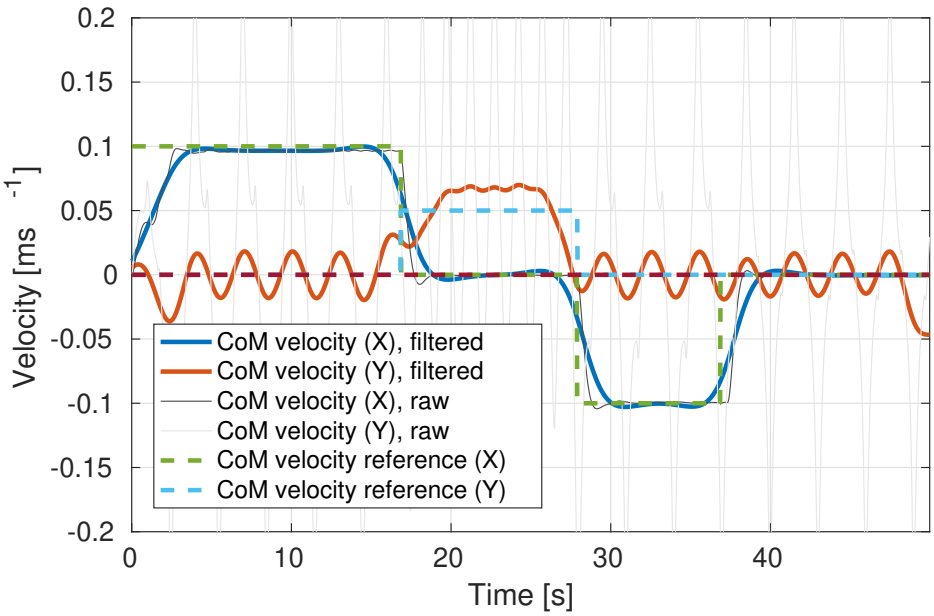


Figure 6.2 Planned CoM velocity profile (solid) against reference (dash). Data were processed through zero-phase low-pass filtering with cutoff frequency $f_c = 0.2$ Hz. Raw data are represented in grey.

orientation task, so that natural rotations arise from the minimization of joint velocities given by the postural task. From the software architecture point of view, the IK runs inside the RT loop at 1 kHz frequency, while the motion planning runs on a NRT ROS node.

Figure 6.4 and 6.2 show the achievable tracking performance. It can be noticed that, as discussed in Section 6.4.5, the forward velocity is tracked smoothly and precisely; on the contrary, lateral velocity is tracked only on average, and with greater steady-state error. Indeed, this behavior is inherited from the approach of (Herdt et al., 2010a), that the present work aims to extend to the quadrupedal case. Finally, it can be visually checked from Figure 6.5 that the proposed method does indeed generate a ZMP which is always *inside* the support polygon, therefore resulting in a feasible motion.

The disturbance rejection capabilities of the proposed method were also tested, by *simulating* an external impulsive force that is applied while the robot is walking. As it can be seen in Figure 6.4 (grey lines), the CoM plan deviates in the same direction of the force, in order to absorb the impact, while at the same time adapting the footsteps as well.

The outcome of our experiment is summarized in Figure 6.6, and in the attached video available at https://youtu.be/cGRRnL1Mfzs.

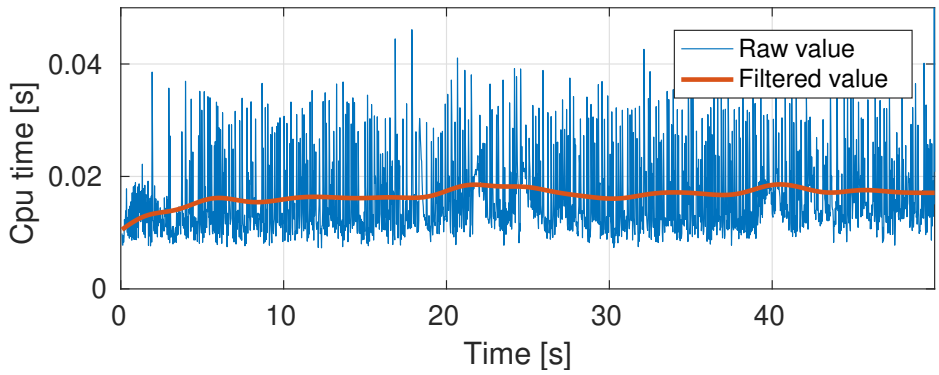


Figure 6.3 CPU time needed to fully set up and solve with a naive implementation the MPC QP problem on an Intel i7-6700@3400Hz CPU.

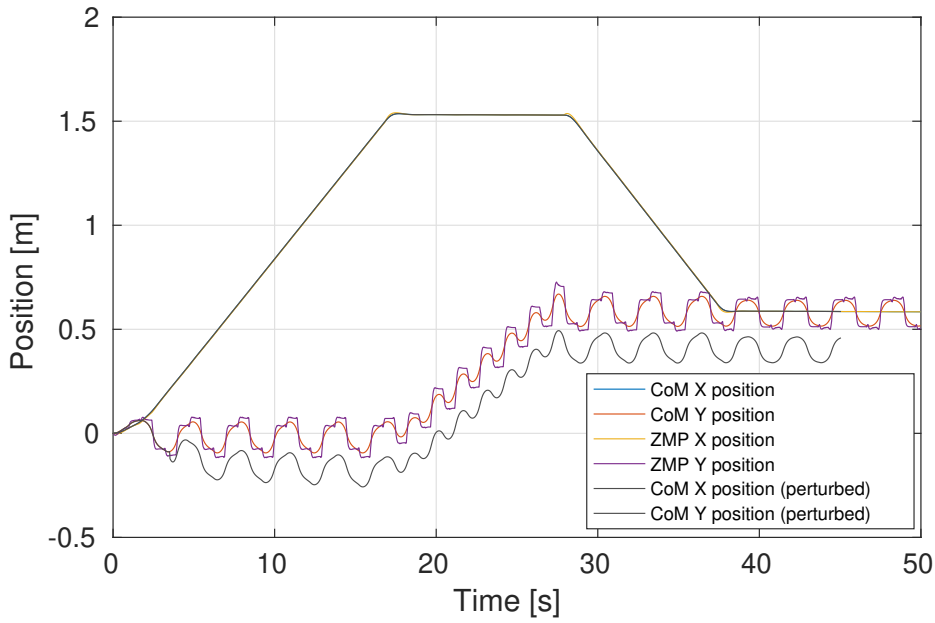


Figure 6.4 Planned CoM and ZMP trajectories without any external disturbance (colored lines), and with an external impulsive force $F = 60$ N applied in the negative y direction for 0.3 s (grey lines). For reference, the robot mass is roughly 90 kg.

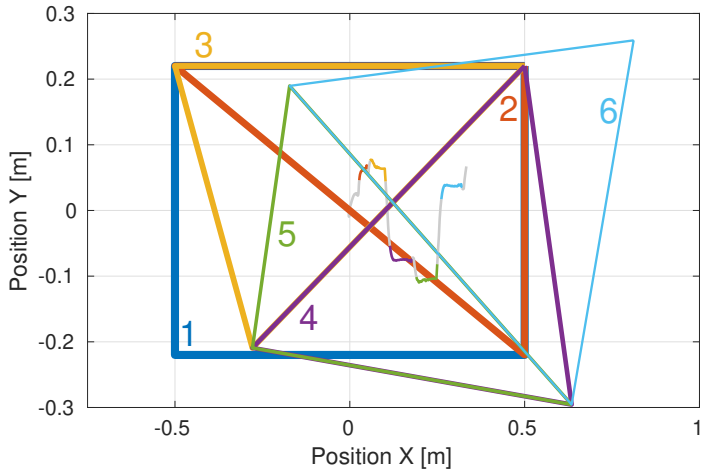


Figure 6.5 Sequence of support polygons generated by the proposed algorithm. The ZMP trajectory is plotted as well, with a color that matches the corresponding polygon (grey corresponds to four-stance phases).

Figure 6.6 Snapshots taken from an experiment on the actual CENTAURO robot. For the sake of clarity, the final backward phase is not included.

6.6 Discussion

The present chapter introduced a way to generate a walking motion of a quadrupedal robot, through the joint optimization of both the CoM trajectory *and* the footsteps as well. The proposed method is more robust than fixing the footsteps a-priori since, as discussed by Herdt et al. (2010a), enforcing the ZMP to always lie inside the pre-planned supports could require excessive CoM motions (or be unfeasible altogether). Besides, differently from the work of Winkler et al. (2017) which allows to find *local* minimizers of a non-convex optimization problem, this work proposes to find the exact global minimizer of an *approximated* QP problem. Such an approximation leads to the loss of some feasible solutions, and to an increased number of decision variables, which indeed represent drawbacks of the introduced formulation; on the other hand, the presented approach eliminates the risk of computing a bad local minimum for a the original non-linear program. In addition, we achieve a significantly faster computation time when compared to Winkler et al. (2017) even with a naive, dense implementation.

First trials on the quadruped robot CENTAURO have shown promising results. A mixed forward-lateral-backward gait was transferred to the actual hardware with little parameter tuning.

Future work will address the integration of in place rotations, that are possible under this framework provided that the orientation trajectory is fixed beforehand, as in Herdt et al. (2010a). Moreover, the role of the *load distribution* α, introduced in Section 6.3, must be further investigated. Such a parameter may also be considered to be time-varying, in order to produce more variegate CoM motions, and recover part of the lost solutions.

Lastly, more optimized solvers can be implemented by avoiding to uselessly recompute and re-factorize the hessian matrix, and also by carrying out an analysis of consecutive MPCs *active sets*.

Part III

Interaction control

Chapter 7

Prioritized force control

This chapter contributes with a *prioritized* Cartesian impedance control framework for fixed-base robots whose formulation resembles classical velocity level hierarchical inverse kinematics schemes, and which is solved by means of Quadratic Programming (QP) optimization. Such a controller is simpler than a full inverse dynamics based controllers, where simultaneous optimization over force and joint accelerations variables is carried out. In addition to strict priorities on force, the presented method can handle inequality constraints, allowing to easily express, e.g., torque bounds.

Starting from the aforementioned work on prioritized force control, a novel method is also presented which addresses the balancing problem for torque controlled legged robots through *post-optimization of contact forces*. The main concept consists in treating a legged robot as a fully actuated fixed-base system in order to compute the desired joint torques according to a fixed-based torque controller. The under-actuated component of the obtained torques is then mapped into contact forces through an optimal distribution problem. Besides extending previous work to the floating-base case, the proposed method has the notable advantage of avoiding the specification of a desired momentum of rotation, in addition to a reduced number of decision variables compared to full-inverse dynamics methods. The effectiveness of the proposed approach has been validated in simulation using two different humanoid platforms: the *Centauro* and the *Coman+* robots, both developed at Istituto Italiano di Tecnologia (IIT). Preliminary experimental results on *Coman+* are also presented.

This chapter is based on the following articles:

- Hoffman, E. M., Laurenzi, A., Muratore, L., Caldwell, D. G., and Tsagarakis, N. G. (2018). Multi-priority cartesian impedance control based on quadratic programming optimization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*
- Laurenzi, A., Hoffman, E. M., Polverini, M. P., and Tsagarakis, N. G. (2018a). Balancing control through post-optimization of contact forces. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 320–326. IEEE

7.1 Introduction and related works

Robots that have to perform tasks in unknown, real scenarios, need to handle contacts, both unintended (such as collisions with the environment) and intentional, e.g. with the objects to manipulate, or interactions with human co-workers. These requirements can be partially addressed by proper hardware (e.g. see Series Elastic Actuators (*SEAs*), Pratt and Williamson (1995)), which endow the robot with an inherent robustness to impacts, yet a predominant role is played by the capability of the controller to handle all these scenarios. Furthermore, it is important to notice that, in general, the forces involved during generic interactions are not usually known a priori.

Among all possible controllers which are suitable to tackle the needs of physical interaction, a notable one is the *Cartesian Impedance Controller*, which relates the wrench exchanged by the robot with the environment to Cartesian pose and velocity errors through Cartesian stiffness and damping matrices:

$$\boldsymbol{F} = \boldsymbol{K}\,\Delta\boldsymbol{x} + \boldsymbol{D}\,\Delta\dot{\boldsymbol{x}}, \tag{7.1}$$

where $\Delta\boldsymbol{x}$ is the pose error computed w.r.t. a reference pose and $\Delta\dot{\boldsymbol{x}}$ is the velocity error w.r.t. a reference velocity. The force $\boldsymbol{F}$ is often projected onto the robot joint torque space to obtained a control effort $\boldsymbol{\tau}$ through the task Jacobian. Equation (7.1) permits to handle interaction in different scenarios by setting appropriate set points and stiffness/damping matrices. Cartesian impedance control has been used for the dynamic interaction between the robot and the environment as well as for motion control.

Cartesian Impedance Control was firstly introduced in Hogan (1985) as an efficient operational space controller, capable to robustly handle contacts with the environment. As a matter of fact, Cartesian Impedance Control does not require the inversion of the kinematics

but only the computation of forward kinematics and, optionally, of the dynamic terms (e.g. feed-forward acceleration can be used to improve tracking quality), and it requires joint level torque control. As usually kinematic parameters are known with much better accuracy compared to dynamic parameters (i.e. masses, inertias), such a feedback law behaves more robustly when the robot model at hand has not been identified precisely, such as in the common case of CAD-generated models. Finally, pose-to-force relations can be used to implement rather complex behaviors, such as collision avoidance.

An application of Cartesian Impedance Control is the so called *Virtual Model Control*, which has been presented and implemented on a walking robot in Pratt et al. (2001). Virtual Model Control applies principles from Cartesian Impedance Control to attach simple virtual components (e.g. springs and dampers) to a selection of limbs of a legged robot, in order to obtained e.g. a walking behavior.

In (Ott, 2008), Cartesian Impedance Control is extended to the case of redundant robots, where the definition of a single priority level can turn out to be a limiting factor. In particular, the *null-space stiffness* concept is introduced as secondary task, with the aim to exploit joint impedance behavior in a Cartesian Impedance controller, without affecting the high priority Cartesian task.

Henze et al. (2016a) apply a hierarchical controller to the humanoid robot TORO for a balancing task using contacts. Cartesian Impedance Control is used to control the end-effectors of the robot. In this work, a QP problem is set up only to find the optimal contact forces distribution but the null-space projectors are explicitly computed *outside* the QP itself.

The work in (Platt Jr et al., 2010) introduces the theory behind a two level Cartesian Impedance controller in which the secondary task does not influence the primary task and it is minimized in accordance with the weighted squared magnitude of the Cartesian acceleration error. Furthermore, they state that the well-known control law of (Sentis and Khatib, 2005) does not guarantee that the secondary task is minimized under the same optimization criterion.

In (Saab et al., 2013a) the inverse dynamics problem is solved in its full extent by formulating a hierarchical QP problem in which torques, accelerations and contact forces are computed all together. On the one hand, such an approach is complete since it encompasses the definition of a tasks in terms of both force and Cartesian position, as well as joint acceleration, torque and contact force constraints. On the other hand, as previously stated, the optimization process involves joint acceleration, joint torque and contact force variables, potentially leading to a heavier computational burden. The proposed formulation is similar to the reduced form presented in (Saab et al., 2013a), but the method introduced in this work further extends it by allowing to handle contacts and motion together using the

aforementioned Cartesian Impedance control. Furthermore, the presented method uses a general formulation that is compatible with any QP solver.

The **first contribution** of this chapter is the introduction, implementation and validation of a multi-priority impedance controller that considers task priorities, inside a Quadratic Programming (QP) optimization framework. The main advantage of the proposed formulation of Cartesian Impedance controller is that such an approach allows to cope with highly redundant robots, taking into account hard priorities between tasks, and easily handling constraints in form of equalities and inequalities. In particular, inequality constraints and bounds permit to consider hardware limitations such as joint torque limits that are fundamental when working with real robotic platforms. The proposed formulation is shown to be simple and efficient, and it is validated both in simulations and with experiments under physical interactions during motions.

The **second contribution** of this chapter is the extension of the aforementioned methodology to floating base systems such as humanoid robots, and the application thereof to a balancing problem where, in addition to manipulation forces, contact forces must be optimized in order to cope in a stable way with external perturbations. Indeed, balancing represents a crucial requirement for legged robots expected to cope with a variety of unstructured terrains and environments. Despite a deep knowledge on the dynamics governing the balancing of articulated bodies, balancing is still considered a challenging control problem, especially when dealing with torque controlled humanoids and legged robots in general. In this respect, the existing control approaches are generally classified in two categories.

The first category achieves balancing through a two-stage methodology. An optimal contact force distribution problem is first solved with respect to the robot centroidal dynamics. This phase will be hereafter referred to as *pre-optimization* of contact forces. The computed contact forces are then mapped to joint torques under quasi-static assumptions, see Hyon (2009); Hyon et al. (2007) and Henze et al. (2016b, 2017); Ott et al. (2011), or through inverse dynamics, as in (Lee and Goswami, 2010, 2012) and (Stephens and Atkeson, 2010).

In opposition to the first category, the humanoid balancing problem can be alternatively addressed in a single-stage fashion by entirely exploiting the full-body inverse dynamics. Several inverse dynamics controllers, e.g. (Khatib et al., 2004; Sentis and Khatib, 2005; Sentis et al., 2010), compute joint torques by modeling contacts as rigid constraints and projecting the dynamics into a constraint free space. In this way an explicit solution of the contact force distribution problem is not required, although optimality of the problem is not guaranteed. Nevertheless, Righetti et al. (2013, 2011) showed that is possible to design inverse dynamics controllers and operational space controllers that are optimal with

respect to any combination of linear and quadratic cost in the contact forces and in the torque commands. On the other hand, methods based on a hierarchical Quadratic Programming (QP) formulation of the full-body inverse dynamics, see (Escande et al., 2014b; Mansard, 2012; Saab et al., 2013b), and (Herzog et al., 2014, 2016), explicitly consider contact forces as variables for the resulting optimization problem.

