UNIVERSITY OF GENOVA

PHD PROGRAM IN BIOENGINEERING AND ROBOTICS

Support polygon in the hybrid legged-wheeled CENTAURO robot: modelling and control

by

Małgorzata Katarzyna KAMEDUŁA

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

December 2019

Nikos G. Tsagarakis	Supervisor
Giorgio Cannata	Head of the PhD program

Thesis Jury:

Marilena Vendittelli, *Sapienza Università di Roma*	External examiner
Amir Jafari, *University of Texas*	External examiner

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.

Małgorzata Katarzyna Kameduła
January 2020

Acknowledgements

I would like to thank everyone involved with the Humanoids and Human Centered Machatronics lab, the IIT's Advanced Robotics department and the CENTAURO project. This thesis would not be possible without all of the support and knowledge shared with me during the past four years.

I would like to specifically thank professor Nikos Tsagarakis for his continuous support, guidance and stimulating discussions as well as professors Darwin Caldwell and Sven Behnke for their advice and genuine interest in this work outcome.

Furthermore, I would like to thank the technicians Paolo Guria, Lorenzo Baccelliere and Diego Vedelago for their timely and indispensable support with hardware. Without their work none of the experiments presented in this work would be possible. I would like to also thank Riccardo Sepe for his support in times of the computer and software malfunctions.

Special thanks belongs to Vishnu Dev Amara and Brian Delhaisse for their friendly support, broad knowledge and time whenever asked for. I would like to also thank Giuseppe Rigano for his support with the lower-level framework and experiments.

Finally, I would like to thank Navvab Kashiri for his advice and work on writing publications and Rajesh Subburaman, Xinyuan Zhao, Domingo Esteban and Yangwei You for their interest in my work and friendly, helpful discussions. Last but not least, I would like to thank Edwin Avila and Octavio Magana for their tranquil presence and advice.

This work has been funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644839 (CENTAURO).

Abstract

Search for the robot capable to preform well in the real-world has sparked an interest in the hybrid locomotion systems. The Hybrid Legged-Wheeled (HLW) robots combine the advantages of the standard legged and wheeled platforms by switching between the quick and efficient wheeled motion on the flat grounds and the more versatile legged mobility on the unstructured terrains. With the locomotion flexibility offered by the hybrid mobility and appropriate control tools, these systems have high potential to excel in practical applications adapting effectively to real world during locomanipuation operations.

In contrary to their standard well-studied counterparts, kinematics of this newer type of robotic platforms has not been fully understood yet. This gap may lead to unexpected results when the standard locomotion methods are applied to HLW robots. To better understand mobility of the HLW robots, the model that describes the support polygon of a general HLW robot as a function of the wheel angular velocities without assumptions on the robot kinematics or wheel camber angle is proposed and analysed in this thesis.

Based on the analysis of the developed support polygon model, a robust omnidirectional driving scheme has been designed. A continuous wheel motion is resolved through the Inverse Kinematics (IK) scheme, which generates robot motion compliant with the Non-Sliding Pure-Rolling (NSPR) condition. A higher-level scheme resolving a steering motion to comply with the non-holonomic constraint and to tackle the structural singularity is proposed.

To improve the robot performance in presence to the unpredicted circumstances, the IK scheme has been enhanced with the introduction of a new reactive support polygon adaptation task. To this end, a novel quadratic programming task has been designed to push the system Support Polygon Vertices (SPVs) away from the robot Centre of Mass (CoM), while respecting the leg workspace limits. The proposed task has been expressed through the developed SPV model to account for the hardware limits.

The omnidirectional driving and reactive control schemes have been verified in the simulation and in the hardware experiments. To that end, the simulator for the CENTAURO robot that models the actuation dynamics and the software framework for the locomotion research have been developed.

Table of contents

List of figures

List of tables

List of source codes

List of acronyms

CoM	Centre of Mass.
CoP	Centre of Pressure.
DoF	Degree of Freedom.
GUI	Graphical User Interface.
HLW	Hybrid Legged-Wheeled.
ICR	Instantaneous Centre of Rotation.
ID	Inverse Dynamics.
IK	Inverse Kinematics.
IMU	Inertial Measurement Unit.
MPC	Model Predictive Control.
NSPR	Non-Sliding Pure-Rolling.
SEA	Series Elastic Actuator.
SOL-FER	Simplifying Operations in Locomotion - Framework for Efficient Research.
SPV	Support Polygon Vertex.
SRDF	Semantic Robot Description Format.
SWMR	Standard Wheeled Mobile Robot.
URDF	Unified Robot Description Format.
ZMP	Zero Moment Point.

Notes on Notation

General Notation

Variable at 't' time step is indicated through $(.)_{|t}$.

Vector Notation

A symbol ${}^{A}\boldsymbol{x}_{B}$ denotes a vector from point A to point B. If the frame in which the vector is expressed affects the discussion/equation, a frame is given as a right side superscript in the parenthesis; for a frame $\mathscr{F}_C$ that reads ${}^{A}\boldsymbol{x}_{B}^{(C)}$. Otherwise, the frame notation is dropped. x, y, z coordinates of a vector ${}^{A}\boldsymbol{x}_{B}$ read ${}^{A}x_{B}$, ${}^{A}y_{B}$, ${}^{A}z_{B}$, respectively.

Vector decomposition

A decomposition of a 3D vector ($\boldsymbol{x} \in \Re^3$) is noted as

$$\boldsymbol{x} = \boldsymbol{x}_{|D} + \boldsymbol{x}_{|P_D},$$

where $\boldsymbol{x}_{|D} \in \Re^3$ and $\boldsymbol{x}_{|P_D} \in \Re^3$ represent – in the original 3D space – the vector component along the vector $\boldsymbol{D}$ and its complement, respectively. Furthermore, $\underline{x}_{|D} \in \Re^1$ and $\underline{x}_{|PD} \in \Re^2$ symbolise the vector component along the vector $\boldsymbol{D}$ and its complement in their respective subspaces, that reads

$$\boldsymbol{x}_{|D} = \boldsymbol{D}\underline{x}_{|D},$$

$$\boldsymbol{x}_{|PD} = \boldsymbol{M}_{D}\underline{x}_{|PD},$$

where $\boldsymbol{M}_D \in \Re^{3\times 2}$ denotes a projection matrix from the null-space of the vector $\boldsymbol{D}$ to the full 3D space.

Combined Notation

If multiple supscripts are required, they are concatenated. For example, ${}^{A}\boldsymbol{x}^{(C)}_{B|n|t}$ represents a component along the ground normal of a vector from a point A to a point B expressed in a frame $\mathscr{F}_C$ a time step t.

Rotation

If $\theta \in \Re^1$ marks an orientation angle, then $\boldsymbol{R}_a(\theta) \in SO^3$ denotes a 3×3 rotation matrix that describes a rotation around axis 'a' by an angle θ, and $\boldsymbol{r}_a(\theta) \in SO^2$ denotes a 2×2 rotation matrix that describes a rotation around axis 'a' by an angle θ in the null-space of the vector 'a'.