It is worth pointing out that, despite a clear advantage in terms of required computation time of the first category of methods over the second, the *pre-optimization* of contact forces raises a major concern about the role of the momentum of rotation in balance control. It is known in fact that the kinetic momentum of rotation is not directly related to the actual orientation of an articulated system (Saccon et al., 2017; Wieber, 2006b; Wieber et al., 2016). As a consequence, controlling the momentum of rotation for a balancing task may end up in a body rotation which is incompatible with the task itself.

The first contribution of the present chapter introduced a prioritized Cartesian impedance controller for redundant fixed-base robots has been proposed, based on a hierarchical QP formulation. Extending this approach to floating-base legged robots to subsequently address the humanoid balancing problem is the main focus of this second contribution. The proposed balancing controller belongs to the first category of methods, i.e. it adopts a two-stage technique. In this respect, aiming to overcome the main limitation inherent in the *pre-optimization* of contact forces, i.e. the control of the momentum of rotation, this work proposes to treat a legged robot as a fully actuated fixed-base system in order to compute the desired joint torques according to (Mingo Hoffman et al., 2018). Only at this point, the under-actuated component of the obtained torques can be mapped into contact forces through an optimal distribution problem, hereafter referred to as *post-optimization* of contact forces. This way the tricky specification of a desired momentum of rotation is circumvented. In addition to the aforementioned advantage over *pre-optimization* methods and the reduced number of decision variables compared to single-stage methods, the add-on nature of the proposed control approach entails no modification of the original algorithm in (Mingo Hoffman et al., 2018). It is also worth noticing that the proposed post-optimization yields a systematic way to adapt a generic fixed-base controller to an under-actuated system.

7.2 Prioritized force control

Let us consider a fixed-base n-dof manipulator, and let $\boldsymbol{q} \in \mathbb{R}^n$ be the corresponding joint configuration vector. Moreover, let $\boldsymbol{x} \in \mathbb{R}^m$ denote a *task*, i.e. some quantity of interest which

can be expressed as a non-linear function of the robot configuration

$$\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{q}). \tag{7.2}$$

Differentiation of (7.2) yields

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}, \tag{7.3}$$

where $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{m\times n}$ is the task Jacobian matrix. Since we are mainly interested in the specific case in which the manipulator is redundant with respect to the given task, the inequality $m < n$ will hold from now on.

Through manipulation of the system dynamics equation in contact with the environment through the term $\boldsymbol{f}_{ext}$

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{\tau} + \boldsymbol{J}^T\boldsymbol{F}_{ext}, \tag{7.4}$$

and the task description at the acceleration level (which can be obtained by further differentiation of (7.3))

$$\ddot{\boldsymbol{x}} = \boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}, \tag{7.5}$$

a task-space dynamics equation can be obtained, that is the following:

$$\boldsymbol{\Lambda}(\boldsymbol{q})\ddot{\boldsymbol{x}} + \boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{F}_{\tau|x} + \boldsymbol{F}_{ext}, \tag{7.6}$$

where $\boldsymbol{\Lambda} \in \mathbb{R}^{m\times m}$ is the task-space inertia matrix, and $\boldsymbol{\mu} \in \mathbb{R}^m$ is the vector of bias forces which are required to achieve zero acceleration. In (7.6) $\boldsymbol{F}_{\tau|x} \in \mathbb{R}^m$ is the vector of task-space forces that are equivalent to the joint torques $\boldsymbol{\tau}$, which for the sake of brevity will be simply denoted as $\boldsymbol{F}$ hereafter. The relation between these two quantities is linear (Siciliano et al., 2009):

$$\bar{\mathbf{J}}^T\boldsymbol{\tau} = \boldsymbol{F}, \tag{7.7}$$

where:

$$\bar{\boldsymbol{J}} = \boldsymbol{B}^{-1}\boldsymbol{J}^T\left(\boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{J}^T\right)^{-1}, \tag{7.8}$$

also known as the *dynamically consistent pseudo-inverse* (Khatib, 1987). Given the measured joint torques it is possible, through (7.7), to compute the force exerted by the robot joints at the end-effector. The inverse problem of (7.7) is to find the joint torques vector that realizes a certain task-space force. One specific solution of such a problem which is well known is the following:

$$\boldsymbol{\tau} = \boldsymbol{J}^T\boldsymbol{F}; \tag{7.9}$$

The solution from (7.9) is the one which makes the whole manipulator move as if it was subjected to the force $\boldsymbol{F}$; however, if we are not interested to this specific property, then the inverse problem of (7.7) can be considered in its full extent. The general, dynamically-consistent, solution of the inverse problem of (7.7) can be shown to be:

$$\boldsymbol{\tau} = \boldsymbol{J}^T \boldsymbol{F} + (\boldsymbol{I} - \boldsymbol{J}^T \bar{\boldsymbol{J}}^T)\boldsymbol{\tau}_0, \tag{7.10}$$

in which the second addend of the right hand side is a vector of torques that generates null-space manipulator motions but no forces (and, consequently, no acceleration) at the considered task.

Let us now move to a slightly more complex case, in which two tasks $\boldsymbol{x}_1 \in \mathbb{R}^{m_1}$ and $\boldsymbol{x}_2 \in \mathbb{R}^{m_2}$ are specified, along with their corresponding desired forces $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$. Furthermore, let us assume that task 1 is given a higher priority, meaning that we want the robot to "do its best" to perform task 2 *without affecting* the performance achieved at executing task 1. To reach this goal, the null-space of task 1 is exploited as in (7.10), choosing a vector $\boldsymbol{\tau}_0$ which makes the difference between the desired force $\boldsymbol{F}_2$ and the actual force $\boldsymbol{F}_2 = \bar{\boldsymbol{J}}_2^T \boldsymbol{\tau}$ as small as possible. This can be done via linear algebra techniques as shown in Flacco and De Luca (2015).

This work aims at specifying a Quadratic Programming (QP) problem, that solves the inverse problem of (7.7), taking into account priorities *and* constraints. QP is a convenient way to solve these kind of problems since it permits to specify inequality constraints that are useful in practical implementations. Furthermore, QP has already been used in the domains of Inverse Kinematics (IK) and Inverse Dynamics (ID). In particular, the proposed approach aims at being simpler, but more robust than full inverse dynamics. The desired virtual forces are assumed to be generated by virtual components placed at certain locations within the robot, or between the robot and the environment (Pratt et al., 2001), as for example a virtual spring:

$$\boldsymbol{F} = \boldsymbol{K}(\boldsymbol{x}_d - \boldsymbol{x}(\boldsymbol{q})), \tag{7.11}$$

where $\boldsymbol{K} \in \mathbb{R}^{m \times m}$ is a virtual stiffness matrix, $\boldsymbol{x}_d \in \mathbb{R}^m$ is the desired pose of the end-effector and $\boldsymbol{x}(\boldsymbol{q}) \in \mathbb{R}^m$ is the actual pose of the end-effector.

In this section, the inverse problem of (7.7) is formulated as a QP problem. More specifically, we want to find a formulation in which the cost functions, as well as the constraints that allow for strict priorities between tasks, are well defined.

7.2.1 Pseudo-inverse free formulation

First, notice that the control law (7.9) can also be obtained by considering the following QP problem:

$$\min_{\boldsymbol{\tau}} \| \bar{\boldsymbol{J}}^T \boldsymbol{\tau} - \boldsymbol{F} \|^2, \tag{7.12}$$

in fact, the quadratic cost function of (7.12) is:

$$F(\boldsymbol{\tau}) = \frac{1}{2}\boldsymbol{\tau}^T \bar{\boldsymbol{J}}\bar{\boldsymbol{J}}^T \boldsymbol{\tau} - \boldsymbol{F}^T \bar{\boldsymbol{J}}^T \boldsymbol{\tau} + \frac{1}{2}\boldsymbol{F}^T \boldsymbol{F}. \tag{7.13}$$

By convexity of the objective function, the solution has to satisfy the necessary and sufficient unconstrained optimality condition:

$$\frac{\partial F(\boldsymbol{\tau})}{\partial \boldsymbol{\tau}} = \bar{\boldsymbol{J}}\bar{\boldsymbol{J}}^T \boldsymbol{\tau} - \bar{\boldsymbol{J}}\boldsymbol{F} = 0. \tag{7.14}$$

If (7.8) is used in (7.14), the latter can be written as:

$$\boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{\tau} = \left(\boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{J}^T\right)\boldsymbol{F}, \tag{7.15}$$

a solution that satisfies (7.15) is given by (7.9), while a parametrization of the general solution is given by (7.10). The main drawback of formulation (7.12) is that the pseudo-inverse $\bar{\boldsymbol{J}}$ need to be computed in order to define the cost function. This is a computationally costly operation that indeed is already handled by the QP solver. To avoid the computation of $\bar{\boldsymbol{J}}$, we can use equation (7.15):

$$\min_{\boldsymbol{\tau}} \| \boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{\tau} - \boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{J}^T\boldsymbol{F} \|^2, \tag{7.16}$$

and it can be checked that the family of solutions satisfying (7.16) is (7.10), and also (7.9) as a special case.

7.2.2 Low priority joint space task

The most simple application of prioritized force control is given by a low priority task in joint space, acting in the null-space of a higher priority Cartesian force task. This can be expressed in terms of the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{\tau}} \|\boldsymbol{\tau} - \boldsymbol{\tau}_0\|^2_{\boldsymbol{W}} \\ &s.t.\ \boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{\tau} = \boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{J}^T\boldsymbol{F}, \end{aligned} \tag{7.17}$$

where $\boldsymbol{W} \in \mathbb{R}^{n\times n}$ is a symmetric positive-definite weight matrix. It is desirable that the solution of (7.17) is in the same form of (7.10), such that the standard statics equation is obtained whenever no constraint is active. To this aim, it is possible to show that the appropriate weight equals the joint-space inertia matrix, i.e., if $\boldsymbol{\tau}^*$ denotes the solution of (7.17), then

$$\boldsymbol{W} = \boldsymbol{B} \;\rightarrow\; \boldsymbol{\tau}^* = \boldsymbol{J}^T\boldsymbol{F} + (\boldsymbol{I} - \boldsymbol{J}^T\bar{\boldsymbol{J}}^T)\boldsymbol{\tau}_0. \tag{7.18}$$

Based on this result, it is now possible to extend this two-layer hierarchical problem to a generic number of priority layers, as depicted in the following section.

7.2.3 Prioritized QP Formulation

When multiple tasks are considered together, we can imagine a cascade of QP problems in which, at each level, a new solution $\boldsymbol{\tau}_i$ is found subject to the optimality constraint of the form:

$$\boldsymbol{J}_{i-1}\boldsymbol{B}^{-1}\boldsymbol{\tau}_i = \boldsymbol{J}_{i-1}\boldsymbol{B}^{-1}\boldsymbol{\tau}^*_{i-1}, \tag{7.19}$$

in which the optimal torques $\boldsymbol{\tau}^*_{i-1}$ are computed by the previous QP problems. Considering the i-th level of priority, the QP problem has the form:

$$\begin{gathered}
\min_{\boldsymbol{\tau}} \|\boldsymbol{J}_i\boldsymbol{B}^{-1}\boldsymbol{\tau}_i - \boldsymbol{J}_i\boldsymbol{B}^{-1}\boldsymbol{J}_i^T\boldsymbol{F}_i\|^2 + \varepsilon\|\boldsymbol{\tau}_i\|^2 \\
s.t.\ \mathbf{b}_\mathrm{l} \leq \mathbf{A}\boldsymbol{\tau}_i \leq \mathbf{b}_\mathrm{u} \\
\mathbf{u}_\mathrm{l} \leq \boldsymbol{\tau}_i \leq \mathbf{u}_\mathrm{u} \\
\boldsymbol{J}_{i-1}\boldsymbol{B}^{-1}\boldsymbol{\tau}_{i-1} = \boldsymbol{J}_{i-1}\boldsymbol{B}^{-1}\boldsymbol{\tau}_i \\
\vdots \\
\boldsymbol{J}_0\boldsymbol{B}^{-1}\boldsymbol{\tau}_0 = \boldsymbol{J}_0\boldsymbol{B}^{-1}\boldsymbol{\tau}_i,
\end{gathered} \tag{7.20}$$

where the speficic task at hand for this example is indeed a Cartesian task. The second term in the cost function is used as regularisation term, subject to a set of bounds $[\mathbf{u}_\mathrm{l},\ \mathbf{u}_\mathrm{u}]$, constraints $[\mathbf{b}_\mathrm{l},\ \mathbf{b}_\mathrm{u}]$ and optimality conditions given by the previous $i-1$ QPs.

Notice that it is possible to find in literature ways to formalize multiple QP problems in just one QP, for example in (Escande et al., 2014c) and (Liu et al., 2015).

7.2.4 Joint Torque Limits

The joint torque limits are an example of bounds for the QP problem previously formulated. These can be easily written in the form:

$$\boldsymbol{\tau}_{min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{max}. \tag{7.21}$$

It is important to notice that any feed-forward term, added to the output torque of the QP, should be subtracted to the available torques. For example, in order to compensate for gravity and Coriolis-centrifugal therms, the desired joint torques will be:

$$\boldsymbol{\tau}_d = \boldsymbol{\tau}_{opt} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}), \tag{7.22}$$

where $\boldsymbol{\tau}_{opt}$ are the optimal joint torques computed by the QP Optimization and $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q})$ are the joint torques to compensate Coriolis-centrifugal therms and gravity respectively. The bounds in (7.21) should be changed in:

$$\boldsymbol{\tau}_{min} - [\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q})] \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{max} - [\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\,\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q})]\,. \tag{7.23}$$

The presented formulation permits to easily specify constraints to handle joint velocity and acceleration limits, as in Del Prete (2018). More examples of possible bounds/constraints can be found also in Flacco and De Luca (2015); Saab et al. (2013a).

7.3 Validation

In this section, a series of experiments using the presented controller formulation, are shown both in Gazebo simulations and real hardware trials, also considering interaction with unknown external forces. We choose the humanoid upper-body of the Centauro robot (Baccelliere et al., 2017b) that consists in a two 7-DOFs arms mounted on top of a 1-DOF torso, as shown in Figure 7.1. At the time when experiments were performed, the platform relied on torque control for the torso joint and the first four joints of the arms chains. The wrist joints were therefore controlled in position mode and set to a fixed position value. For this reason, tasks have been defined only for the translation part of the end-effectors pose, and no command is sent to the wrist joints. For these experiments, two priority levels were defined, listed in descending priority order:

1. Left and right arm Cartesian impedance (translation only)

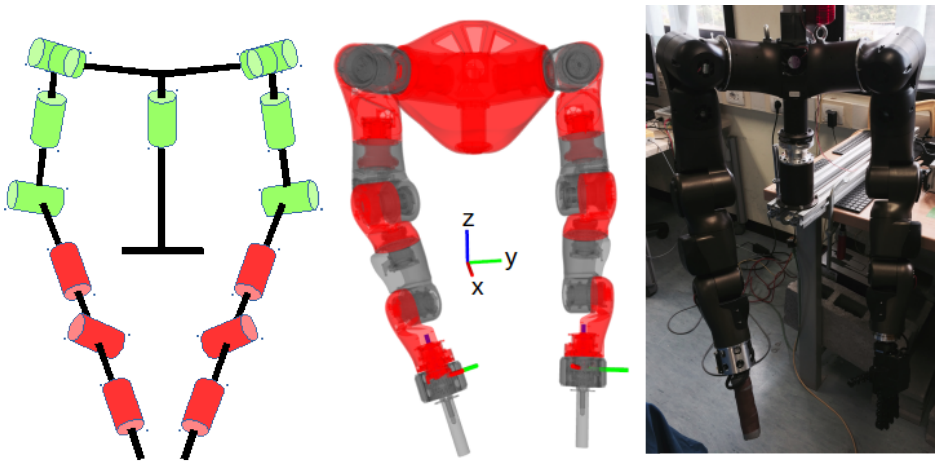


Figure 7.1 The green joints are controlled in torque mode while, the red ones are controlled in position mode and set to a fixed position values (left picture); the middle picture shows the frames used for the control; the right picture shows the real hardware platform

2. Joint space stiffness

subject to joint torque limits. The forces to generate the motion of the arms are generated using a virtual spring-damper:

$$\boldsymbol{F} = \boldsymbol{K}(\boldsymbol{x}_d - \boldsymbol{x}) - \boldsymbol{D}\dot{\boldsymbol{x}}, \tag{7.24}$$

Cartesian impedance values for the left and right arm are set to $\boldsymbol{K} = 700$ Nm^{-1} and $\boldsymbol{D} = 70$ Nm^{-1}s in all the directions. The Null-space stiffness is set to $\boldsymbol{K}_p = 5$ Nm^{-1} and $\boldsymbol{D}_p = 2$ Nm^{-1}s to all the joints. The controller is written in C++, inside the *OpenSoT* (Mingo Hoffman et al., 2015, 2017) library and the *inequality Hierarchical QP* (Kanoun et al., 2011) is implemented using the *qpOASES* (Ferreau et al., 2014b) solver. The controller is executed under the XBotCore framework (Muratore et al., 2017d), in real time (using *Xenomai 2.6*), within a control loop running at 1 kHz.