Assumptions

Without loss of generality, it is assumed the robot has $p \in \mathbb{N}$ legs, set $\mathbb{F}$ includes all legs in a ground contact and $|\mathbb{F}| = f \leq p$. Furthermore, each leg is characterised by $n_{leg} \in \mathbb{N}$ DoFs, the vector of actuated joints $\boldsymbol{q}_a$ is organized so that i_{th} leg coordinates $\boldsymbol{q}_i$ are grouped, i.e. $\boldsymbol{q}_a = [\boldsymbol{q}_1^T \quad \boldsymbol{q}_2^T \quad \dots \quad \boldsymbol{q}_p^T]^T$. The robot base link, refereed as 'pelvis' in this work, is chosen in a way that the vector from an inertial frame to the leg contact point depends solely on the floating base and i_{th} leg coordinates.

Finally, it is assumed the standard rigid body model is known, and so is the kinematic state of robot links. Therefore, no details on derivation of position/velocity vectors and respective Jacobians for points rigidly attached to the robot links are given throughout the work with exception of the short theoretical background on the velocity and acceleration of the point on the rigid-body in the section Section 3.

Chapter 1

Introduction

In the past years, robots have been entering the industry outperforming humans in the repeatable, high-precision manufacturing tasks. However, to take robots out of the factory floor to the real-world, they need to be able to move around in the various environments effectively and effortlessly. To that end, the legged robots design has been shifting towards the implementation of hybrid locomotion systems including IIT's CENTAURO [1], NASA's Robosimian [2], ETH's wheeld Anymal [3], Boston Dynamics' Handle, UBO's Momaro [4], AZIMUT [5], AirHopper [6], HyTRo-I [7], Quattroped [8] and Roller-Walker [9]. The hybrid legged-wheeled design provides a solution for the limitation of the standard legged and wheeled robots benefiting from the fast wheeled motion on the flat grounds while switching to the legged motion, when the robot encounters a more challenging terrain.

With the locomotion flexibility offered by the hybrid mobility and appropriate control tools, these systems have high potential to excel in practical applications adapting effectively to real world during locomanipuation operations. Effective usage of the legged-wheeled robots in the real-world scenarios requires a development of the controllers that will permit to take advantage of the flexibility offered by the hybrid mobility to ensure more effective, safer and robust locomanipuation operations. However, in contrary to their standard well-studied counterparts, kinematics of this newer type of robotic platforms has not been fully understood yet. This gap may lead to an unsatisfactory behaviour when the standard locomotion methods are applied to hybrid legged-wheeled platforms.

In this work, the kinematics and control of the Hybrid Legged-Wheeled (HLW) robots is studied, and framework for the locomotion control of the CENTAURO robot is proposed. It includes the robust omnidirectional driving scheme, and a reactive support polygon adaptation.

To that end, the first and second-order kinematics of the HLW platforms has been studied and analyzed. As a result, new kinematic model for the HLW robot has been developed to better understand their hybrid mobility feature. The proposed model describes the Support Polygon Vertex (SPV) of a general HLW platform as a function of the wheel angular velocities without any assumptions on the robot kinematics or wheel camber angle.

Developed kinematic model forms the basis for the design of an omnidirectional driving scheme. The first-order Inverse Kinematics (IK) control scheme to regulate the robot posture and Support Polygon Vertices (SPVs) is proposed. To ensure the SPV convergence, a higher-level steering adjustment scheme, which accounts for a non-holonomic constraint and its structural singularity, is proposed.

To improve the robot performance in presence of the unpredicted disturbance, the omnidirectional driving scheme has been enhanced with the introduction of a new reactive control scheme. The developed controller takes advantage of the CENTAURO robot six Degrees of Freedom (DoFs) legged-wheeled structure that allows for the continuous support polygon regulation in the entire 2-D space of the ground plane. To this end, a novel quadratic programming task has been designed to push the system SPVs away from the robot Centre of Mass (CoM), while respecting the leg workspace limits.

The omnidirectional driving and reactive control schemes have been verified in the simulation and in the hardware experiments. To implement and test these controllers, a simulator framework for the Series Elastic Actuator (SEA) actuated CENTAURO robot that accounts for the actuators dynamics has been developed with the ROS middleware software and Gazebo simulator. Furthermore, the Simplifying Operations in Locomotion - Framework for Efficient Research (SOL-FER) software has been developed in this work to provide a framework to quickly try out new controllers by maximizing the flexibility and reconfigurability of the developed modules. Finally, for the experimental validation of the proposed controllers, the state estimators for the robot world posture and ground reaction forces have been developed.

This thesis is organized as follows: In Section 2 the literature review for the simulation, modelling and control of the HLW robots is given, in Section 3 theoretical basis for the work is laid out and the CENTAURO robot hardware is introduced. In Section 4 the Gazebo-ROS simulation for the CENTAURO robot is presented, and details on the Simplifying Operations in Locomotion - Framework for Efficient Research (SOL-FER) are given in Section 5. Kinematic model for the HLW robots is proposed in Section 6, Section 7 introduces the non-holonomic omnidirectional driving scheme for the HLW CENTAURO robot, and

Section 8 proposes the reactive support polygon adaptation algorithm. Finally, in Section 9 the conclusion is drawn and in Appendix A state estimation algorithms are described.

Chapter 2

Literature Review

In this chapter, the literature review is given for the robotics simulators in Section 2.1, robotics middlewares in Section 2.2, control of the HLW robots in Section 2.3 and reactive control for the HLW and standard legged quadrupeds in Section 2.4.

2.1 Dynamic Simulation of a Robotic System

Simulators are a necessary mean for the development of dynamical systems and permit the risk-free evaluation and tuning of new control methods before implementation on real systems. As a result, simulation tools can significantly accelerate the system control and software development and reduce the maintenance time and cost of potential damages during the early stage of testing. Hence, it is essential to exploit a comprehensive framework comprising all individual elements of the real platform.

The increasing attention paid to robotics during the past four decades motivated the development of a large number of simulation tools for this class of dynamic systems. Dynamic simulators are mainly classified in two categories: physics engines and simulating environments. The former includes light and efficient libraries solving the system dynamic equations, while the latter comprises computer programs typically possessing a Graphical User Interface (GUI), visualisation, and model editor features, in addition to various robotics toolboxes i.a. control, planning, navigation, vision.

An attempt to compare different physics engines in an objective manner was presented in [10]. In this group, one can refer to Open Dynamic Engine (ODE) described in [11], Open Robotics Automation Virtual Environment (OpenRAVE) reported by [12], Simbody explained in [13], Multi-Joint dynamics with Contact (MuJoCo) introduced in [14], and Bullet presented in [15].

As for the simulating environments for robotics one can mention Urban Search And Rescue simulation (USARSim) described in [16], Robot Control Simulator (ROCOS) explained by [17], Gazebo detailed in [18], Webot described in [19], Verosim examined by [20] and CoppeliaSim presented in [21]. Among this class of simulators the most commonly used is Gazebo [22]; an open-source simulator developed by an Open Source Robotic Foundation. Within its advantages one can mention a choice between multiple physics engines, an intensive development and a strong community as well as a simplicity in creating a robot model with the Semantic Robot Description Format (SRDF) and the Unified Robot Description Format (URDF) files as well as an integrated GUI. A possibility to extend the simulation by plugins is Gazebo another plus.

The software environments dedicated to mobile robots are highly-focused examples of simulating environments. AMORsim [23] is dedicated to three-wheeled robots, Roborobo! [24] proposes a lightweight solution created to work with multiple models simultaneously, Autonomous Robots Go Swarming (ARGoS) [25] allows to assign different physics engines to each part of the simulation. Finally, in [26] a simulator dedicated to mobile robots for astronaut assistance was introduced.

The simulation and software framework used for legged robots are studied in several works. To take humanoids simulators and platforms architectures as examples, we can report [27] developed for iCub robot, [28] created for COMAN, [29] built for DarwinOP, [30] implemented for NimbRo-Open Platform and [31] designed for ARMAR-III. Moreover, a simulator of quadruped BIOSBOT is detailed in [32].

A few papers focuses on a certain aspect of study rather than a particular dynamical structure, thus, [33] proposes general quadruped gait simulator, whereas [34] writes about hydraulically actuated quadrupeds, and [35] uses decoupled tree-structure approach for quadruped simulation.

2.2 Software Arichtecture

Communication of a higher-level architecture with a robot/simulator requires an operating platform capable of incorporating different control architectures while interacting with various devices. To this end, middleware software layers have been developed as an important component of the robotic framework.

Amongst numerous middleware software, one may report Robotic Operating System (ROS) discussed in [36], Yet Another Robot Platform (YARP) studied in [37], Player introduced in [38], Universal Robotic Software Platform (Urbi) expounded in [39], Middleware for Robotic Applications (MIRA) proposed by [40] and Open Robot Control Software (OROCOS) outlined in [41]. [42] provides a survey over exisiting middlewares, and [43] discusses the importance of cross-system compatibility.

According to [22], ROS and YARP are the most commonly employed middleware software solutions in humanoid robotics. ROS-based open-source simulators are currently available for several robots such as the NAO [44] and NimbRO [30] robots. As for the HUBO [45], a ROS interface with a real-time control system is exploited to illustrate the ROS capabilities of providing a communication layer for the robot; meanwhile, the software architecture of the iCub robot is developed in YARP [46]. In addition, the WALK-MAN and COMAN robots are also utilising a YARP-based architecture as one of the operating software [28]. Finally, note that NAO has been integrated with Urbi for a RoboCup simulation competition [47].

2.3 Control of the Hybrid Legged-Wheeled Robots

With the development of HLW platforms, dedicated motion controller schemes have been proposed for PAW robot [48, 49], Roller-Walker [50], KaMERo1 [51] and Shrimp[52]. While these controllers relay on a given robot kinematics, for more general solution, [53] shows that the generic decentralized control is not sufficient to stabilize legged-wheeled systems. A centralized controller composed of a gravity compensation scheme and elasticity model [54] can be implemented to improve the performance of the joint-space trajectory tracking. To translate the higher-level control requirements to the joint-space, an operational-space control [55] is often employed to deal with redundant systems, e.g. in case of standard quadrupeds [56], [57] and legged-wheeled quadrupeds [3]. Furthermore, in [58] the Model Predictive Control (MPC) based on the Zero Moment Point (ZMP) has been proposed for the trajectory optimisation of the HLW robot with fixed wheel.

Most research on the stability of legged robots have been dedicated to bipeds, e.g. [59, 60, 61, 62], while quadrupeds, as statically stable systems, have attracted less attention. Nevertheless, an extension of methods developed for humanoid robots to multi-legged systems is not always straightforward due to the workspace limitations and discontinuities in a support polygon [63, 64]. Finally, a sequential quadratic programming based on the ZMP constraint for balancing of legged-wheeled structures is examined in 2D [65, 66] and a preview controller and a zero-phase low-pass filter methods based on the ZMP constraint are tested in 3D in [67].

Non-holonomic constraint

Platforms with only steerable wheels can perform omnidirectional driving if the wheel orientation is controlled to comply with a set of non-holonomic constraints [68]. Most studies in this subject have been done for standard mobile robots, and the proposed solutions are based on the Instantaneous Centre of Rotation (ICR). In [69] the ICR is computed in spherical coordinates, in [70] the steering reference is extracted without computing the ICR explicitly, and in [71] the path discontinuities are considered. Furthermore, to handle a structural singularity in computation of a steering reference, artificial potential fields are used in [72, 73]. To incorporate the wheel steering into the IK control, the mixed velocity/acceleration schemes have been proposed for the standard mobile platforms with only steerable wheels in [74, 75]; the steering singularity is then handled through the damped IK.

For a legged-wheeled robots, the approaches in [76, 77] propose to separate the ankle yaw and body posture kinematics, and solve the former for a desired steering angle, while the work in [78] employs the ICR. Furthermore, the control scheme in [79] is based on a second order kinematics model described by mixed velocity/acceleration state. The leg kinematics of the platforms considered in these papers allows for a relative wheel-body motion in one direction only, and constrains wheels to zero camber angles. On the contrary, the kinematics analysed in [80, 81, 82] allows for a wheel-base motion along the lateral and longitudinal axes. In these works, operational space control framework [55] with a second order kinematics model described by the mixed velocity/acceleration state is used to develop motion schemes; a zero-camber angle assumption holds. As reported in [80], a first-order kinematics scheme with steering reference based on the ICR alone causes the support polygon of these systems to diverge.

2.4 Reactive Control

So far, literature on the reactive control of the legged-wheeled platforms has been sparse. In [83] the balancing of the legged-wheeled systems has been discussed to avoid a system tip over, and in [84] authors optimise the reaction forces for the obstacle crossing. However, these works consider a disturbance aligned with the wheel orientation, and no wheel steering control has been applied. To the best of the author knowledge, no reactive control scheme has been proposed so far for the general legged-wheeled platforms that accounts for the steerable wheels.

More studies can be found focusing on the disturbance rejection in quadrupeds. These schemes are limited with the constant support polygon, or have to resolve to the stepping/sliding actions. The former relies on the robustness of the lower-level whole-body control [85], and the latter typically consists in online MPC planning schemes [86, 87, 88]. Alternatively, in [89] adaptive fuzzy logic learning has been applied, while the Any-time-Repairing A* algorithm has been implemented in [90], and the N-step capturability has been used in [91]. Although, many of the solutions developed for standard quadrupeds can be directly applied for the CENTAURO robot, they neglect its hybrid-mobility feature.

2.5 Summary

In this chapter, the literature review for the software, modelling and control of the HLW robots has been presented.