7.3.1 Gazebo: Torque Limits

A first simulated session has been carried out, in order to validate the performance of the proposed approach, under ideal conditions. More specifically, the controller behavior under limited torque was verified. To this aim, torque limits of all the joints are restricted to $\tau_{\max} = 20$ Nm. The simulated robot has to keep its end-effectors pose while an external force of magnitude $F = 20$ N is applied, for a duration $\Delta T_F = 2$ seconds, at the right end-effector. Figure 7.2 shows how the generated torques are bounded inside the given joint limits and

thanks to the QP formulation, an optimal solution given the constraints is found. When the

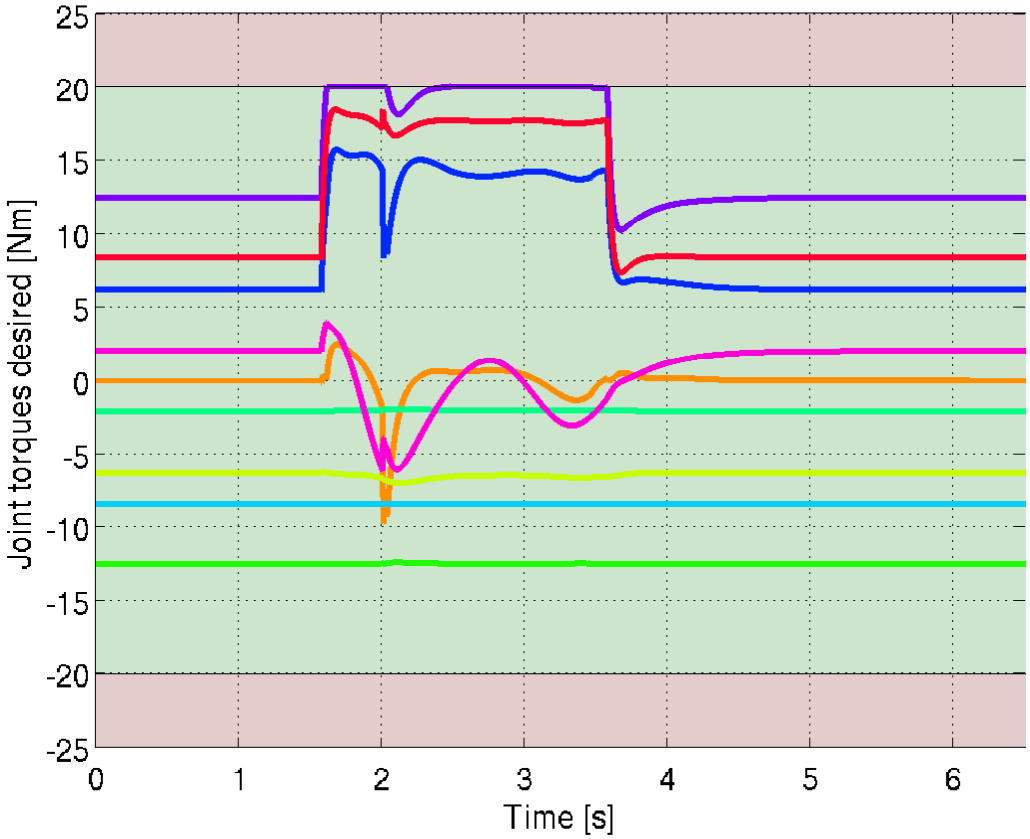


Figure 7.2 Joint torques computed by the controller: the green area represent the admissible region for the joint torques, the red area are the bounds

external force is not applied anymore, the end-effector goes back to its previous position.

7.3.2 Gazebo: Cartesian Circular Trajectory

In the second Gazebo experiment, the left arm of the robot is commanded to execute a circular trajectory while the right arm has to keep the end-effector pose steady. Multiple external forces are applied to both the end-effectors and the structure of the robot, as shown in Figure 7.3. In particular, external wrenches are applied for $\Delta T_F = 2$ seconds in the form of (i) a pure torque $\tau = 30$ Nm^{-1} about the torso joint axis, (ii) a force $F = 50$ N acting on the left elbow, (iii) a force $F = 30$ N on the right end-effector, and (iv) finally a force $F = 50$ N acting on the left end-effector. Figure 7.4 shows the desired Cartesian trajectories for the left end-effector along the x, y and z axis w.r.t. the computed ones. As expected, as long as the moving end-effector is not directly perturbed, the external forces applied to the structure

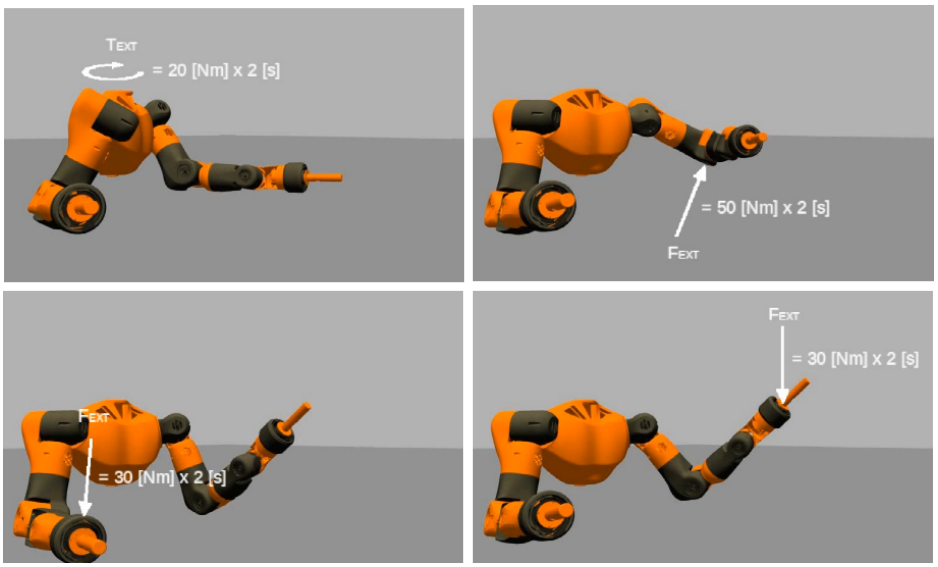


Figure 7.3 During the Gazebo simulation, the robot end-effector and structure are disturbed by unknown high external forces

of the robot have little effect on the motion result, as a consequence of the virtual stiffness acting on the moving link, and of priority consistency.

7.3.3 Robot: Cartesian Circular Trajectory

The previous experiment is replicated now in the real robotic hardware. Figure 7.5 shows where the external forces were applied during the motion. Figure 7.6 shows the comparison between desired and computed Cartesian trajectories for the left end-effector. As expected, there is a small error due to the uncompensated dynamical effects, the imperfect torque tracking (Figure 7.7) and the unmodeled dynamics. In this regard, it is worth to point out that CAD-based inertial properties were used in order to compute the robot dynamics, without any further identification procedure. Despite this, we can see a fairly good tracking quality and interaction behaviorr. As in the previous case, larger error tracking are present when the end-effector is directly disturbed while, when the structure is disturbed, the tracking quality remains the same, thanks to the imposed priorities.

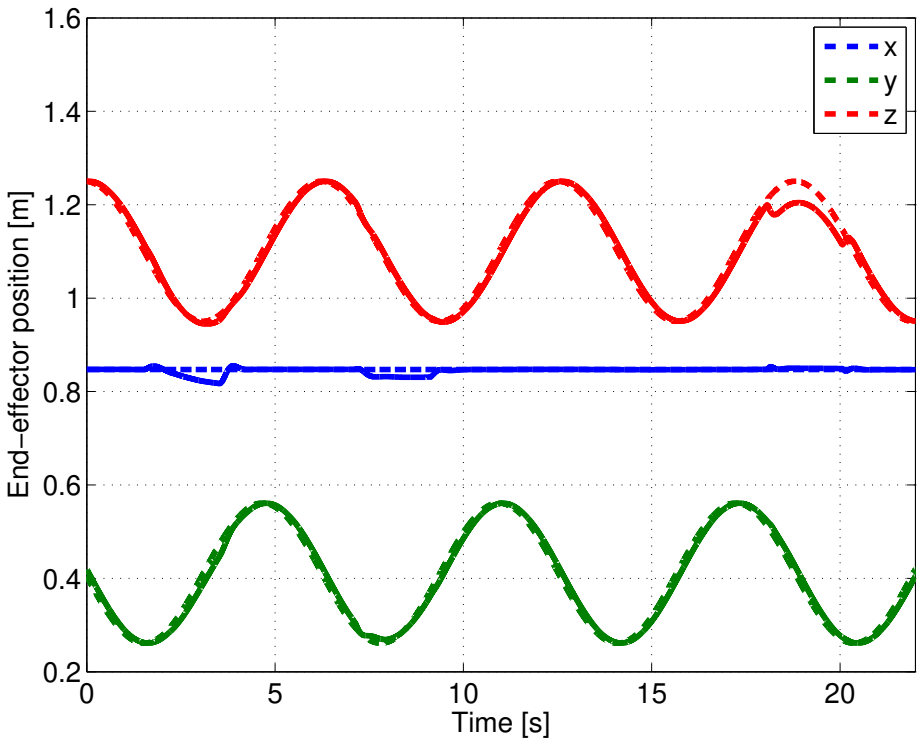


Figure 7.4 Desired (dashed) versus computed (continuous) Cartesian trajectories for the left end-effector under external pushes in the Gazebo experiment

7.4 Extension to floating base systems

This section presents an extension of the algorithm that was presented in the previous section to the floating-base case. From a modeling point of view, a legged robot shows two structural differences with respect to a fixed-base one:

- *under-actuation*: floating-base robots can be described in terms of n degrees of freedom (one for each joint) plus six additional coordinates describing the pose of some robot link w.r.t. to an inertial world frame. Such a link is usually called *floating-base*. These additional DoFs are usually modelled by introducing a virtual six-dof chain of passive (unactuated) joints, which is known as *virtual chain*.

- *Contact forces*: legged robots must always interact with the environment in order to be controlled in their full $(6+n)$-dimensional coordinate space. Indeed, from the so-called centroidal dynamics equation, we know that the "global motion" of the robot is entirely given by the contact forces.

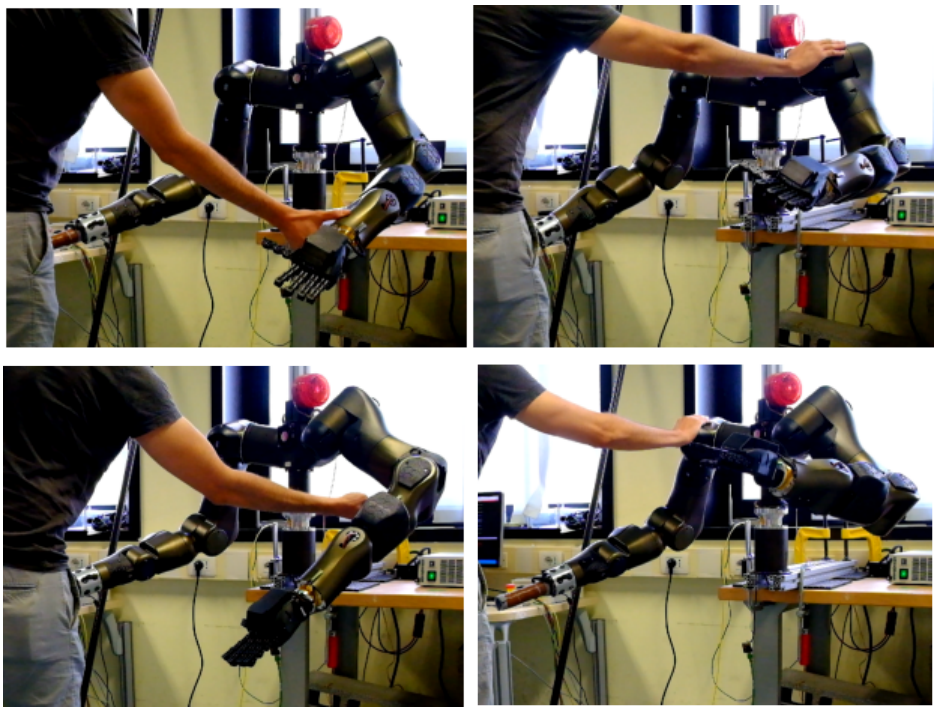


Figure 7.5 During the experiment the robot end-effector and structure are disturbed by unknown high external forces

The virtual chain formulation allows to easily extend the fixed-base dynamics (7.4) to the floating-base case. We just augment the generalized coordinate vector with six virtual joints as in the following equation[1]:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_u \\ \boldsymbol{q}_a \end{bmatrix}, \tag{7.25}$$

where $\boldsymbol{q} \in \mathbb{R}^{6+n}$ is obtained by stacking the configuration vector of virtual joints $\boldsymbol{q}_u \in \mathbb{R}^6$ with the one corresponding to the n actuated joints $\boldsymbol{q}_a \in \mathbb{R}^n$. Then, the floating-base dynamics equation is given by

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_{\mathrm{C}}^T \boldsymbol{F}_{\mathrm{C}} + \boldsymbol{J}^T \boldsymbol{F}_{\mathrm{ext}}; \tag{7.26}$$

compared to (7.4), the joint torques vector is pre-multiplied by a matrix $\boldsymbol{S} \in \mathbb{R}^{(n+6)\times n}$ that maps actuation torques into torques for the full floating-base robot:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{0}_{6\times n} \\ \boldsymbol{I}_{n\times n} \end{bmatrix}. \tag{7.27}$$

[1] We use the subscripts "u" for *unactuated*, and "a" for *actuated*, respectively.

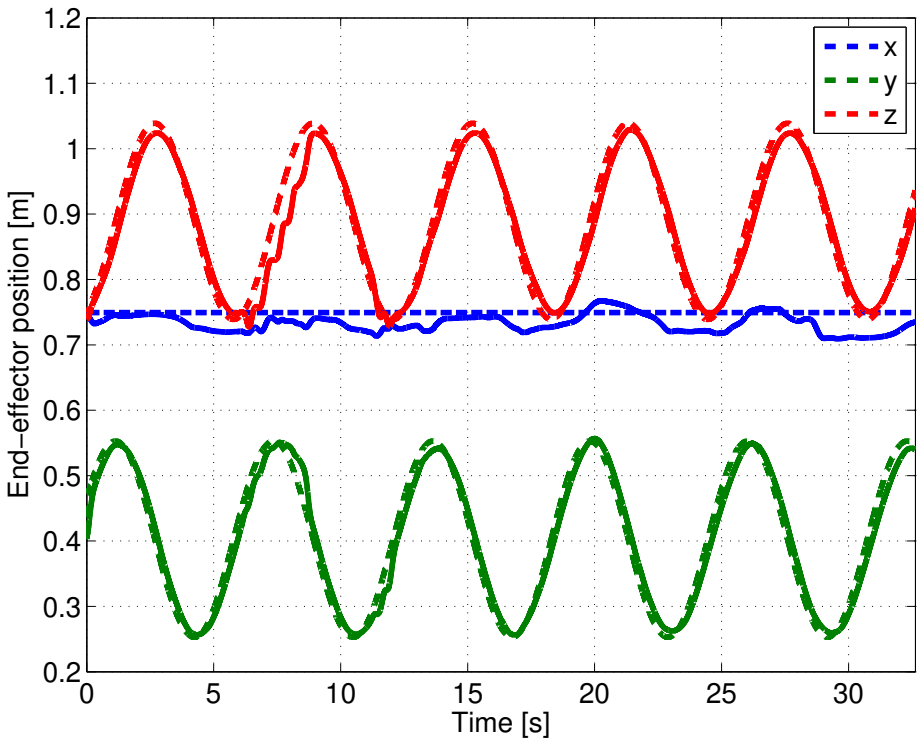


Figure 7.6 Desired (dashed) versus computed (continuous) Cartesian trajectories for the left end-effector under external pushes in the real hardware experiment

Finally, contact forces are taken into account by introducing the Jacobian of all support links $\boldsymbol{J}_{\mathrm{C}} \in \mathbb{R}^{k\times(n+6)}$ and the corresponding overall contact wrench $\boldsymbol{F}_{\mathrm{C}} \in \mathbb{R}^{k}$, with k equal to the contact constraint dimension (e.g. $k = 12$ for a humanoid in double support).