Chapter 3

Background

In this chapter, theoretical background that forms a basis for the this thesis contribution is presented. First, the mathematical definition of the non-holonomic systems is recalled in Section 3.1, then the modelling of the robotic structures is discussed in Section 3.2, and control methods in robotics are described in Section 3.3. Finally, a quaternion decomposition is delineated in Section 3.4, the CENTAURO robot hardware is introduced in Section 3.5, and a conclusion is given in Section 3.6.

3.1 Non-holonomic robotic system

The robotic system is holonomic if all the constraints are integrable, i.e., they can be written in a form [92]

$$f(\boldsymbol{q},t) = 0, \tag{3.1}$$

where $\boldsymbol{q} \in \Re^n$ describes the state of the system, $t \in \Re^1$ represents time and $n \in \mathbb{N}_0$ stands for the number of system DoFs. If the genaralized velocity satisfies the condition that cannot be writen as an equivalent condition of the generalised position (i.e., cannot be written in a form (3.1)), a system is called non-holonomic for a given task [92].

3.2 Modelling

In this section, modelling methods for the robotic strucutres are laid out. First, the equations for the velocity and acceleration of the point on the rigid-body are recalled in Section 3.2.1, then the standard approaches to model the robotic wheel are discussed in Section 3.2.2 including the Non-Sliding Pure-Rolling (NSPR) constraint. Models for the standard wheeled mobile robots are delineated next in Section 3.2.3 and Section 3.2.4. Models for hybrid legged-wheeled robots are discussed in Section 3.2.5 and for legged robots in Section 3.2.6. Finally, model for the SEA is presented in Section 3.2.7.

3.2.1 Point on the Rigid-Body

Here, the first and second-order 3D point rigid-body kinematics is recalled, on an example point 'a' described in the wheel frame $\mathscr{F}_w$. The rigid body transformation reads [93]

$$^{o}\boldsymbol{x}_a = {}^{o}\boldsymbol{x}_w + {}^{w}\boldsymbol{x}_a, \tag{3.2}$$

where $^{o}\boldsymbol{x}_w \in \Re^3$ stands for the vector from the inertial frame origin to the wheel center, $^{w}\boldsymbol{x}_a \in \Re^3$ represents the vector from the wheels centre to the considered reference point 'a', and $^{o}\boldsymbol{x}_a \in \Re^3$ denotes the vector from the origin of the intertial frame to the reference point 'a'. The velocity of the point 'a' is described by

$$^{o}\dot{\boldsymbol{x}}_a = {}^{o}\dot{\boldsymbol{x}}_w + {}^{w}\dot{\boldsymbol{x}}_a = {}^{o}\dot{\boldsymbol{x}}_w + \boldsymbol{\omega}_w \times {}^{w}\boldsymbol{x}_a + {}^{w}\dot{\boldsymbol{x}}_a^{'}, \tag{3.3}$$

where $\boldsymbol{\omega}_w \in \Re^3$ symbolises the wheel angular velocity, and $^{w}\dot{\boldsymbol{x}}_a^{'} \in \Re^3$ represents the point velocity with respect to the wheel frame origin. The acceleration of the point 'a' reads [94]

$$^{o}\ddot{\boldsymbol{x}}_a = {}^{o}\ddot{\boldsymbol{x}}_w + {}^{w}\ddot{\boldsymbol{x}}_a = {}^{o}\ddot{\boldsymbol{x}}_w + \dot{\boldsymbol{\omega}}_w \times {}^{w}\boldsymbol{x}_a + \boldsymbol{\omega}_w \times \boldsymbol{\omega}_w \times {}^{w}\boldsymbol{x}_a + 2\boldsymbol{\omega}_w \times {}^{w}\dot{\boldsymbol{x}}_a^{'} + {}^{w}\ddot{\boldsymbol{x}}_a^{'}. \tag{3.4}$$

3.2.2 Wheeled Robots

This section presents standard methods to model the robotic wheel: from the definition of the terms typically used to discus the wheeled robots through overview over the wheel geometries to the Non-Sliding Pure-Rolling (NSPR) contact point assumption.

Wheel geometry

The **wheel axis** is placed at the wheel geometric centre and is oriented in a way that the wheel geometry is invariant towards the rotation around the wheel axis.

The **wheel plane** divides the wheel in the two reflected halves and is orthogonal to the wheel axis. Thus, the wheel plane describes the plane of rotation of the rotation around the wheel axis.

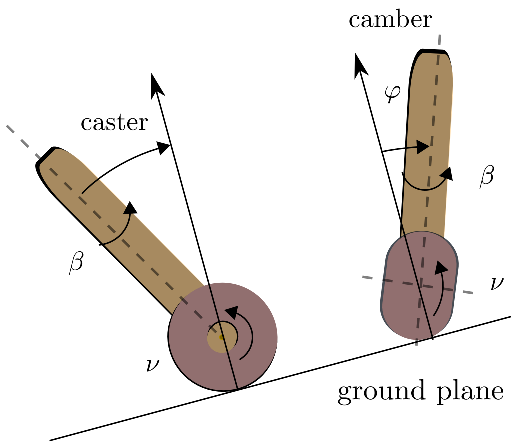


Figure 3.1 Camber and caster angles in the robotic wheel assembly.

To define the wheel orientation with respect to the ground, let us define the following angles. The **steering angle** ($\beta \in \Re^1$) describes the wheel orientation around the ground normal, the **camber angle** ($\varphi \in \Re^1$) measures the angle between the ground normal and the wheel plane (see Fig. 3.1) that is a complementary acute angle to the angle from the ground vector to the wheel axis. The **rolling angle** ($\nu \in \Re^1$) describes the rotation around the axis orthogonal to the rotation axes of the steering and camber angles, and the **caster angle** measures the angle in the wheel plane between the ground normal and the rotation axis of the actuator directly attached to the wheel, see Fig. 3.1.

To model the Standard Wheeled Mobile Robot (SWMR) and HLW robots on the rigid ground, a common simplification is to consider the wheel as a sphere or a cylinder what provides a sufficient approximation for systems with fixed camber angle. Multiple models with more complex geometries have been developed to tackle the flexible tires and compliant environments [95].