By repeating the same steps as for the fixed-base case, the task dynamics is obtained as follows:

$$\boldsymbol{\Lambda}(\boldsymbol{q})\ddot{\boldsymbol{x}} + \boldsymbol{\mu}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{F}_{\tau|x} + \boldsymbol{F}_{\mathrm{C}|x} + \boldsymbol{F}_{\mathrm{ext}}. \tag{7.28}$$

It can be seen that the contribution from actuated joint torques is changed slightly w.r.t. the fixed-base case, and it has the following expression:

$$\boldsymbol{F}_{\tau|x} = \left(\bar{\boldsymbol{J}}^{T}\boldsymbol{S}\right)\boldsymbol{\tau}; \tag{7.29}$$

however, this does not affect the mathematical formulation of the prioritized controller. The most important change is given by the coupling between the task dynamics and the contact wrenches, through the term

$$\boldsymbol{F}_{\mathrm{C}|x} = \bar{\boldsymbol{J}}^{T}\boldsymbol{J}_{\mathrm{C}}^{T}\boldsymbol{F}_{C} = \boldsymbol{\Lambda}\left[\boldsymbol{J}\boldsymbol{B}^{-1}\boldsymbol{J}_{\mathrm{C}}^{T}\right]\boldsymbol{F}_{C}; \tag{7.30}$$

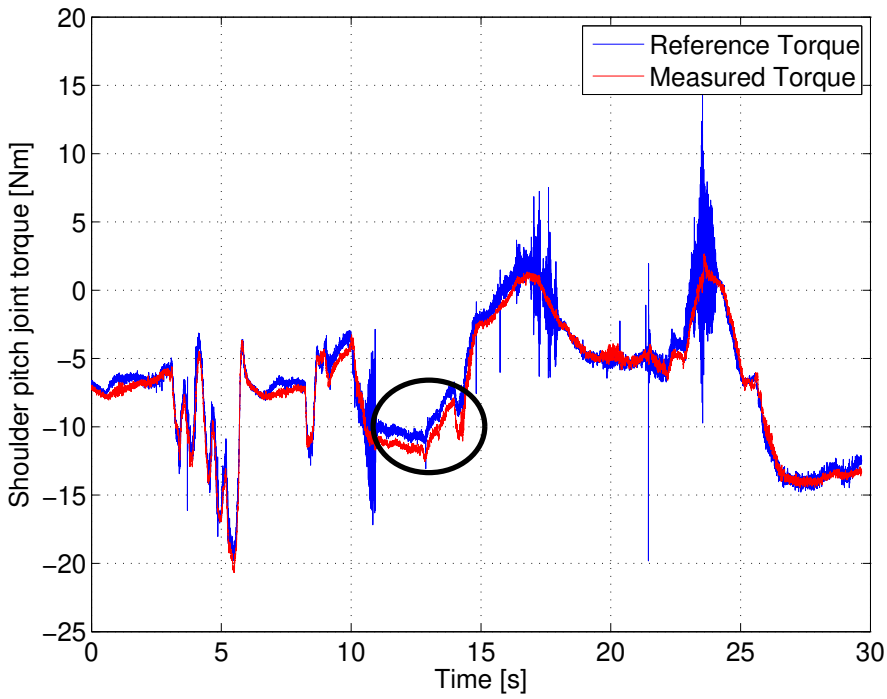


Figure 7.7 Comparison between measured torques and references, computed by the proposed algorithm, for the left shoulder pitch joint. It can be notice that there is, in some situation, a tracking error of approximately 10% as shown in the black circle

on this regard, it is worth noticing that coupling matrix between square brackets in (7.30) is non-zero for any task $\boldsymbol{x}$ that is specified w.r.t. the world frame: indeed, all tasks are coupled through the virtual chain connecting the floating-base to the world frame itself.

Such a coupling hinders the direct application of the algorithm of (Mingo Hoffman et al., 2018); in order to tackle the problem, three possible methods can be employed, as explained below:

- *pre-optimization of contact forces*. This approach has been successfully used in (Ott et al., 2011) for balancing control applied to the lower-body of the *TORO* robot. From the robot centroidal dynamics, the authors compute contact forces that achieve the desired center-of-mass and angular momentum behavior. More specifically, the angular momentum is used to perform orientation control of the robot base link.

  Once that such forces have been obtained, they can be made to disappear from the dynamics formulation by, for instance, redefining the bias torque vector as

$$\hat{\boldsymbol{h}} = \boldsymbol{h} - \boldsymbol{J}_{\mathrm{C}}^{T}\boldsymbol{F}_{\mathrm{C}}. \tag{7.31}$$

However, it is the author's belief that such an idea contains a pitfall, namely that the centroidal dynamics of the robot is constrained to the value obtained during the pre-optimization phase (see remark in Section 7.5.1). Consequently, the center-of-mass behaviour is always a first-priority task for the resulting controller, and the same applies to the angular momentum. This is undesirable for a prioritized controller; moreover, it is not clear which reference should be assigned to the angular momentum, given its non-holonomy as stated in (Wieber, 2006b).

- *Joint optimization of joint torque and contact forces.* This approach is conceptually similar to the one of (Herzog et al., 2016), where the optimization is carried out over contact forces and joint accelerations. Full control over the task hierarchy and system momentum is retained at the cost of an increased number of optimization variables.

- *Post-optimization of contact forces.* A third option, which is the main contribution of the present work, consists in treating the floating-base robot as a fixed-base one, by mathematically replacing the sum of under-actuated joint torques and contact torques with an equivalent completely-actuated torque vector. Then, a post-optimization phase is set up in order to map back the obtained virtual joint torques and forces to equivalent contact wrenches. At the best of the author's knowledge, this approach has not been explored before. A full description of such method is the subject of the following section.

7.4.1 Post-optimization of contact forces

Starting from (7.26), let us define an *equivalent fully-actuated torque vector* $\bar{\boldsymbol{\tau}}$

$$\bar{\boldsymbol{\tau}} = \boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_{\mathrm{C}}^{T}\boldsymbol{F}_{\mathrm{C}}; \tag{7.32}$$

with such a definition, the floating-base dynamics formally resembles the fixed-base one:

$$\boldsymbol{B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{h}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \bar{\boldsymbol{\tau}} + \boldsymbol{J}^{T}\boldsymbol{F}_{\mathrm{ext}}; \tag{7.33}$$

consequently, the algorithm of Section 7.2 can be applied without any modification, yielding some optimized value $\bar{\boldsymbol{\tau}}^{*}$ for the fully-actuated torque vector (7.32) that permits to achieve the desired hierarchical motion and interaction.

Our problem is then to recover the contact force information, by taking into account the under-actuated nature of the system. Essentially, this amounts to solving the system

of equations (7.32) for $\boldsymbol{\tau}$ and $\boldsymbol{F}_C$. On this regard, notice that the number of equations is $N_{\text{eq}} = n+6$, while the number of variables is $N_{\text{var}} = n+k$; this means that in an unconstrained, full-rank case the problem admits $\infty^{N_{\text{r}}}$ many solutions, with $N_{\text{r}} = k-6$.

The number of unknowns can be reduced by exploiting the structure of the actuation matrix (7.27). By focusing on the first six rows of (7.32), we obtain

$$\bar{\boldsymbol{\tau}}_u = \boldsymbol{J}_{\text{C},u}^T \boldsymbol{F}_{\text{C}}, \tag{7.34}$$

where the subscript "u" indicates the sub-matrix corresponding to the unactuated virtual joints. Equation (7.34) is torque-independent, and contains only the contact forces as variables. Once that these have been determined, joint torques can be recovered by looking at the bottom n rows of (7.32) and solving for $\boldsymbol{\tau}$:

$$\boldsymbol{\tau} = \bar{\boldsymbol{\tau}}_a - \boldsymbol{J}_{\text{C},a}^T \boldsymbol{F}_{\text{C}}. \tag{7.35}$$

The transpose of the unactuated part of the contact Jacobian acts as a *grasp matrix* $\boldsymbol{G} \in \mathbb{R}^{6\times k}$:

$$\boldsymbol{G} = \boldsymbol{J}_{\text{C},u}^T; \tag{7.36}$$

hence, we can draw inspiration from (Ott et al., 2011) and obtain a least-squares solution for $\boldsymbol{F}_{\text{C}}$:

$$\boldsymbol{F}_{\text{C}}^* = \boldsymbol{G}^\dagger \, \bar{\boldsymbol{\tau}}_u, \tag{7.37}$$

where the dagger symbol † denotes the pseudo-inverse. However, we can also exploit a QP formulation in order to enforce inequality constraints, as for instance friction cones, as follows:

$$\begin{aligned} &\min_{\boldsymbol{F}_{\text{C}}} \|\boldsymbol{G}\boldsymbol{F}_{\text{C}} - \bar{\boldsymbol{\tau}}_u\|^2 \\ &\text{s.t. } \boldsymbol{b}_{\text{l}} \leq \boldsymbol{D}\boldsymbol{F}_{\text{C}} \leq \boldsymbol{b}_{\text{u}} \\ &\quad \boldsymbol{u}_{\text{l}} \leq \boldsymbol{F}_{\text{C}} \leq \boldsymbol{u}_{\text{u}}. \end{aligned} \tag{7.38}$$

Torque constraints can be introduced as well by considering the dependency on the contact forces as given by (7.35).

7.4.2 Discussion

The main advantage of the proposed post-optimization formulation with respect to the pre-optimization is clearly given by the proper handling of priorities between tasks, since neither the Center of Mass (CoM) motion nor the robot angular momentum need to be set *a-priori*.

Moreover, the user is relieved from specifying a target angular momentum to the robot, which seems to be problematic as discussed in Section 7.4. With the proposed approach, target values for the centroidal dynamics are obtained from the fixed-base solution $\bar{\boldsymbol{\tau}}^*$, and *only then* the corresponding contact wrenches are optimized.

The main drawback is that the first optimization stage may give back a solution that is not feasible under force constraints (e.g. friction cones), which means that the objective value of (7.38) will be greater than zero. This could lead to loss of performance and, in the worst case, instability of the closed loop system. However, this situation can eventually be used to detect the need to perform a *recovery* action, e.g. a step.

On the other hand, a joint torque-force optimization strategy could perform better in such a case, since constraints on forces are taken into account for motion/force control as well. The price to pay is an increased number of decision variables.

As a final consideration, the proposed post-optimization approach gives us the possibility of adapting the fixed-base formulation to the floating-base case without any need for modification, since the under-actuation is dealt with at a separate stage that is *completely decoupled* from force/motion control. As a matter of fact, the proposed formulation permits to adapt *any* fixed-base torque controller to the floating base case, while fully retaining its behavior.

7.5 Implementation and experiments

In order to validate the proposed extension of the controller of Section 7.2 to the floating-base case, we set up two simulation scenarios in Gazebo involving two different legged platforms: our *Centauro* robot, and *Coman+*, which is a 28 DoF, 1.70 meters tall humanoid robot. Both robots are fully torque-controlled by feeding back the measured link-side joint torques. Moreover, our control architecture *XBotCore* (Muratore et al., 2017b) allows for transparent switch between simulation and real hardware, and hard real-time control on the actual robot. We conclude the section with a brief overview of some preliminary experimental results on the *Coman+* robot.

7.5.1 Gazebo simulations

The controller's first stage is formalized in the following way:

$$\begin{pmatrix} \left(\sum_i \ {}^{\text{World}}\mathscr{T}_{\text{Foot}_i}\right) / \\ {}^{\text{World}}\mathscr{T}_{\text{Waist}} / \\ \left({}^{\text{World}}\mathscr{T}_{\text{LHand}} + {}^{\text{World}}\mathscr{T}_{\text{RHand}}\right) / \\ \mathscr{T}_{\text{Posture}} \end{pmatrix} << \left(\mathscr{C}_{\substack{\text{Joint Torque}\\ \text{Limits}}}\right), \tag{7.39}$$

where the symbol ${}^{\text{A}}\mathscr{T}_{\text{B}}$ denotes a Cartesian impedance task of the frame B relative to the frame A; such a task is implemented by commanding simple virtual wrenches as in the following expression:

$$\boldsymbol{F}_d = \mathbf{K}_d\left(\boldsymbol{x}_d - \boldsymbol{x}\right) + \mathbf{D}_d\left(\boldsymbol{x}_d - \dot{\boldsymbol{x}}\right), \tag{7.40}$$

where $\mathbf{x}$, $\mathbf{x}_d$, $\dot{\mathbf{x}}$ and $\dot{\mathbf{x}}_d$ represent the actual and desired Cartesian poses and twists, respectively, whereas $\mathbf{K}_d$ and $\mathbf{D}_d$ denote the desired Cartesian impedance. These are combined by means of the operators "+" and "/", which are used to set aggregation and null-space relations, respectively. Finally, the symbol "<<" denotes insertion of constraints into the problem.

Recall from Section 7.4.1 that the first stage of the proposed method computes a fully actuated torque vector, that is then mapped to an under-actuated torque vector via post-optimization of contact forces. The post-optimization stage is implemented as in (7.38), considering linearized friction cones as inequality constraints:

$$|\boldsymbol{F}_t| \leq \frac{\sqrt{2}\mu}{2}\boldsymbol{F}_n, \ \ \boldsymbol{F}_n \geq 0; \tag{7.41}$$

where $\boldsymbol{F}_t$ and $\boldsymbol{F}_n$ are the tangential and normal components of contact forces, respectively, and $\mu = 0.3$ is the considered friction coefficient. As a final observation, for all our simulation experiments the state of the floating-base link is directly taken from the simulator.

Centauro balancing under external disturbances

The first simulation scenario consists on a balancing task for the *Centauro* robot under the disturbance of external forces applied on the robot waist. Screen shots from the performed simulations are reported in Fig. 7.8. The desired Cartesian impedance for the waist position task has been set equal to: $\mathbf{K}_d = \begin{bmatrix} 500 & 500 & 500 \end{bmatrix}$ Nm^{-1} and $\mathbf{D}_d = \begin{bmatrix} 200 & 200 & 200 \end{bmatrix}$ Nm^{-1}s.

A constant force of 200 N is first applied downward along the z-direction for 2 s. The obtained contact forces along the z-direction are shown in Fig. 7.9(a), while the corresponding waist position error is shown in Fig. 7.9(b). As expected, the maximum value of the actual

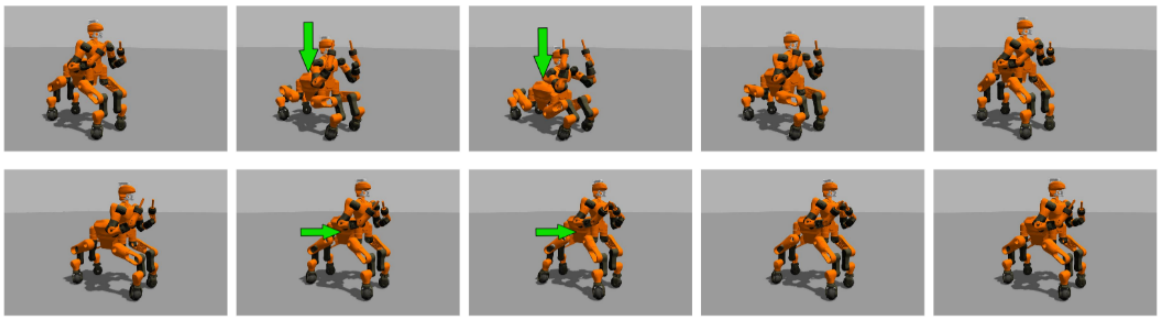


Figure 7.8 Screen shots from *Centauro* simulations in Gazebo. In the upper plots a constant force of 200 N is applied downward on the robot waist, while in the lower plots a constant force of 90 N is applied sideways.

waist position error (blue solid line), approximately 0.4 m, is consistent with the expected value (black solid line) obtained through (7.40).

To further highlight the impact of friction cones, a second simulation involves the application of a constant force of 90 N sideways along the y-direction, see Fig. 7.10. Note that, according to (7.41), the x and y components of the contact force on the rear right leg (yellow lines) are driven to zero at once with the z-component.

Coman+ picking a box

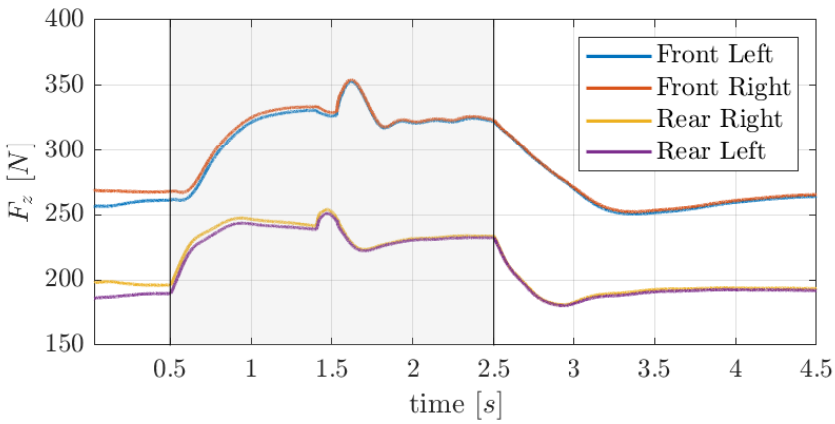
In the second simulation scenario we consider the *Coman+* robot picking a two kilogram box from the knee level. However, the box weight is not known to the controller. The task has to be performed using whole-body motions to reach the box and pick it up.

Notice that, in order to be able to reach the box, the z-translation task is removed from the waist control. As a side-note, the orientation is managed using a quaternion formulation as in (Nakanishi et al., 2008). The outcome of the simulation experiment can be seen in the video available at https://youtu.be/p8fwwV_zZa8.

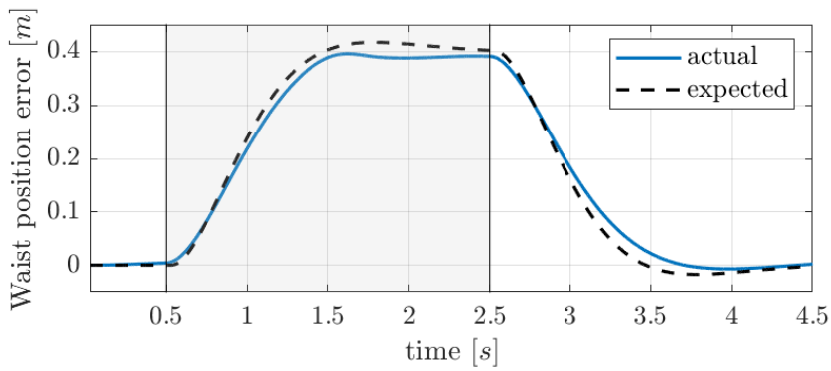
Remark: Notice that the obtained whole-body motion would be impossible under a pre-optimization framework, in which the CoM trajectory is decided beforehand. In this case, the only solution is to explicitly decrease the CoM height in order for the hands to reach the box.

7.5.2 Preliminary Experiments on the Coman+ platform

In this preliminary experiment we test the presented controller, with the real humanoid robot *Coman+*. As the only difference compared to our simulations, in this case we need to



(a) z-component of contact forces.



(b) Actual (blue solid line) vs. expected (black dashed line) waist position error along the pushing direction, i.e. the z-direction.

Figure 7.9 Time histories from *Centauro* simulation: an external constant force of 200 N is applied downward (z-direction) on the robot waist for 2 s (shaded area).

estimate the state of the robot base-link in terms of pose and twist w.r.t. an inertial world frame. In order to do so, kinematic information from the lower-body is simply fused with IMU measurements, assuming fixed contacts.

After some controller tuning, the robot could balance on two feet, even when subject to small external perturbations (see Figure 7.11*b*). Despite the encouraging results, instabilities and strong vibrations were observed, in a very similar way as described in (Englsberger et al., 2018); indeed, implementation of full-torque controllers on legged robots appears to be challenging, and it needs further investigation in order to improve the robustness of the controller and successfully deploy it to real systems. As intermediate step, a simplified architecture has been used in order to implement a real demonstrator with the Centauro robot, as described in the following section.

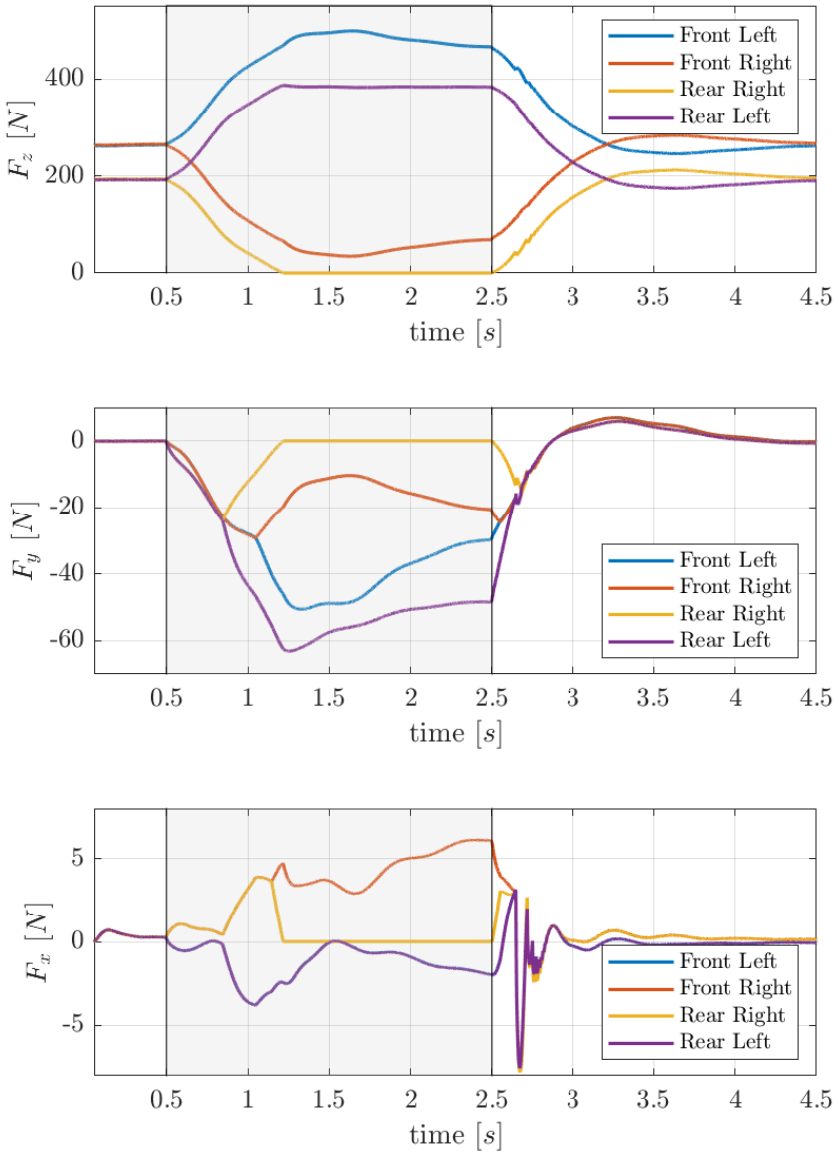
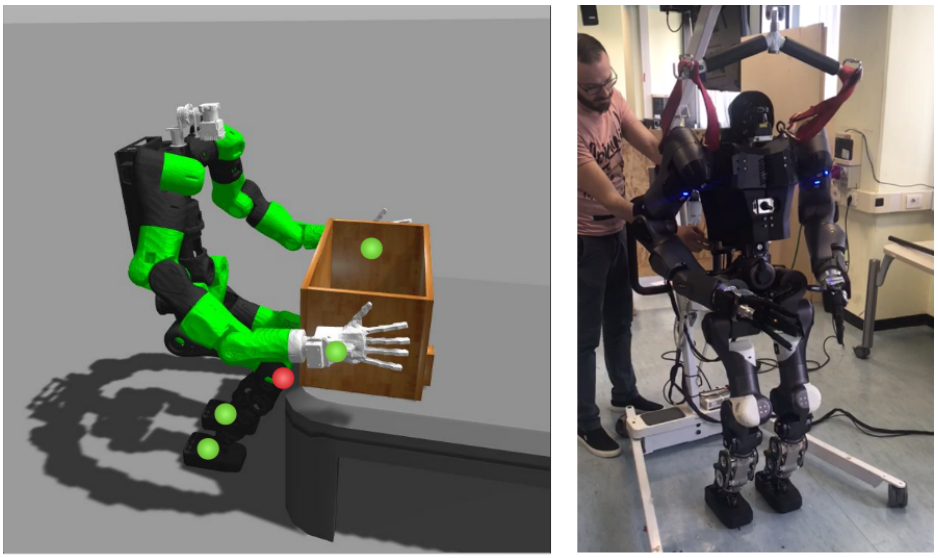


Figure 7.10 Contact forces' time history from *Centauro* simulation: an external constant force of 90 N is applied sideways (y-direction) on the robot waist for 2 s (shaded area).

7.6 Heavy object pushing demonstrator

The demonstrator described in this section addresses the control problem inherent in the pushing task of a heavy object with a centaur-type humanoid. This is a clear example of a multi-contact loco-manipulation task: due to the heavy weight of the object combined with the friction coefficient of the ground, it is not possible to simultaneously fulfill the balancing and the manipulation task, while keeping all the legs in contact with the ground.



a. b.

Figure 7.11 On the left, *Coman+* in simulation. During the squat motion needed to pick up the box, the knees of the robot (red dot) enter in contact with the table generating unwanted forces which perturb the CoM. Green dots represent intentional contacts. On the right preliminary experiments with the real *Coman+* platform

As a matter of fact, only by interacting with a different environment surface, e.g. by pushing with the rear legs against a wall, it is possible to successfully perform the task. In order to address this control problem, a control architecture has been implemented for multi-contact loco-manipulation tasks, conceived for torque-controlled legged systems such as Centauro. The proposed control architecture comprises:

- a *multi-contact planner*, that reasons about the robot centroidal dynamics under quasi-static assumptions, and relies on a continuous description of the environment, e.g. through *superquadric* functions;
- a *multi-contact controller* responsible for the tracking of the planned contacts and for reactive balancing, by means of hierarchical Inverse Kinematics (HIK) and instantaneous contact force distribution, respectively.

Finally, experimental validation has been performed on the pushing task of a heavy object using the Centauro robot. It is important to notice how this architecture is simplified with respect to the full floating-base Cartesian impedance controller of our simulated results as presented in Section 7.5.1, because of quasi-static assumptions. Nevertheless, as the task execution is expected to be slow, reasonable performance can be obtained on the real hardware.

More in detail, the manipulation task considered in this demonstrator consists in the pushing of a wooden cabinet, loaded with bricks, for a total weight exceeding 120 kg, see Fig. 7.12. Consequently, considering a Coulomb friction coefficient of 0.5, this task requires a pushing force greater than 600 N along the x-axis. The considered scenario is an example of a manipulation task that inherently requires multi-contact loco-manipulation capabilities to be successfully accomplished. As it can be noticed from Fig. 7.12(a), if all the legs of the robot stay in contact with the ground, it is not possible to effectively push the object by simply imposing a forward motion to the two hands (here the robot is controlled in position mode). In fact, the robot loses grip and slides backwards, while the object keeps its original position. Note that, due to the object weight and the inadequate level of grip forces that can be generated between the legs and the ground, the only way to accomplish the pushing task is to let the robot interact with an environment surface, different from the ground, that allows to exert a force along the pushing direction, without exceeding the friction limits of the contact surfaces. This can be done e.g. by pushing with the rear legs against a wall located behind the robot, as shown in Fig. 7.12(b). Assuming an a-priori knowledge on the surface location, this section introduces and validate a possible control architecture that enables to autonomously produce this behavior.

7.6.1 Continuous environment description

Representing the environment with simple but separate convex-hulls can be useful in contacts planning approaches, although it necessarily requires mixed-integer optimization as in Deits and Tedrake (2014); Ponton et al. (2016). In this respect, we hereafter propose to adopt a *unique* continuous description of the environment complexity, based on a *superquadric* function, which allows for continuous optimization. This choice is particularly effective in modelling walls, as required by the pushing task described in the previous section. This approach is similar to the work of Winkler et al. (2018), where all environment surfaces are combined and approximated with a smooth height-map.

The general form of a superquadric function with fully independent axis orders is given by:

$$S_{\mathrm{C}} : \left|\frac{x-C_x}{R_x}\right|^{P_x} + \left|\frac{y-C_y}{R_y}\right|^{P_y} + \left|\frac{z-C_z}{R_z}\right|^{P_z} = 1 \tag{7.42}$$

where the superquadric center is defined by the vector $\boldsymbol{C} = \begin{bmatrix} C_x & C_y & C_z \end{bmatrix}^T$, while the superquadric axial radii are defined by the vector $\boldsymbol{R} = \begin{bmatrix} R_x & R_y & R_z \end{bmatrix}^T$. Finally, $\boldsymbol{P} = \begin{bmatrix} P_x & P_y & P_z \end{bmatrix}^T$ defines the axial order, or curvature, which determines the actual shape of the superquadric.

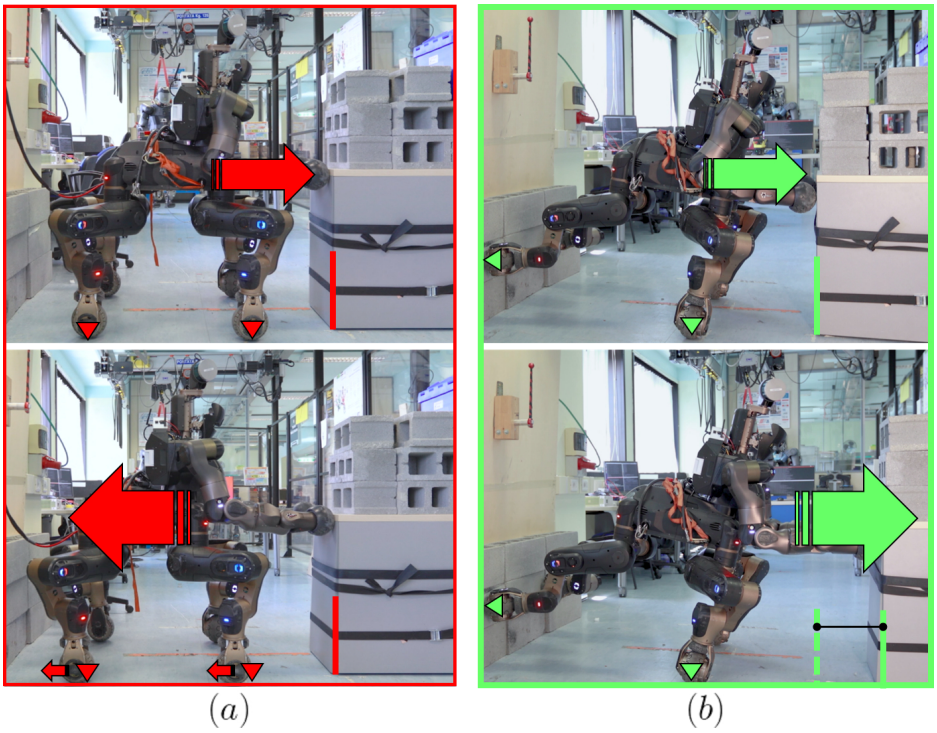


Figure 7.12 Centauro pushing a heavy object. When all the legs of the robot stay in contact with the ground (left-hand side) and a forward motion is imposed to the two hands, the robot structure moves backwards, while the object keeps its original position (see bottom left figure). Instead, as an outcome of the proposed approach, by pushing with the rear legs against a wall located behind the robot (right-hand side), Centauro can accomplish the manipulation task and effectively move the object.

If each element of $\boldsymbol{P}$ is greater than 2, (7.42) is continuously differentiable and defines a cube with smooth edges and corners, see Fig. 7.13. In the following, we will consider the

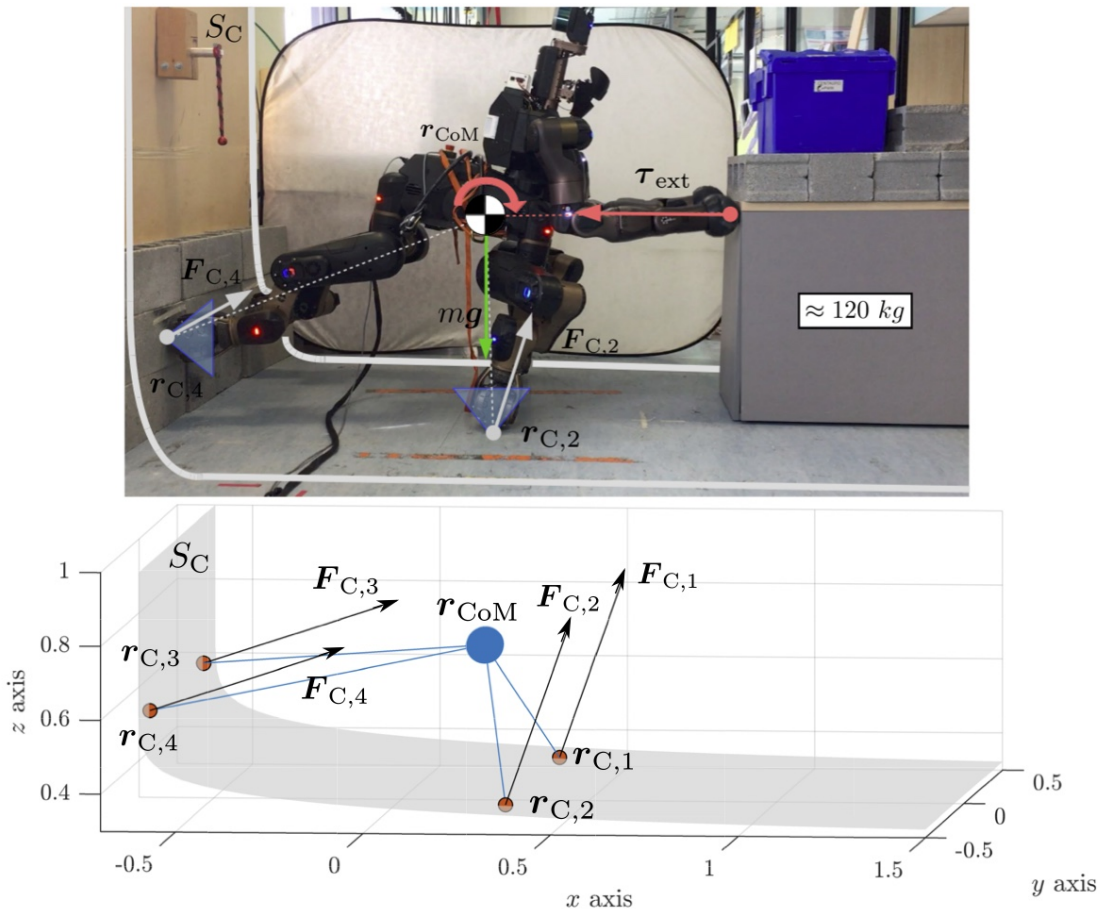


Figure 7.13 Graphical representation of the quantities involved in the planning problem to retrieve the final pose. The environment complexity, consisting of the ground and a vertical wall (see upper part), has been modeled through a single superquadric function (7.42) with $\boldsymbol{P} = [8\ 8\ 8]^T$ (see bottom part).

environment superquadric function to be manually specified, assuming the surfaces locations to be known. Alternatively, the superquadric could be automatically generated from point cloud data.

7.6.2 Multi-contact planning

Based on: (i) the robot centroidal statics model

$$\begin{cases} \mathbf{0} &= m\boldsymbol{g} + \sum_i \boldsymbol{f}_i \\ \mathbf{0} &= \sum_i (\boldsymbol{p}_{C,i} - \boldsymbol{p}_{com}) \times \boldsymbol{f}_i + \boldsymbol{m}_i \end{cases}, \tag{7.43}$$

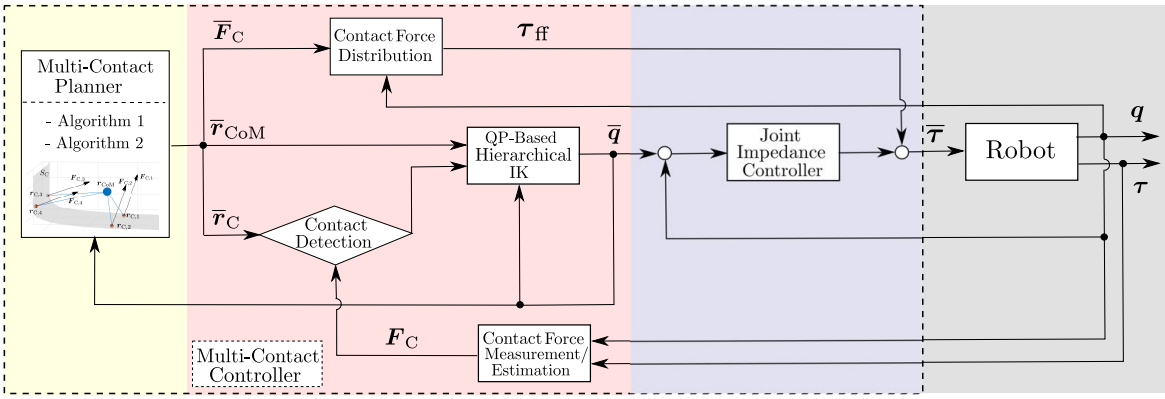


Figure 7.14 Overview of the control architecture. "Algorithm 1" is the *multi-contact loco-manipulation problem*, whereas "Algorithm 2" is the *contact-lifting problem*, as described in the text.