The Non-Sliding Pure-Rolling Contact

For the wheeled robots, a Non-Sliding Pure-Rolling (NSPR) assumption is commonly adopted [95]. It is a heuristic assumption that imposes the point of a wheel in the contact with the ground to have no instantaneous velocity (${}^{o}\dot{\boldsymbol{x}}_{cp} = \mathbf{0}$). The NSPR assumption constraints the point rigidly attached to the robot wheel (i.e., ${}^{w}\dot{\boldsymbol{x}}'_{cp} = \mathbf{0}$); from (3.3) the contact point assumption reads

$$
{}^{o}\dot{\boldsymbol{x}}_{cp} = {}^{o}\dot{\boldsymbol{x}}_{w} + {}^{w}\dot{\boldsymbol{x}}_{cp} = \mathbf{0} \implies {}^{o}\dot{\boldsymbol{x}}_{w} = {}^{w}\boldsymbol{x}_{cp} \times \boldsymbol{\omega}_{w}, \tag{3.5}
$$

where ${}^{o}\boldsymbol{x}_{cp}$ denotes the vector from the origin of the intertial frame to the contact point. To compute the contact point acceleration in the rolling motion, note that at each time step a different point of the wheel touches the ground. As a result in (3.5) the only constant point is the wheel center; and thus (3.5) describes the condition for the wheel centre to maintain the rolling motion. At any given time t, (3.5) gives

$$ {}^{o}\dot{\boldsymbol{x}}_{w|t} = {}^{w}\boldsymbol{x}_{cp|t} \times \boldsymbol{\omega}_{w|t}. \tag{3.6} $$

A difference between the wheel centre velocity at two time steps reads

$$ {}^{o}\dot{\boldsymbol{x}}_{w|t+T} - {}^{o}\dot{\boldsymbol{x}}_{w|t} = {}^{w}\boldsymbol{x}_{cp|t+T} \times \boldsymbol{\omega}_{w|t+T} - {}^{w}\boldsymbol{x}_{cp|t} \times \boldsymbol{\omega}_{w|t}, \tag{3.7} $$

where $T \in \Re^{1}$ marks the time step. And thus the wheel centre acceleration has to read

$$ {}^{o}\ddot{\boldsymbol{x}}_{w} = \lim_{T\to 0} \left({}^{o}\dot{\boldsymbol{x}}_{w|t+T} - {}^{o}\dot{\boldsymbol{x}}_{w|t} \right) = \frac{d}{dt} \left({}^{w}\boldsymbol{x}_{SPV} \times \boldsymbol{\omega}_{w} \right), \tag{3.8} $$

where ${}^{w}\boldsymbol{x}_{SPV} \in \Re^{3}$ is a vector that describes a set of consecutive positions of a robot contact in the ground with respect to the wheel centre. This function is referred to as SPV in this work. Note, that, by definition, at each time step ${}^{w}\boldsymbol{x}_{SPV|t} \triangleq {}^{w}\boldsymbol{x}_{cp|t}$. However, their derivatives may differ ${}^{w}\dot{\boldsymbol{x}}_{SPV|t} \neq {}^{w}\dot{\boldsymbol{x}}_{cp|t}$.

With $r \in \Re^{+}$ representing the wheel readius, for the standard wheeled mobile robots in the inertial frame ${}^{w}\boldsymbol{x}_{SPV} = \begin{bmatrix} 0 & 0 & -r \end{bmatrix}^{T}$ what is a constant value. Thus, the wheel acceleration in the rolling motion reads

$$ {}^{o}\ddot{\boldsymbol{x}}_{w} = {}^{w}\boldsymbol{x}_{SPV} \times \dot{\boldsymbol{\omega}}_{w}. \tag{3.9} $$

Since, the contact point constraint is applied to the point rigidly attached to the wheel (i.e., ${}^{w}\dot{\boldsymbol{x}}'_{cp} = \boldsymbol{0}$, ${}^{w}\ddot{\boldsymbol{x}}'_{cp} = \boldsymbol{0}$) (3.4) for a point in the ground contact reads

$$ {}^{o}\ddot{\boldsymbol{x}}_{cp} = {}^{o}\ddot{\boldsymbol{x}}_{w} + \dot{\boldsymbol{\omega}}_{w} \times {}^{w}\boldsymbol{x}_{cp} + \boldsymbol{\omega}_{w} \times \boldsymbol{\omega}_{w} \times {}^{w}\boldsymbol{x}_{cp}, $$

taking into account (3.9) the constraint on the contact point acceleration for the standard wheeled mobile robot to maintain the rolling motion reads

$$ {}^{o}\ddot{\boldsymbol{x}}_{cp} = \boldsymbol{\omega}_{w} \times \boldsymbol{\omega}_{w} \times {}^{w}\boldsymbol{x}_{cp}. $$

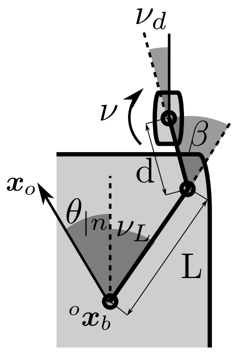


Figure 3.2 General structure of the conventional wheel assembly.

3.2.3 Standard Wheeled Mobile Robot

In this section, modelling of the SWMR is discussed for the standard wheel assemblies, and the mobility of the SWMR for the omnidirectional driving is outlined based on [95].

Mobility of the SWMR is defined by the type of the wheel assembly that includes the fixed, caster and steerable wheels[1]. Fig. 3.2 shows the general wheel assembly, where L and d represent constants lengths, ν_d and ν_L symbolise constant angles, $\boldsymbol{q} = [\beta \quad \nu]$ mark the joint-space variables, and $\theta_{|n}$ stands for the robot base rotation around the ground normal. The position of the robot reference point (${}^o\boldsymbol{x}_b \in \Re^3$) remains to fully describe a robot state. Finally, frame '$\mathscr{F}_b$' is rigidly attached to the robot base.

A different wheel assemblies are generated by defining some parameters in Fig. 3.2 to zero. In particular, in a fixed wheel $\beta = d = 0$, in a steerable wheel $\nu_d = d = 0$, and in a caster wheel $\nu_d = 0$. The contact point velocity for the general wheel assembly reads

$$\begin{aligned}\begin{bmatrix}\dot{x}_{cp}\\ \dot{y}_{cp}\\ \dot{z}_{cp}\end{bmatrix} &= \begin{bmatrix} d\sin(\nu_d)+L\sin(\beta+\nu_d)\\ d\cos(\nu_d)+L\cos(\beta+\nu_d)\\ 0\end{bmatrix}\underline{\omega}_{|n} + \begin{bmatrix} d\sin(\nu_d)\\ d\cos(\nu_d)\\ 0\end{bmatrix}\dot{\beta} + \begin{bmatrix}-r\\ 0\\ 0\end{bmatrix}\dot{\nu} + \\ &\begin{bmatrix}\cos(\nu_L+\beta+\nu_d) & \sin(\nu_L+\beta+\nu_d) & 0\\ -\sin(\nu_L+\beta+\nu_d) & \cos(\nu_L+\beta+\nu_d) & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos(\theta_{|n}) & \sin(\theta_{|n}) & 0\\ -\sin(\theta_{|n}) & \cos(\theta_{|n}) & 0\\ 0 & 0 & 1\end{bmatrix}{}^o\dot{\boldsymbol{x}}_b^{(b)}.\end{aligned} \tag{3.10}$$

[1]Unless directly state otherwise, SWMR section is based on [95] and [68]

With the NSPR assumption (3.5), (3.10) reads that the robot base motion is confined on the ground plane, i.e. only the planar elements of the robot base (${}^{o}\boldsymbol{x}_{b|Pn}$, $\underline{\omega}_{|n}$) are not constraint.