(ii) a continuous description of the environment (7.42), and (iii) Coulomb friction cones, it is possible to set up two different nonlinear programming (NLP) problems.

The first one, referred to as *multi-contact loco-manipulation problem*, produces a quasi-static pose, in terms of CoM position $\overline{\boldsymbol{p}}_{com}$, contact positions stacked together into a vector $\overline{\boldsymbol{p}}_{\mathrm{C}}$, and contact forces $\overline{\boldsymbol{F}}_{\mathrm{C}}$, given a desired manipulation wrench at the CoM frame $\boldsymbol{\tau}_{\mathrm{ext}}$, and the environment superquadric S_{C}.

The second one, referred to as *contact-lifting problem*, produces the CoM position $\overline{\boldsymbol{p}}_{com}$ and the contact forces $\overline{\boldsymbol{F}}_{\mathrm{C}}$, which are needed to reach the final pose produced by the first state, by lifting one leg at the time. In this case, a number of successive optimizations is performed, every time fixing the feet positions and setting the force of the swing foot to zero.

Both problems can be obtained as sub-problems of the following NLP:

$$
\begin{aligned}
&\min_{\boldsymbol{x}} \|\boldsymbol{p}_{\mathrm{com}} - \boldsymbol{p}_{\mathrm{com}}^{\mathrm{des}}\|^2_{2,\mathbf{W_{CoM}}} + \|\boldsymbol{p}_{\mathrm{C}} - \boldsymbol{p}_{\mathrm{C}}^{\mathrm{des}}\|^2_{2,\mathbf{W_r}} + \|\boldsymbol{F}_{\mathrm{C}}\|^2_{2,\mathbf{W_F}} \\
&\text{subject to} \\
&\quad m\boldsymbol{g} + \boldsymbol{G}_{\mathrm{CD}}\boldsymbol{F}_{\mathrm{C}} + \boldsymbol{\tau}_{\mathrm{ext}} = \mathbf{0} \\
&\quad \boldsymbol{p}_{\mathrm{C},i} \in S_{\mathrm{C}} \\
&\quad \{\boldsymbol{F}_{\mathrm{C},i}, \boldsymbol{p}_{\mathrm{C},i}\} \in \mathscr{F}(\boldsymbol{F}_{\mathrm{C},i}, \boldsymbol{p}_{\mathrm{C},i}, \mu_i) \\
&\quad \underline{\mathbf{u}} \leq \boldsymbol{x} \leq \overline{\mathbf{u}}
\end{aligned}
\quad , \tag{7.44}
$$

where (i) $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}_{\mathrm{com}} & \boldsymbol{p}_{\mathrm{C}} & \boldsymbol{F}_{\mathrm{C}} \end{bmatrix}^T$ is the vector of optimization variables, (ii) $\boldsymbol{G}_{\mathrm{CD}}$ maps contact forces to wrenches at the robot CoM according to (7.43), and (iii) $\mathscr{F}$ denotes the set of contact forces satisfying friction constraints.

7.6.3 Contact Force Distribution

Building upon the work of Section 7.4, it is possible to formulate the following contact force distribution problem in order to track the planned contact forces $\overline{\boldsymbol{F}}_{\mathrm{C}}$, while providing reactive balance capabilities:

$$\begin{aligned} &\min_{\boldsymbol{F}_{\mathrm{C}}} \ \|\boldsymbol{F}_{\mathrm{C}} - \overline{\boldsymbol{F}}_{\mathrm{C}}\|_2^2 \\ &\text{subject to} \\ &\quad \boldsymbol{G}\boldsymbol{F}_{\mathrm{C}} = \boldsymbol{g}_{\mathrm{u}}(\boldsymbol{q}) \\ &\quad \boldsymbol{b}_{\mathrm{l}} \leq \boldsymbol{D}\boldsymbol{R}_{\mathrm{C}}\boldsymbol{F}_{\mathrm{C}} \leq \boldsymbol{b}_{\mathrm{u}} \end{aligned} \tag{7.45}$$

The equality constraint ensures closed-loop balancing by employing (7.43) under quasi-static assumptions. The grasp matrix $\boldsymbol{G}$, see (7.36), and the floating-base gravitational term $\boldsymbol{g}_{\mathrm{u}}(\boldsymbol{q}) \in \mathbb{R}^6$ are computed w.r.t. the closed-loop joint coordinates, based on IMU feedback. The inequality constraints account for polyhedral linearized friction cones:

$$F^z_{\mathrm{C,i}} \geq 0, \qquad |F^{x,y}_{\mathrm{C,i}}| \leq \tilde{\mu}_i \, F^z_{\mathrm{C,i}} \tag{7.46}$$

where $\tilde{\mu}_i = \frac{\sqrt{2}}{2}\mu_i$ models the inner approximation of the conic Coulomb friction constraints $\mathscr{F}$, and the $\boldsymbol{R}_C$ rotation matrix maps contact forces from the world frame to the contact local frame, whose z-axis is the outward normal unit vector.

Finally, the actuated torque $\boldsymbol{\tau}_{\mathrm{ff}} \in \mathbb{R}^n$ to be sent to the *joint-level controller* as a torque feed-forward term (see Fig. 7.14) is computed according to (7.35).

7.6.4 Experiments

The *multi-contact planner* is executed inside a ROS node running at 10 Hz. Nonlinear optimizers have been implemented using IFOPT (Winkler, 2018), an Eigen-based C++ interface to the nonlinear programming solver IPOPT (Wächter and Biegler, 2006). To decrease the nonlinearity of $\mathscr{F}(\boldsymbol{F}_{\mathrm{C},i}, \boldsymbol{p}_{\mathrm{C},i}, \mu_i)$ and speed up the solver computation, the contact-surface normals $\boldsymbol{n}_{\mathrm{C}}$ were included in the set of decision variables. The considered set of contact points $\{\mathrm{C}_i\}$ comprises the four legs, while a pure manipulation force of 800 N along the x-axis is applied at the two hands; the considered manipulation wrench is $\boldsymbol{\tau}_{\mathrm{ext}} = \begin{bmatrix} 800 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$. The environment superquadric axial curvature has been chosen equal to $\boldsymbol{P} = \begin{bmatrix} 8 & 8 & 8 \end{bmatrix}^T$, while the vectors $\boldsymbol{C}$ and $\boldsymbol{R}$ have been chosen based on an approximate knowledge of the wall position w.r.t. the robot base (see Fig. 7.13).

The *multi-contact control* layer runs at 100 Hz. The HIK is managed by the *CartesI/O* framework within a ROS node, as described in Chapter 4. The considered SoT can be written using the Math of Task (MoT) formalism as in Mingo Hoffman et al. (2017) as follows:

$$\begin{pmatrix} \left(\sum_i {}^{\text{World}}\mathscr{T}^{[\text{XYZ}]}_{\text{Foot}_i}\right) / \\ \left({}^{\text{World}}\mathscr{T}_{\text{CoM}} + {}^{\text{World}}\mathscr{T}_{\text{LHand}} + {}^{\text{World}}\mathscr{T}_{\text{RHand}}\right) / \\ \left(\mathscr{T}_{\text{Posture}} + \sum_i {}^{\text{World}}\mathscr{T}^{[\text{RPY}]}_{\text{Ankle}_i} + {}^{\text{World}}\mathscr{T}^{[\text{RPY}]}_{\text{Waist}}\right) \end{pmatrix} << \left(\mathscr{C}_{\substack{\text{Pos.}\\ \text{Lims}}} + \mathscr{C}_{\substack{\text{Vel.}\\ \text{Lims}}}\right), \tag{7.47}$$

Here the $+$ and $/$ symbols are used to specify *soft* and *hard* priorities among tasks, respectively. ${}^{\text{A}}\mathscr{T}_{\text{B}}$ denotes a Cartesian task of the frame B relative to the frame A, while the $<<$ symbol is used to specify constraints to the whole stack or to single levels of the stack. A separate ROS node is dedicated to the *contact force distribution* problem for the considered set of contact points, i.e. the four legs. The *contact detection* has been implemented by evaluating the following logic condition:

$$\text{sign}\left(\boldsymbol{F}^{(\text{est})}_{\text{C,i}} \cdot \boldsymbol{n}_{\text{C,i}} - F_{\text{cnt}}\right) \tag{7.48}$$

where $F_{\text{cnt}} > 0$ is a scalar contact threshold. Since Centauro is not equipped with any force/torque sensors mounted at the end-effectors, the estimated contact forces $\boldsymbol{F}^{(\text{est})}_{\text{C}}$ are computed under quasi-static assumptions as:

$$\boldsymbol{F}^{(\text{est})}_{\text{C}} = \boldsymbol{J}^{\dagger T}_{\text{C,a}}\left(\boldsymbol{g}_a(\boldsymbol{q}) - \boldsymbol{\tau}_{\text{mes}}\right) \tag{7.49}$$

being $\boldsymbol{\tau}_{\text{mes}} \in \mathbb{R}^n$ the measured link-side joint torques and $\boldsymbol{g}_a(\boldsymbol{q}) \in \mathbb{R}^n$ the actuated gravitational term computed based on closed-loop joint coordinates, while † stands for the Moore-Penrose pseudo-inverse. Finally, the *joint-level controller* runs at 2 kHz.

Snapshots from a complete experiment are shown in Fig. 7.15 (these results are also illustrated in the video available at https://youtu.be/Xhn3MwHh1X0).

7.7 Discussion

In this work, a novel method to to formulate task-space inverse dynamics of floating-base robots has been developed as a follow up of the work in (Mingo Hoffman et al., 2018). The method consists in a two step optimization: during a first optimization stage, a fully-actuated torque vector that realizes the desired tasks is found using (Mingo Hoffman et al., 2018);

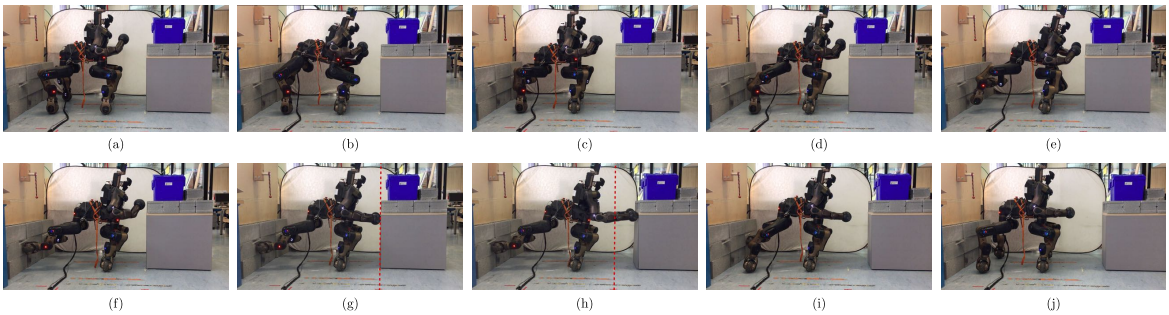


Figure 7.15 Screenshots from a pushing experiment. In (a)-(f) Centauro reaches the final multi-contact loco-manipulation pose produced by the first stage. This is done by iteratively applying the contact lift planner in order to be able to lift and move the rear legs toward the planned contact locations on the wall, see (d) and (f). In (g) the two arms approach the object in velocity control mode, until contact is detected. In (h) the contact forces $\boldsymbol{F}_{\mathrm{C}}^{(1)}$ given by the post-optimization stage proposed in Section 7.4 are applied to the leg end-effectors, while the manipulation force is distributed between the arm end-effectors, effectively producing a forward motion of the object. Finally, in (i), (j) Centauro returns to the initial pose

such a vector contains non-zero torques at the virtual joints, and is not directly applicable to the robot. Hence, during a second optimization stage this wrench is mapped back to the available contacts. The proposed method, in comparison to (Ott et al., 2011), has the main advantage of avoiding to constrain the robot centroidal dynamics.

The method has been tested first in simulation using two floating base robots with substantially different kinematic structure: the *Centauro* and the *Coman+* robot. Preliminary results are reported in the real *Coman+* robot which is subject to small perturbations, and a simplified architecture has been used on Centauro to implement a heavy object pushing demonstrator, which exploits contacts in order to overcome friction constraints.

Future works will address first the implementation on the real hardware which, as stated previously, at the moment presents some stability problems, very similar to the one in Englsberger et al. (2018). Second, we would like to explore a single-stage torque-force optimization, and compare it to Herzog et al. (2016). Finally, it is the author's belief that this method can be extended to other problem that can be treated by introducing virtual kinematic chains, for example when contact forces to pick a box has to be computed.

Chapter 8

Conclusions

This work of thesis has dealt with the control of a novel robotic platform, a fully torque-controlled, hybrid wheeled legged quadruped with humanoid upper body named *Centauro*. Most of this work has therefore concerned the control design, development, and implementation of loco-manipulation capabilities, with the aim to explore most of the potential of such a unique machine. The outcome of this research has been especially applied and validated on tasks related to the CENTAURO H2020 project.

8.1 Summary of main achievements

During the first period of this work, a **novel real-time robotic middleware** named *XBotCore* was developed, which has later been extensively used in all the successive contributions of this thesis, and in all our lab as well. Because the XBotCore framework is configured from standard description files such as URDF and SRDF, it was easily applied to other robots which were developed inside the *Humanoid and Human Centred Mechatronics (HHCM)* lab, such as *Walkman*, *Coman*, and the newer-generation *Coman+*, and to a *Kuka LWR* as well. The XBotCore middleware is inspired by the OROCOS framework but aims at being simpler to understand and maintain, and also at providing ready-to-use components for e.g. hardware abstraction, high-level robot and model API, ROS integration, and others.

Later on, in an effort to standardize and improve the **online Cartesian control** capabilities of Centauro, the *CartesI/O* framework was developed to be the standard Cartesian control interface for HHCM users. CartesI/O offers both (i) programmatic interfaces, allowing easy integration of a complex Cartesian controller into the client C++ code, and (ii) ROS interfaces providing the capability to interact with a remote Cartesian control solver via topics, services, and action. Furthermore, integration with XBotCore allows to run the solver

inside a real-time plugin, i.e. the part of the middleware which is closest to the hardware, and that suffers from the smallest jitter and delay. At the core of CartesI/O, different QP-based hierarchical inverse kinematic strategies are offered based on the *OpenSoT* library, which notably provides a real-time friendly implementation. Differently from the popular MoveIt framework, our work targets *online* control, as opposed to motion planning.

Based on the aforementioned achievements, **legged locomotion strategies** for quadrupeds were also studied. A novel walking gait generation scheme was proposed, which automatically optimizes future footsteps and center of mass motions, in a joint fashion, while retaining linearity of constraints. The resulting QP problem is faster to solve with respect to state of the art non-linear approaches, and its global optimum is guaranteed to be found. Experiments on real hardware have shown promising results, and demonstrated the transferability of the proposed algorithm to the actual robot. Furthermore, a novel **Cartesian controller for wheeled-legged robots** was introduced, which is based on an augmented kinematic model that also entails a virtual reference frame acting as *local world*. The presented formulation allows to fully exploit the motion potential of the robot, by freely mixing support polygon adjustments (w.r.t. the local world), local trunk control, and manipulation w.r.t. the world.

Finally, interaction controllers exploiting the robot torque control capabilities have been studied. Starting from a work which introduces a **prioritized force controller** for fixed base robots (with application to Cartesian impedance control), an extension to floating base systems was proposed, leveraging on the principle of *contact force post-optimization*. By means of this strategy, it is possible to adapt a generic torque-based controller for a fixed robot into a floating base controller. Results have been validated in simulation, whereas a simplified control architecture based on the same principles was used for environment-aware pushing of a heavy object. With respect to the state of the art, the presented method permits to decouple the control design from the actuation part, without losing the possibility to have the centroidal momentum variation to be produced automatically by the controller.

8.2 Dissemination and exploitation

The presented work of thesis has generated several outcomes, in the form of (i) scientific dissemination, (ii) software packages, and (iii) deliverables and demonstrators for a number of project beyond the CENTAURO EU. The most significant outcomes are summarized in the remainder of this section.

Articles as first author

1. Article on walking pattern generation for quadrupeds based on linear MPC (see Chapter 6). Presented at *IROS 2018*, and published to the corresponding proceedings.

   Laurenzi, A., Hoffman, E. M., and Tsagarakis, N. G. (2018b). Quadrupedal walking motion and footstep placement through linear model predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2267–2273. IEEE

2. Architectural paper on the CartesI/O framework (see Chapter 4), describing its design, implementation and validation. Presented at *ICRA 2019* and published to the corresponding proceedings.

   Laurenzi, A., Mingo Hoffman, E., Muratore, L., and Tsagarakis, N. G. (2019c). CartesI/O: A ROS Based Real-Time Capable Cartesian Control Framework. In *IEEE International Conference on Robotics and Automation (ICRA)*

3. Article on balancing control for legged robots based on post-optimization of contact forces (see Chapter 7). Presented at *Humanoids 2018*, and published to the corresponding proceedings.

   Laurenzi, A., Hoffman, E. M., Polverini, M. P., and Tsagarakis, N. G. (2018a). Balancing control through post-optimization of contact forces. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 320–326. IEEE

4. An integration work involving XBotCore, CartesI/O, and a perception pipeline was presented as short paper to the IEEE *IRC 2019* international conference, and published to the corresponding proceedings.

   Laurenzi, A., Kanoulas, D., Hoffman, E. M., Muratore, L., and Tsagarakis, N. (2019a). Whole-body stabilization for visual-based box lifting with the COMAN+ robot. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 445–446. IEEE

5. Article on the control of wheeled-legged platforms though an augmented kinematic model (see Chapter 5)

   Laurenzi, A., Mingo Hoffman, E., Parigi Polverini, M., and Tsagarakis, N. G. (2020). An Augmented Kinematic Model for the Cartesian Control of the Robot CENTAURO. *IEEE Robotics and Automation Letters*. Accepted, to appear

Articles as second author (selection)

1. Architectural article on the XBotCore framework, in its early design version.

   Muratore, L., Laurenzi, A., Hoffman, E. M., Rocchi, A., Caldwell, D. G., and Tsagarakis, N. G. (2017b). Xbotcore: A real-time cross-robot software platform. In *Robotic Computing (IRC), IEEE International Conference on*, pages 77–80. IEEE

2. Journal extension of Muratore et al. (2017b):

   Muratore, L., Laurenzi, A., Hoffman, E. M., Rocchi, A., Caldwell, D. G., and Tsagarakis, N. G. (2017a). On the design and evaluation of xbotcore, a cross-robot real-time software framework. *J. of Software Engineering for Robotics (JOSER)*, 8:164–170

3. Work on prioritized force control, with application to Cartesian impedance control (see Chapter 7):

   Mingo Hoffman, E., Laurenzi, A., Tsagarakis, N. G., and Caldwell, D. G. (2018). Multi-priority cartesian impedance control based on quadratic programming optimization. In *IEEE International Conference on Robotics and Automation, ICRA*

Other works

1. Application of the Cartesian controller presented in Chapter 5 to autonomous navigation and obstacle avoidance.

   Raghavan, V. S., Kanoulas, D., Laurenzi, A., Caldwell, D. G., and Tsagarakis, N. G. (2019). Variable Configuration Planner for Legged-Rolling Obstacle Negotiation Locomotion: Application on the CENTAURO Robot. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE

2. Application of the CartesI/O online Cartesian control framework of Chapter 4 to the robot tele-operation using cheap perception sensors

   Rolley-Parnell, E.-J., Kanoulas, D., Laurenzi, A., Delhaisse, B., Rozo, L. D., Caldwell, D. G., and Tsagarakis, N. G. (2018). Bi-manual articulated robot teleoperation using an external rgb-d range sensor. *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 298–304

3. Application of *mixed-integer* programming to the contact force post-optimization problem, presented in Chapter 7

Parigi Polverini, M., Mingo Hoffman, E., Laurenzi, A., and Tsagarakis, N. (2019). Sparse optimization of contact forces for balancing control of multi-legged humanoids. *IEEE Robotics and Automation Letters*, 4(2):1117–1124

4. Application of CartesI/O to an hybrid limit-cycle and ZMP-based strategy for humanoid walking

Ruscelli, F., Laurenzi, A., Hoffman, E. M., and Tsagarakis, N. G. (2019). Synchronizing Virtual Constraints and Preview Controller: a Walking Pattern Generator for the Humanoid Robot COMAN+. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE

5. Tele-interaction strategy based on autonomous impedance regulation

Muratore, L., Laurenzi, A., Mingo Hoffman, E., Baccelliere, L., Kashiri, N., Caldwell, D. G., and Tsagarakis, N. G. (2018). Enhanced tele-interaction in unknown environments using semi-autonomous motion and impedance regulation principles. In *IEEE International Conference on Robotics and Automation (ICRA)*

6. Centauro robot overview paper

Kashiri, N., Baccelliere, L., Muratore, L., Laurenzi, A., Ren, Z., Hoffman, E. M., et al. (2019). Centauro: A hybrid locomotion and high power resilient manipulation platform. *IEEE Robotics and Automation Letters*, 4(2):1595–1602

7. CENTAURO H2020 project overview paper

Klamt, T., Schwarz, M., Lenz, C., Baccelliere, L., Buongiorno, D., Cichon, T., DiGuardo, A., Droeschel, D., Gabardi, M., Kamedula, M., Kashiri, N., Laurenzi, A., Leonardis, D., Muratore, L., Pavlichenko, D., Periyasamy, A. S., Rodriguez, D., Solazzi, M., Frisoli, A., and Behnke, S. (2019). Remote mobile manipulation with the centauro robot: Full body telepresence and autonomous operator assistance. *Journal of Field Robotics*

Tutorials, workshops and media

- the XBotCore framework, along with the OpenSoT and CartesI/O libraries were presented during a tutorial session of the *IROS 2018* international conference, named *"A hands-on tutorial on XBotCore: a real-time cross-robot and cross-framework software architecture"* [1]; this tutorial was co-organized by the Author.

[1] XBotCore tutorial web page available at https://xbotcoretutorial.weebly.com/

- The demonstrator of (Laurenzi et al., 2019b) was showcased during the presentation event of the COMAN+ humanoid robot, at the Cogimon European Project booth at *ICRA 2019*[2].
- Multiple demonstrators based on the works of Chapter 4 were showcased during the *MARS 2019* private event held by Amazon during March 2019.
- Multiple demonstrators based on the works of Chapter 4 were part of the official presentation video of the Centauro robot[3]

Software packages

The *XBotControl* package, containing the *XBotCore*, *OpenSoT* and *CartesI/O* libraries has been released[4].

Funded projects

The scientific and technological outcomes of this work have contributed to the successful results of the following funded projects:

- CENTAURO[5] (H2020 project);
- CogIMon[6] (H2020 project)
- Pholus[7] (Italian Ministry of Defense project)

8.3 Future works

The present thesis has dealt with a variety of problems that arose while developing a control system for a modern, torque-controlled legged robot which leverages on a novel, hybrid locomotion concept, based on both wheeled maneuvers and stepping motions. Most of the designed strategies have lead to successful experimental implementations and validations, leaving for future work a more complete interaction and integration among the control

[2]https://cogimon.eu/cogimon-icra-exhibition
[3]The Centauro presentation video is available at https://www.youtube.com/watch?v=L7JssknlCvw
[4]https://github.com/ADVRHumanoids/XBotControl
[5]https://www.centauro-project.eu/
[6]https://www.cogimon-project.eu/
[7]https://www.difesa.it/Amministrazionetrasparente/segredifesa/navarm/Documents/7_Divisione/DetermineAContrarre/20151028_55_PHOLUS.pdf

system components. To this aim, it will be necessary to incorporate the vision system inside the control loop; indeed, the variety of tasks that can be accomplished by a blind robot is limited, no matter how complex and flexible the platform is. As a result, the Centauro robot will be able to tackle more real-world tasks such as (i) dynamically overcome small perceived obstacles with hybrid driving and stepping; (ii) stepping over complex terrain (e.g. debris) while computing optimal contact forces in order e.g. to avoid slippage; (iii) improve manipulation precision with online visual servoing; and others.

In addition to the integration between core components, all specific areas to which this thesis has contributed must be considered as *ongoing projects*, rather than completed activities. Some ideas for future contributions follow:

- The XBotCore middleware is designed to be compatible with different robotic platforms using different fieldbus (e.g. CAN, Ethernet, ...), through the hardware abstraction layer component. However, if a custom component is added to the robot (e.g. a foot pressure sensor, a gripper, ...), several manual steps are required in order to make it available at all the architectural layers (real-time, ROS, ...), which usually require altering some parts of the framework source code. As a future work, a plugin architecture for custom devices could be designed, in order to better decouple the framework from the specific robot components.

- The single-process design of the XBotCore middleware can be regarded as a weak point, as user-defined real-time plugins, in the event of a failure, can crash the whole system causing the loss of communication with the robot. To improve the system robustness, a client-server multi-process scheme can be adopted, which exploits the level of process insulation guaranteed by the operating system. Futhermore, the client could even be run on a remote machine.

- Our Cartesian control framework *CartesI/O* could be extended to better support torque-based controllers, by also providing interfaces to quantities related to interaction tasks, such as forces and impedance levels.

- Prioritized online Cartesian control as implemented by CartesI/O could be exploited in order to design a *sampling-based* motion planner for floating base systems, a capability which currently is missing even from the popular *MoveIt!* framework.

- Walking pattern generation and compliant controllers could be merged, in order to make the robot able to traverse complex terrains. Optimization of contact forces applied to the legged locomotion would be beneficial as well. Finally, the Cartesian

controller of Chapter 5 could be employed to implement a mixed wheeled-legged locomotion strategy.

- Concerning interaction control, further investigation will be needed in order to have a robust implementation. To this aim, a reliable implementation of a floating-base state estimation module will be needed. This will ultimately allow to go beyond the simplified architecture of our heavy object pushing demonstrator, and fully exploit prioritized, Cartesian space compliance.

As last remark, only limited attention has been devoted in this work to the matter of closed loop robustness of the designed strategies w.r.t. to real world, non-ideal scenarios. Despite most of the achieved results have been experimentally validated in the laboratory, or in other simplified settings, there is a need to implement more effective feedback strategies in order to cope with the uncertainties of a fully unstructured environment.

References

Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., and Yoon, W.-K. (2005). Rt-middleware: distributed component middleware for rt (robot technology). In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3933–3938. IEEE.

Baccelliere, L. et al. (2017a). Development of a human size and strength compliant bi-manual platform for realistic heavy manipulation tasks. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 5594–5601. IEEE.

Baccelliere, L., Kashiri, N., Muratore, L., Laurenzi, A., Kamedula, M., Margan, A., Cordasco, S., Malzahn, J., and Tsagarakis, N. G. (2017b). Development of a human size and strength compliant bi-manual platform for realistic heavy manipulation tasks. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, pages 5594–5601.

Baerlocher, P. and Boulic, R. (1998). Task-priority formulations for the kinematic control of highly redundant articulated structures. In *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, volume 1, pages 323–329 vol.1.

Barasuol, V., Buchli, J., Semini, C., Frigerio, M., De Pieri, E. R., and Caldwell, D. G. (2013). A reactive controller framework for quadrupedal locomotion on challenging terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2554–2561.

Bischoff, R., Kurth, J., Schreiber, G., Koeppe, R., Albu-Schäffer, A., Beyer, A., Eiberger, O., Haddadin, S., Stemmer, A., Grunwald, G., et al. (2010). The kuka-dlr lightweight robot arm-a new reference platform for robotics research and manufacturing. In *Robotics (ISR), 2010 41st international symposium on and 2010 6th German conference on robotics (ROBOTIK)*, pages 1–8. VDE.

Bjelonic, M., Bellicoso, C. D., de Viragh, Y., Sako, D., Tresoldi, F. D., Jenelten, F., and Hutter, M. (2019). Keep rollin'—whole-body motion control and planning for wheeled quadrupedal robots. *IEEE Robotics and Automation Letters*, 4(2):2116–2123.

Brown, J. H. (2012). How fast is fast enough? choosing between xenomai and linux for real-time applications. *Twelfth Real-Time Linux Workshop*.

Bruyninckx, H. (2002). OROCOS: design and implementation of a robot control software framework. *Proc. IEEE RAS EMBS Int. Conf. Biomed. Robot. Biomechatron.*

Buchli, J., Kalakrishnan, M., Mistry, M., Pastor, P., and Schaal, S. (2009). Compliant quadruped locomotion over rough terrain. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 814–820. IEEE.

Caron, S. and Nakamura, Y. (2015). Teleoperation system design of valve turning motions in degraded communication conditions. *The 33-rd Annual Conference of the RSJ*.

Chitta, S., Sucan, I., and Cousins, S. (2012). Moveit! *IEEE Robotics & Automation Magazine*, 19(1):18–19.

Dehio, N., Smith, J., Wigand, D. L., Xin, G., Lin, H., Steil, J. J., and Mistry, M. (2018). Modeling and control of multi-arm and multi-leg robots: Compensating for object dynamics during grasping. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 294–301.

Deits, R. and Tedrake, R. (2014). Footstep planning on uneven terrain with mixed-integer convex optimization. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 279–286.

Del Prete, A. (2018). Joint position and velocity bounds in discrete-time acceleration/torque control of robot manipulators. *IEEE Robotics and Automation Letters*, 3(1):281–288.

Diankov, R. (2010). *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute.

Diedam, H., Dimitrov, D., Wieber, P.-B., Mombaur, K., and Diehl, M. (2008). Online walking gait generation with adaptive foot positioning through linear model predictive control. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1121–1126. IEEE.

Elkady, A. and Sobh, T. (2012). Robotics middleware: A comprehensive literature survey and Attribute-Based bibliography. *Journal of Robotics*, 2012.

Englsberger, J., Mesesan, G., Werner, A., and Ott, C. (2018). Torque-based dynamic walking - A long way from simulation to experiment. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*.

Englsberger, J., Ott, C., Roa, M. A., Albu-Schäffer, A., and Hirzinger, G. (2011). Bipedal walking control based on capture point dynamics. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4420–4427.

Escande, A., Mansard, N., and Wieber, P.-B. (2014a). Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028.

Escande, A., Mansard, N., and Wieber, P.-B. (2014b). Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028.

Escande, A., Mansard, N., and Wieber, P.-B. (2014c). Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028.

Fallon, M., Kuindersma, S., Karumanchi, S., Antone, M., Schneider, T., Dai, H., D'Arpino, C. P., Deits, R., DiCicco, M., Fourie, D., et al. (2015). An architecture for online affordance-based perception and whole-body planning. *Journal of Field Robotics*, 32(2):229–254.

Featherstone, R. (2010). A beginner's guide to 6-d vectors (part 1). *IEEE Robotics Automation Magazine*, 17(3):83–94.

Featherstone, R. (2010). A beginner's guide to 6d vectors (part 2). *IEEE Robotics Automation Magazine*, 17:88–99.

Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., and Diehl, M. (2014a). Qpoases: a parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363.

Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., and Diehl, M. (2014b). qpOASES: a parametric active-set algorithm for quadratic programming. *Math. Program. Comput.*, 6(4):327–363.

Flacco, F. and De Luca, A. (2015). Discrete-time redundancy resolution at the velocity level with acceleration/torque optimization properties. *Robotics and Autonomous Systems*, 70:191–201.

Flacco, F., Luca, A. D., and Khatib, O. (2015). Control of redundant robots under hard joint constraints: Saturation in the null space. *IEEE Transactions on Robotics*, 31:637–654.

Geilinger, M., Poranne, R., Desai, R., Thomaszewski, B., and Coros, S. (2018). Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels. *ACM Transactions on Graphics (TOG)*, 37(4):160.

Gossow, D., Leeper, A., Hershberger, D., and Ciocarlie, M. T. (2011). Interactive markers: 3-d user interfaces for ros applications [ros topics]. *IEEE Robot. Automat. Mag.*, 18:14–15.

Henze, B., Dietrich, A., and Ott, C. (2016a). An approach to combine balancing with hierarchical whole-body control for legged humanoid robots. *IEEE Robotics and Automation Letters*, 1:700–707.

Henze, B., Dietrich, A., and Ott, C. (2016b). An approach to combine balancing with hierarchical whole-body control for legged humanoid robots. *IEEE Robotics and Automation Letters*, 1(2):700–707.

Henze, B., Dietrich, A., Roa, M. A., and Ott, C. (2017). Multi-contact balancing of humanoid robots in confined spaces: Utilizing knee contacts. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 697–704. IEEE.

Herdt, A., Diedam, H., Wieber, P.-B., Dimitrov, D., Mombaur, K., and Diehl, M. (2010a). Online walking motion generation with automatic footstep placement. *Advanced Robotics*, 24(5-6):719–737.

Herdt, A., Perrin, N., and Wieber, P.-B. (2010b). Walking without thinking about it. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 190–195. IEEE.

Herzog, A., Righetti, L., Grimminger, F., Pastor, P., and Schaal, S. (2014). Balancing experiments on a torque-controlled humanoid with hierarchical inverse dynamics. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 981–988. IEEE.

Herzog, A., Rotella, N., Mason, S., Grimminger, F., Schaal, S., and Righetti, L. (2016). Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. *Autonomous Robots*, 40(3):473–491.

Hoffman, E. M., Laurenzi, A., Muratore, L., Caldwell, D. G., and Tsagarakis, N. G. (2018). Multi-priority cartesian impedance control based on quadratic programming optimization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*.

Hoffman, E. M., Rocchi, A., Laurenzi, A., and Tsagarakis, N. G. (2017). Robot control for dummies: Insights and examples using opensot. In *17th IEEE-RAS International Conference on Humanoid Robotics, 2017*, pages 736–741.

Hogan, N. (1985). Impedance control - An approach to manipulation. I - Theory. II - Implementation. III - Applications. *ASME Transactions Journal of Dynamic Systems and Measurement Control B*, 107:1–24.

Houliston, T., Fountain, J., Lin, Y., Mendes, A., and others (2016). NUClear: A loosely coupled software architecture for humanoid robot systems. *Frontiers in Robotics*.

Hyon, S.-H. (2009). Compliant terrain adaptation for biped humanoids without measuring ground surface and contact forces. *IEEE Transactions on Robotics*, 25(1):171–178.