For the caster wheel assembly ($v_d = 0$), (3.10) reads

$$\boldsymbol{A}_C(\beta)\boldsymbol{r}_n(-\theta_{|n})\underline{\dot{x}}^{(b)}_{b|Pn} + \boldsymbol{B}_C(\beta)\underline{\omega}_{|n} = \begin{bmatrix} 0 & r \\ -d & 0 \end{bmatrix} \begin{bmatrix} \dot{\beta} \\ \dot{v} \end{bmatrix}, \tag{3.11}$$

where $\boldsymbol{A}_C$ and $\boldsymbol{B}_C$ are transformation matrices dependent on the wheel steering angle (β). It shows that for the caster wheel assembly, any robot base planar motion can be adopted at any time. Thus the caster wheel assembly does not restrict the robot base motion, and the system with only caster wheels permits holonomic omnidirectional driving.

On the other hand, for the fixed wheel under the rolling assumption. i.e. $\beta = 0$, $\dot{\beta} = 0$, (3.10) reads

$$\boldsymbol{A}_F\boldsymbol{r}_n(-\theta_{|n})\underline{\dot{x}}^{(b)}_{b|Pn} + \boldsymbol{B}_F\underline{\omega}_{|n} = \begin{bmatrix} r \\ 0 \end{bmatrix} \dot{v}, \tag{3.12}$$

where $\boldsymbol{A}_F$ and $\boldsymbol{B}_F$ are constant transformation matrices. (3.12) shows that the NSPR assumption in the fixed wheel constraints one direction of the robot base velocity, and thus the omnidirectional driving satisfying the NSPR assumption is not possible in the vehicle with a fixed wheel.

Finally, for the steerable wheel assembly (i.e $v_d = d = 0$), (3.10) reads

$$\boldsymbol{A}_S(\beta)\boldsymbol{r}_n(-\theta_{|n})\underline{\dot{x}}^{(b)}_{b|Pn} + \boldsymbol{B}_S(\beta)\underline{\omega}_{|n} = \begin{bmatrix} r \\ 0 \end{bmatrix} \dot{v}. \tag{3.13}$$

where $\boldsymbol{A}_S$ and $\boldsymbol{B}_S$ are transformation matrices dependent on the wheel steering angle (β). Similarly, to the fixed wheel assembly, the steerable wheel constraints one direction of the robot base planar velocity. However, if one extends the state $\boldsymbol{q} = [\beta \quad v]$ to consider an integral over the wheel steering angle (β), what reads $\boldsymbol{q} = [\beta \quad v \quad \int \beta(t)\,dt]$, one can control the allowed direction of motion. As a result the omnidirectional driving may be achieved. However, the (3.13) does not satisfy (3.1), and thus the mobile robot with the steerable wheel is restricted to non-holonomic omnidirectional driving.

3.2.4 Non-holonomic constraint in the SWMR.

In this section, the standard methods to satisfy the non-holonomic constraint in the SWMR with only steerable wheels are presented.

(3.13) imposes that all axes of steerable wheels have to cross at one point – the ICR of robot motion – for the robot to execute the omnidirectional driving [95]. The ICR is defined as a point on the rigid body moving in a planar motion that has no velocity at a given time; it reads [71, 72, 78]

$$\boldsymbol{x}_{ICM|Pn} = \boldsymbol{x}_{b|Pn} + \frac{1}{||\underline{\boldsymbol{\omega}}_{|n}||^2}\boldsymbol{\omega}_{w|n} \times \dot{\boldsymbol{x}}_{b|Pn}. \tag{3.14}$$

The desired steering that satifies the non-holonomic constraint in (3.13), and thus orients a wheel axis to cross the ICR, reads [71, 78]

$$\beta = \tan^{-1}\left(\frac{y_{ICM} - y_{cp}}{x_{ICM} - x_{cp}}\right) + 0.5\pi. \tag{3.15}$$

The singularity in (3.15) arises, when desired ICR coincide with the contact point (i.e., $x_{ICM} = x_{cp}$, $y_{ICM} = y_{cp}$).

An alternative approach to control the SWMR with only steerable wheels is throught the second-order kinematics, e.g., [71, 74]. To that end, (3.13) is differentiated to

$$\begin{aligned} &\boldsymbol{A}_S(\beta)\frac{d\boldsymbol{r}_n(-\theta_{|n})}{dt}\dot{\underline{x}}^{(b)}_{b|Pn} + \boldsymbol{A}_S(\beta)\boldsymbol{r}_n(-\theta_{|n})\ddot{\underline{x}}^{(b)}_{b|Pn} + \\ &\boldsymbol{B}_S(\beta)\dot{\underline{\omega}}_{|n} = \begin{bmatrix} r \\ 0 \end{bmatrix}\ddot{v} - \left(\frac{d\boldsymbol{A}_S(\beta)}{d\beta}\boldsymbol{r}_n(-\theta_{|n})\dot{\underline{x}}^{(b)}_{b|Pn} + \frac{d\boldsymbol{B}_S(\beta)}{d\beta}\underline{\omega}_{|n}\right)\dot{\beta}. \end{aligned} \tag{3.16}$$

(3.16) can be solved for state $\begin{bmatrix} \dot{\beta} & \ddot{v} \end{bmatrix}^T$ with any desired robot motion $\begin{bmatrix} \ddot{\underline{x}}^{(b)T}_{b|Pn} & \dot{\underline{\omega}}_{|n} \end{bmatrix}^T$, and standard IK methods for the model with the mixed velocity/acceleration state can be used to control the robot.

3.2.5 Hybrid legged-wheeled robots

Here, the standard approaches to model the hybrid legged-wheeled systems are outlined.

Kinematics of the HLW systems can be divided into two main categories, robots where the relative motion between the robot base and the contact point is constrained in at least one direction, and the systems where the contact point has 3DoFs with respect to the robot base.

While, the former design – e.g., Hylos [77], Momaro [4] – is more common than the later – e.g., CENTAURO [1], Robosimian [2], it also constraints the wheel caster and camber angles to permit the non-holonomic omnidirectional driving.

For a hybrid legged-wheeled system, in contrary to legged robots with point feet (with negligible foot geometry) or standard wheeled platforms (with fixed placement of wheels w.r.t to the base), it is not possible to define a constant transformation between a wheel contact point with the ground and a standard rigid-body reference frame attached to a robot link/joint. Nevertheless, by assuming zero camber angle and known ground normal, the contact point position can be computed based on the system forward kinematics [77]

$$\boldsymbol{x}_{cp} = \boldsymbol{x}_w - \boldsymbol{n}r, \tag{3.17}$$

where r stands for the wheel radius. Note, that (3.17) assumes the spherical wheel geometry.

With (3.17), the desired joint-space command is computed from the rigid-body kinematics, solving the NSPR constraint (see Section 3.2.2) for the desired robot base motion (e.g., [77, 78, 80])

$$\mathbf{J}_{cp|a}\boldsymbol{q}_a = -\mathbf{J}_{cp|b}\boldsymbol{q}_{b,\text{des}}, \tag{3.18}$$

where

$$\mathbf{J}_{cp} = \begin{bmatrix} \mathbf{J}_{cp|a} & \mathbf{J}_{cp|b} \end{bmatrix}. \tag{3.19}$$

However, for the first type of the HLW platforms, i.e., when the motion of the wheel contact point is constrained in one direction, the constraints equation takes a form similar to (3.13), and the non-sliding constraint imposes the system non-holonomy. Typically, the methods described in Section 3.2.4 are then used to implement the omnidirectional driving (e.g., [76, 77, 78]).

For the second type of the HLW robots, where the contact point has 3DoFs with respect to the robot base, the non-zero camber/caster angle is still typically assumed. Then, the contact point Jacobian takes the form similar to (3.13) where wheel steering motion has no direct influence on the contact point motion. However, as reported in [80], a first-order kinematics scheme with steering reference based on the ICR alone causes the support polygon of these systems to diverge. The second-order approach similar to (3.16) is then used to provide a non-holonomic omnidirectional driving (e.g., [79, 80, 81, 82]).

3.2.6 Legged Robots

In this section, the typical approach to model the legged robot is presented, the contact points modelling is discussed, and the ZMP constraint is outlined.

The supporting DoFs in legged robot change depending on which legs are in the ground contact. To incorporate the variable support state and flight phases, a floating base model is typically adopted for the legged robots. The generalised coordinates read [96]

$$\bar{\boldsymbol{q}} = \begin{bmatrix} \bar{\boldsymbol{q}}_b^T & \boldsymbol{q}_a^T \end{bmatrix}^T, \tag{3.20}$$

where $\bar{\boldsymbol{q}}_b \in \{\mathrm{SE}^3 \mid \Re^{n_b}\}$ refers to coordinates of any floating base representation, $n_b \in \mathbb{N}$ denotes the size of the floating base coordinates vector, and $\boldsymbol{q}_a \in \Re^{n_a}$ represents the vector of link-side positions of actuated joints with $n_a \in \mathbb{N}$ symbolising the number of actuated joints. Therefore, the overall system is described by $n_c = n_a + n_b$ coordinates. We can also define a new set of generalised coordinates

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_b^T & \boldsymbol{q}_a^T \end{bmatrix}^T, \tag{3.21}$$

with $\boldsymbol{q}_b \in \{\mathrm{SE}^3 \mid \Re^6\}$ denoting the floating base coordinates, and $\boldsymbol{q} \in \Re^n$ with $n = n_a + 6$ therefore holds. The system's dynamic model can then be presented by [96]

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{F}_g(\boldsymbol{q}) = \mathbf{S}^T \boldsymbol{\tau}_t(\boldsymbol{q}_a, \dot{\boldsymbol{q}}_a, \boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) + \mathbf{J}_c^T(\boldsymbol{q})\boldsymbol{\lambda}, \tag{3.22}$$

where $\mathbf{M} \in \Re^{n \times n}$ stands for the corresponding generalised inertia matrix, $\boldsymbol{c} \in \Re^n$ represents Coriolis/centrifugal forces, $\boldsymbol{F}_g \in \Re^n$ denotes the gravitational torque vector, $\boldsymbol{\tau}_t \in \Re^k$ refers to the vector of transmission torques applied by the passive elements, and $\boldsymbol{\theta} = [\theta_1, ..., \theta_{n_a}]^T$ refers to the motor position vector. $\mathbf{S} = \begin{bmatrix} \mathbf{0}^{n_a \times n_b} & \mathbf{I}^{n_a \times n_a} \end{bmatrix}$ symbolises the actuation selection matrix, $\mathbf{J}_c \in \Re^{k \times n}$ describes the constraints Jacobian, and $\boldsymbol{\lambda} \in \Re^k$ stands for reaction forces with $k \in \mathbb{N}$ expressing the number of system constraints.

Ground Contact In The Legged Robots

The kinematic and dynamic models relay on the contact point assumption. To that end, dynamic models for the tangential and normal forces at the contact point are discussed [96] in this section.[2]

A **Coulomb friction** model is the most commonly adopted approach to model the tangential force at the contact point of the legged robots. It reads that for the contact point to remain static the tangential force has to remain below the threshold proportional to the normal reaction force squared. If static condition is not met, the contact point slides with the tangential force proportional to the normal force

$$\begin{cases} ||\underline{F}_{cp|Pn}|| \leq \mu \underline{F}^2_{cp|\boldsymbol{n}} & \text{if } \dot{\boldsymbol{x}}_{cp} = \boldsymbol{0} \\ \underline{F}_{cp|Pn} = -\mu \underline{F}_{cp|\boldsymbol{n}} \frac{\dot{\boldsymbol{x}}_{cp}}{||\dot{\boldsymbol{x}}_{cp}||} & \text{if } \dot{\boldsymbol{x}}_{cp} \neq \boldsymbol{0} \end{cases}. \tag{3.23}$$

For the robot foot in the ground contact the unilaterality of the reaction forces is typically assumed; that reads

$$\underline{F}_{cp|\boldsymbol{n}} \geq 0. \tag{3.24}$$

Violation of this constraint would correspond to the robot exerting the force while raising the leg above the ground and thus breaking the contact.

Three models are prevalent when discussing the normal force at the contact point: the **rigid-body model**, the **compliant model** and the **impact model**. The rigid-body model does not allow for any ground penetration; it reads

$$\begin{cases} \underline{F}_{cp|\boldsymbol{n}} \geq \boldsymbol{0} & \text{if } \dot{\boldsymbol{x}}_{cp} = \boldsymbol{0} \\ \underline{F}_{cp|\boldsymbol{n}} = \boldsymbol{0} & \text{if } \dot{\boldsymbol{x}}_{cp} \geq \boldsymbol{0} \end{cases}. \tag{3.25}$$

On the other hand, the compliant model captures the ground elasticity by modelling the ground-contact as a spring

$$\begin{cases} \underline{F}_{cp|\boldsymbol{n}} \geq \boldsymbol{0} & \text{if } \dot{\boldsymbol{x}}_{cp} = \boldsymbol{0} \\ \underline{F}_{cp|\boldsymbol{n}} = -K_g \boldsymbol{x}_{cp} - D_g \dot{\boldsymbol{x}}_{cp} & \text{if } \dot{\boldsymbol{x}}_{cp} \leq \boldsymbol{0} \end{cases}, \tag{3.26}$$

where $K_g \in \Re^1$ sybmolises the stiffnes of the robot-ground interaction, and $D_g \in \Re^1$ denotes the damping of the robot-ground interaction. Finally, the impact model is used in conjunction with the rigid-body model to compute the robot state at an instance after the contact point

[2]Unless directly stated otherwise, Ground Contact In The Legged Robots section is based on [96].

touched the ground at time t. At this moment discontinuity in the rigid-body model appears. The impact model assumes the contact point is static right after the impact; it reads

$$\dot{\boldsymbol{x}}_{cp|t+T} = \mathbf{J}_{cp}\dot{\boldsymbol{q}}_{t+T} = \mathbf{0}, \tag{3.27}$$

where a subscript $t+T$ symbolises variables right after the impact.

Stability Measures

The literature presents several measures to evaluate the system stability that includes the most widely recognised CoM position, ZMP/Centre of Pressure (CoP) [97].