Hyon, S.-H., Hale, J. G., Cheng, G., et al. (2007). Full-body compliant human-humanoid interaction: Balancing in the presence of unknown external forces. *IEEE Trans. Robotics*, 23(5):884–898.

Jeongsoo, L., Jungho, L., and Jun-Ho, O. (2014). Development of robot software framework podo: Toward multi-processes and multi-users. *Workshop on software architectures and methodologies for developing humanoid robots, IEEE HUMANOIDS 2014*.

Johnson, M., Shrewsbury, B., Bertrand, S., Wu, T., Duran, D., Floyd, M., Abeles, P., Stephen, D., Mertins, N., Lesman, A., Carff, J., Rifenburgh, W., Kaveti, P., Straatman, W., Smith, J., Griffioen, M., Layton, B., de Boer, T., Koolen, T., Neuhaus, P., and Pratt, J. (2015). Team IHMC's lessons learned from the DARPA robotics challenge trials. *J. Field Robotics*, 32(2):192–208.

Kajita, S., Hirukawa, H., Harada, K., and Yokoi, K. (2014). *Introduction to humanoid robotics*, volume 101. Springer.

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa, H. (2003). Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1620–1626. IEEE.

Kamedula, M., Kashiri, N., and Tsagarakis, N. G. (2018). On the Kinematics of Wheeled Motion Control of a Hybrid Wheeled-Legged CENTAURO robot. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2426–2433.

Kanoun, O., Lamiraux, F., and Wieber, P.-B. (2011). Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, 27(4):785–792.

Kashiri, N., Baccelliere, L., Muratore, L., Laurenzi, A., Ren, Z., Hoffman, E. M., et al. (2019). Centauro: A hybrid locomotion and high power resilient manipulation platform. *IEEE Robotics and Automation Letters*, 4(2):1595–1602.

Kashiri, N., Baccelliere, L., Muratore, L., Laurenzi, A., Ren, Z., Hoffman, E. M., Kamedula, M., Rigano, G. F., Malzahn, J., Cordasco, S., Guria, P., Margan, A., and Tsagarakis, N. G. (2019). Centauro: A hybrid locomotion and high power resilient manipulation platform. *IEEE Robotics and Automation Letters*, 4(2):1595–1602.

Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53.

Khatib, O., Sentis, L., Park, J., and Warren, J. (2004). Whole-body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics*, 1(01):29–43.

Klamt, T., Schwarz, M., Lenz, C., Baccelliere, L., Buongiorno, D., Cichon, T., DiGuardo, A., Droeschel, D., Gabardi, M., Kamedula, M., Kashiri, N., Laurenzi, A., Leonardis, D., Muratore, L., Pavlichenko, D., Periyasamy, A. S., Rodriguez, D., Solazzi, M., Frisoli, A., and Behnke, S. (2019). Remote mobile manipulation with the centauro robot: Full body telepresence and autonomous operator assistance. *Journal of Field Robotics*.

Koolen, T., de Boer, T., Rebula, J., Goswami, A., and Pratt, J. (2012). Capturability-based analysis and control of legged locomotion, part 1: Theory and application to three simple gait models. *The International Journal of Robotics Research*, 31(9):1094–1113.

Kröger, T. (2011). Opening the door to new sensor-based robot applications—the reflexxes motion libraries. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE.

Lanari, L., Hutchinson, S., and Marchionni, L. (2014). Boundedness issues in planning of locomotion trajectories for biped robots. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 951–958. IEEE.

Laurenzi, A., Hoffman, E. M., Polverini, M. P., and Tsagarakis, N. G. (2018a). Balancing control through post-optimization of contact forces. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 320–326. IEEE.

Laurenzi, A., Hoffman, E. M., and Tsagarakis, N. G. (2018b). Quadrupedal walking motion and footstep placement through linear model predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2267–2273. IEEE.

Laurenzi, A., Kanoulas, D., Hoffman, E. M., Muratore, L., and Tsagarakis, N. (2019a). Whole-body stabilization for visual-based box lifting with the COMAN+ robot. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 445–446. IEEE.

Laurenzi, A., Kanoulas, D., Mingo Hoffman, E., Muratore, L., and Tsagarakis, N. (2019b). Whole-Body Stabilization for Visual-based Box Lifting with the COMAN+ Robot.

Laurenzi, A., Mingo Hoffman, E., Muratore, L., and Tsagarakis, N. G. (2019c). CartesI/O: A ROS Based Real-Time Capable Cartesian Control Framework. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Laurenzi, A., Mingo Hoffman, E., Parigi Polverini, M., and Tsagarakis, N. G. (2020). An Augmented Kinematic Model for the Cartesian Control of the Robot CENTAURO. *IEEE Robotics and Automation Letters*. Accepted, to appear.

Lee, S.-H. and Goswami, A. (2010). Ground reaction force control at each foot: A momentum-based humanoid balance controller for non-level and non-stationary ground. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3157–3162. IEEE.

Lee, S.-H. and Goswami, A. (2012). A momentum-based balance controller for humanoid robots on non-level and non-stationary ground. *Autonomous Robots*, 33(4):399–414.

Lim, J., Lee, I., Shim, I., Jung, H., Joe, H. M., Bae, H., Sim, O., Oh, J., Jung, T., Shin, S., Joo, K., Kim, M., Lee, K., Bok, Y., Choi, D.-G., Cho, B., Kim, S., Heo, J., Kim, I., Lee, J., Kwon, I. S., and Oh, J.-H. (2017). Robot System of DRC-HUBO+ and Control Strategy of Team KAIST in DARPA Robotics Challenge Finals: Robot System of DRC-HUBO+ and Control Strategy of Team KAIST. *Journal of Field Robotics*, 34(4):802–829.

Liu, M., Tan, Y., and Padois, V. (2015). Generalized hierarchical control. *Auton. Robots*, pages 1–15.

Ma, S., Tomiyama, T., and Wada, H. (2005). Omnidirectional static walking of a quadruped robot. *IEEE Transactions on Robotics*, 21(2):152–161.

Maciejewski, A. and Klein, C. (1985). Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research*, 4.

Mansard, N. (2012). A dedicated solver for fast operational-space inverse dynamics. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4943–4949. IEEE.

McGhee, R. B. and Frank, A. A. (1968). On the stability properties of quadruped creeping gaits. *Mathematical Biosciences*, 3:331–351.

Metta, G., Fitzpatrick, P., and Natale, L. (2006). Yarp: Yet another robot platform. *International Journal on Advanced Robotics Systems*.

Mingo Hoffman, E., Laurenzi, A., Tsagarakis, N. G., and Caldwell, D. G. (2018). Multi-priority cartesian impedance control based on quadratic programming optimization. In *IEEE International Conference on Robotics and Automation, ICRA*.

Mingo Hoffman, E., Rocchi, A., G. Tsagarakis, N., and G. Caldwell, D. (2015). Opensot: a whole-body control library for the compliant humanoidrobot coman. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6248–6253.

Mingo Hoffman, E., Rocchi, A., Laurenzi, A., and Tsagarakis, N. G. (2017). Robot control for dummies: Insights and examples using opensot. In *17th IEEE-RAS International Conference on Humanoid Robots, Humanoids*, pages 736–741.

Muratore, L., Laurenzi, A., Hoffman, E. M., Rocchi, A., Caldwell, D. G., and Tsagarakis, N. G. (2017a). On the design and evaluation of xbotcore, a cross-robot real-time software framework. *J. of Software Engineering for Robotics (JOSER)*, 8:164–170.

Muratore, L., Laurenzi, A., Hoffman, E. M., Rocchi, A., Caldwell, D. G., and Tsagarakis, N. G. (2017b). Xbotcore: A real-time cross-robot software platform. In *Robotic Computing (IRC), IEEE International Conference on*, pages 77–80. IEEE.

Muratore, L., Laurenzi, A., Hoffman, E. M., Rocchi, A., Caldwell, D. G., and Tsagarakis, N. G. (2017c). Xbotcore: A real-time cross-robot software platform. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 77–80.

Muratore, L., Laurenzi, A., Hoffman, E. M., Rocchi, A., Caldwell, D. G., and Tsagarakis, N. G. (2017d). Xbotcore: A real-time cross-robot software platform. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 77–80.

Muratore, L., Laurenzi, A., Mingo Hoffman, E., Baccelliere, L., Kashiri, N., Caldwell, D. G., and Tsagarakis, N. G. (2018). Enhanced tele-interaction in unknown environments using semi-autonomous motion and impedance regulation principles. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Murphy, R. R. (2014). *Disaster Robotics*. The MIT Press.

Murphy, R. R., Tadokoro, S., and Kleiner, A. (2016). *Disaster Robotics*, pages 1577–1604. Springer International Publishing, Cham.

Nakanishi, J., Cory, R., Mistry, M., Peters, J., and Schaal, S. (2008). Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757.

Nakaoka, S., Kajita, S., and Yokoi, K. (2010). Intuitive and flexible user interface for creating whole body motions of biped humanoid robots. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1675–1682. IEEE.

Natale, L., Paikan, A., Randazzo, M., and Domenichelli, D. E. (2016). The icub software architecture: Evolution and lessons learned. *Frontiers in Robotics and AI*, 3:24.

Neunert, M., Farshidian, F., Winkler, A. W., and Buchli, J. (2017). Trajectory optimization through contacts and automatic gait discovery for quadrupeds. *IEEE Robotics and Automation Letters*, 2(3):1502–1509.

Ott, C. (2008). *Cartesian impedance control of redundant and flexible-joint robots*. Springer tracts in advanced robotics. Springer, Berlin.

Ott, C., Roa, M. A., and Hirzinger, G. (2011). Posture and balance control for biped robots based on contact force optimization. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 26–33. IEEE.

Parigi Polverini, M., Mingo Hoffman, E., Laurenzi, A., and Tsagarakis, N. (2019). Sparse optimization of contact forces for balancing control of multi-legged humanoids. *IEEE Robotics and Automation Letters*, 4(2):1117–1124.

Platt Jr, R., Abdallah, M. E., and Wampler, C. W. (2010). Multi-priority cartesian impedance control. In *Robotics: Science and Systems*.

Pongas, D., Mistry, M., and Schaal, S. (2007). A robust quadruped walking gait for traversing rough terrain. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1474–1479. IEEE.

Ponton, B., Herzog, A., Schaal, S., and Righetti, L. (2016). A convex model of humanoid momentum dynamics for multi-contact motion generation. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 842–849.

Pratt, G. and Williamson, M. (1995). Series elastic actuators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 1, pages 399–406.

Pratt, J., Chew, C.-M., Torres, A., Dilworth, P., and Pratt, G. (2001). Virtual model control: An intuitive approach for bipedal locomotion. *The International Journal of Robotics Research*, 20(2):129–143.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009a). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009b). ROS: an open-source Robot Operating System. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan.

Raghavan, V. S., Kanoulas, D., Laurenzi, A., Caldwell, D. G., and Tsagarakis, N. G. (2019). Variable Configuration Planner for Legged-Rolling Obstacle Negotiation Locomotion: Application on the CENTAURO Robot. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Rigano, G. F., Muratore, L., Laurenzi, A., Hoffman, E. M., and Tsagarakis, N. G. (2018). A mixed real-time robot hardware abstraction layer (r-hal). *Encyclopedia with Semantic Computing and Robotic Intelligence*.

Righetti, L., Buchli, J., Mistry, M., Kalakrishnan, M., and Schaal, S. (2013). Optimal distribution of contact forces with inverse-dynamics control. *The International Journal of Robotics Research*, 32(3):280–298.

Righetti, L., Buchli, J., Mistry, M., and Schaal, S. (2011). Inverse dynamics control of floating-base robots with external constraints: A unified view. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1085–1090. IEEE.

Rolley-Parnell, E.-J., Kanoulas, D., Laurenzi, A., Delhaisse, B., Rozo, L. D., Caldwell, D. G., and Tsagarakis, N. G. (2018). Bi-manual articulated robot teleoperation using an external rgb-d range sensor. *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 298–304.

Ruscelli, F., Laurenzi, A., Hoffman, E. M., and Tsagarakis, N. G. (2019). Synchronizing Virtual Constraints and Preview Controller: a Walking Pattern Generator for the Humanoid Robot COMAN+. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Saab, L., Ramos, O. E., Keith, F., Mansard, N., Soueres, P., and Fourquet, J.-Y. (2013a). Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics*, 29(2):346–362.

Saab, L., Ramos, O. E., Keith, F., Mansard, N., Soueres, P., and Fourquet, J.-Y. (2013b). Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *IEEE Transactions on Robotics*, 29(2):346–362.

Saccon, A., Traversaro, S., Nori, F., and Nijmeijer, H. (2017). On centroidal dynamics and integrability of average angular velocity. *IEEE Robotics and Automation Letters*, 2(2):943–950.

Schwarz, M., Rodehutskors, T., et al. (2016). Nimbro rescue: Solving disaster-response tasks with the mobile manipulation robot momaro. *Journal of Field Robotics*, 34(2):400–425.

Schwarz, M., Rodehutskors, T., Schreiber, M., and Behnke, S. (2016). Hybrid driving-stepping locomotion with the wheeled-legged robot momaro. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5589–5595.

Scianca, N., Cognetti, M., De Simone, D., Lanari, L., and Oriolo, G. (2016). Intrinsically stable mpc for humanoid gait generation. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 601–606. IEEE.

Sentis, L. and Khatib, O. (2005). Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(04):505–518.

Sentis, L., Park, J., and Khatib, O. (2010). Compliant control of multicontact and center-of-mass behaviors in humanoid robots. *IEEE Transactions on robotics*, 26(3):483–501.

Siciliano, B., Sciavicco, L., and Villani, L. (2009). *Robotics : modelling, planning and control*. Advanced Textbooks in Control and Signal Processing. Springer, London. 013-81159.

Siciliano, B. and Slotine, J. J. (1991). A general framework for managing multiple tasks in highly redundant robotic systems. *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*, pages 1211–1216 vol.2.

Smith, J., Stephen, D., Lesman, A., and Pratt, J. (2014). Real-time control of humanoid robots using openjdk. In *Proceedings of the 12th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '14, pages 29:29–29:36, New York, NY, USA. ACM.

Sreenath, K., Park, H.-W., Poulakakis, I., and Grizzle, J. W. (2011). A compliant hybrid zero dynamics controller for stable, efficient and fast bipedal walking on mabel. *The International Journal of Robotics Research*, 30(9):1170–1193.

Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. (2017). OSQP: An operator splitting solver for quadratic programs. *ArXiv e-prints*.

Stephens, B. J. and Atkeson, C. G. (2010). Dynamic balance force control for compliant humanoid robots. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1248–1255. IEEE.

Şucan, I. A., Moll, M., and Kavraki, L. E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82. http://ompl.kavrakilab.org.

Tahara, K., Arimoto, S., and Yoshida, M. (2010). Dynamic object manipulation using a virtual frame by a triple soft-fingered robotic hand. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4322–4327.

Tsagarakis, N. G., Caldwell, D. G., et al. (2017a). Walkman: A high performance humanoid platform for realistic environments. *Journal of Field Robotics*, 34(7):1225–1259.

Tsagarakis, N. G., Caldwell, D. G., Negrello, F., Choi, W., Baccelliere, L., Loc, V., Noorden, J., Muratore, L., Margan, A., Cardellino, A., et al. (2017b). Walk-man: A high-performance humanoid platform for realistic environments. *Journal of Field Robotics*, 34(7):1225–1259.

Tsagarakis, N. G., Morfey, S., Cerda, G. M., Zhibin, L., and Caldwell, D. G. (2013). Compliant humanoid coman: Optimal joint stiffness tuning for modal frequency control. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 673–678. IEEE.

Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57.

Wang, Y., Smith, C., Karayiannidis, Y., and Ögren, P. (2015). Cooperative control of a serial-to-parallel structure using a virtual kinematic chain in a mobile dual-arm manipulation application. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2372–2379.

Wieber, P.-B. (2002). On the stability of walking systems. In *Proceedings of the International Workshop on Humanoid and Human Friendly Robotics*, Tsukuba, Japan.

Wieber, P.-B. (2006a). *Holonomy and Nonholonomy in the Dynamics of Articulated Motion*, pages 411–425. Springer Berlin Heidelberg, Berlin, Heidelberg.

Wieber, P.-B. (2006b). Holonomy and nonholonomy in the dynamics of articulated motion. In *Fast motions in biomechanics and robotics*, pages 411–425. Springer.

Wieber, P.-B. (2006c). Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 137–142. IEEE.

Wieber, P.-B. (2008). Viability and predictive control for safe locomotion. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1103–1108. IEEE.

Wieber, P.-B., Tedrake, R., and Kuindersma, S. (2016). Modeling and control of legged robots. *Springer Handbook of Robotics*, pages 1203–1234.

Winkler, A. W. (2018). Ifopt - A modern, light-weight, Eigen-based C++ interface to Nonlinear Programming solvers Ipopt and Snopt.

Winkler, A. W., Bellicoso, C. D., Hutter, M., and Buchli, J. (2018). Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*.

Winkler, A. W., Farshidian, F., Neunert, M., Pardo, D., and Buchli, J. (2017). Online walking motion and foothold optimization for quadruped locomotion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5308–5313. IEEE.

Winkler, A. W., Mastalli, C., Havoutis, I., Focchi, M., Caldwell, D. G., and Semini, C. (2015). Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5148–5154.

Wu, Q., Liu, C., Zhang, J., and Chen, Q. (2009). Survey of locomotion control of legged robots inspired by biological concept. *Science in China Series F: Information Sciences*, 52(10):1715–1729.

Zhou, C., Li, Z., Wang, X., Tsagarakis, N., and Caldwell, D. (2016). Stabilization of bipedal walking based on compliance control. *Autonomous Robots*, 40(6):1041–1057.