On the flat ground the robot is balanced if the following condition holds [96][3]

$$\begin{aligned} &\underline{x}_{CoM|Pn} - \frac{\underline{x}_{CoM|n}}{\ddot{\underline{x}}_{CoM|n} + \underline{g}_{g|n}}\left(\ddot{\underline{x}}_{CoM|Pn} + \underline{g}_{g|Pn}\right) + \\ &+ \frac{1}{m(\ddot{\underline{x}}_{CoM|n} + \underline{g}_{g|Pn})}\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}\dot{\underline{L}}_{|Pn} = \underline{x}_{CoP} \in \operatorname{conv}\{\boldsymbol{x}_{cp}\} \end{aligned}, \tag{3.28}$$

where $\boldsymbol{L} \in \Re^3$ stands for the angular momentum, $\boldsymbol{g}_g \in \Re^3$ symbolises the gravity acceleration vector, and $\boldsymbol{x}_{CoP} \in \Re^3$ denotes the CoP that reads

$$\underline{x}_{CoP} = \frac{\sum\left(\boldsymbol{F}_{cp|n}\boldsymbol{x}_{cp}\right)}{\sum \boldsymbol{F}_{cp|n}}. \tag{3.29}$$

(3.28) reads the robot remains balanced when its CoP projects on the ground within the convex polygon confined by the feet placement also called a **support polygon**.

In static conditions, i.e. when $\dot{\boldsymbol{L}} = \mathbf{0}$, $\ddot{\boldsymbol{x}}_{CoM} = \mathbf{0}$, (3.28) reads that for the robot to remain balanced its CoM has to project, towards the gravity direction, within the robot support polygon; i.e.,

$$\underline{x}_{CoM|Pn} - \frac{\underline{x}_{CoM|n}}{\underline{g}_{g|n}}\underline{g}_{g|Pn} = \underline{x}_{CoP} \in \operatorname{conv}\{\boldsymbol{x}_{cp}\}. \tag{3.30}$$

For control, a linear model for (3.30) is often used. It assumes the robot is moving on the flat ground ($\underline{g}_{g|P_n} = \mathbf{0}$), the robot CoM does no accelerate vertically and the robot angular

[3]The remainder of the Stability Measures section is based on [96]

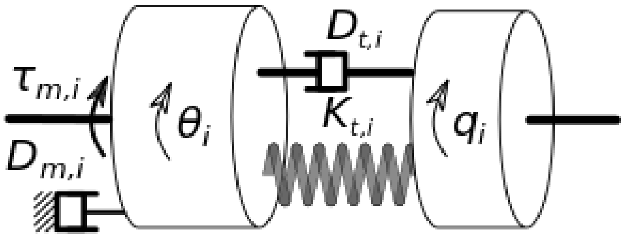


Figure 3.3 Mechanical model of $i-$th series viscoelastic actuator.

momentum does not change. With these assumptions, the balancing condition reads

$$\underline{x}_{CoM|Pn} - \frac{\underline{x}_{CoM|n}}{\underline{g}_{g|n}} \ddot{\underline{x}}_{CoM|Pn} = \underline{x}_{CoP} \in \mathrm{conv}\{\boldsymbol{x}_{cp}\}.$$

3.2.7 Series Elastic Actuators

Here, the model for the Series Elastic Actuators (SEAs) is presented based on [98].

To improve the robot performance, when it is interacting with the environment, recent robots are often powered by the SEA. Motors do not directly drive the links when the system is powered by SEAs, as the motor torques are transmitted to links through compliant elements, see Fig. 3.3. When taking the passive compliance dynamics into account, a n_a-DoF robotic linkage includes $2n_a$ DoFs. At the joint-space level the dynamics of the unconstrained system are then

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{F}_g(\boldsymbol{q}) = \boldsymbol{\tau}_t(\boldsymbol{\phi},\dot{\boldsymbol{\phi}}), \tag{3.31}$$

$$\mathbf{B}_m\ddot{\boldsymbol{\theta}} + \mathbf{D}_m\dot{\boldsymbol{\theta}} + \boldsymbol{\tau}_t(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) = \boldsymbol{\tau}_m, \tag{3.32}$$

$$\boldsymbol{\tau}_t(\boldsymbol{\phi},\dot{\boldsymbol{\phi}}) = \mathbf{K}_t\boldsymbol{\phi} + \mathbf{D}_t\dot{\boldsymbol{\phi}}, \tag{3.33}$$

where $\boldsymbol{q} = [q_1, ..., q_{n_a}]^T$ and $\boldsymbol{\theta} = [\theta_1, ..., \theta_{n_a}]^T$ are the link and motor position vectors, respectively, $\mathbf{K}_t \in \Re^{n_a \times n_a}$ and $\mathbf{D}_t \in \Re^{n_a \times n_a}$ stand for the stiffness and damping matrices corresponding to passive elements, $\boldsymbol{\phi} = \boldsymbol{\theta} - \boldsymbol{q}$ is the transmission displacement vector, whereas $\boldsymbol{\tau}_m \in \Re^{n_a}$ denotes the motor torque vector, $\mathbf{B}_m \in \Re^{n_a \times n_a}$ symbolises the motor inertia matrix and $\mathbf{D}_m \in \Re^{n_a \times n_a}$ expresses the motor damping matrix.

3.3 Motion Control

In this section, methods to control the robotic structure are presented. First, the gravity compensation methods for constrained, compliant robots are discussed in Section 3.3.1, then the impedance control is outlined in Section 3.3.2, and the Inverse Kinematics (IK) methods are discussed in details in Section 3.3.3 and Section 3.3.4.

3.3.1 Gravity Compensation for Compliant Joint Legged Robots

In this section, the gravity compensation approach for the constrained, compliant robots is presented.

In implementation of the gravity compensation scheme for an overconstrained robot like quadruped or humanoid upper-body in contact with environment, the external forces in the dynamic equation (3.22) have to be considered. Over the years, several solutions based on an orthogonal decomposition of the constraints Jacobian have been proposed to address this issue including [99], [100], and [101]. A QR decomposition of the constraints Jacobian [102] is one of the commonly used approaches. It can be expressed by

$$\mathbf{J}_c^T(\boldsymbol{q}) = \mathbf{Q}(\boldsymbol{q})\mathbf{R}(\boldsymbol{q}) \tag{3.34}$$

where $\mathbf{Q} \in \Re^{n\times n}$ is an orthogonal matrix, and $\mathbf{R} \in \Re^{n\times k}$ is an upper triangular matrix with $\mathrm{rank}(\mathbf{R}) = l$; l denotes number of independent contact constraints. Then, by applying (3.34) on (3.22), one can obtain an equivalent dynamic model as

$$\mathbf{S}_c\mathbf{Q}^T(\boldsymbol{q})\left(\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{F}_g(\boldsymbol{q})\right) = \mathbf{S}_c\mathbf{Q}^T(\boldsymbol{q})\mathbf{S}^T\boldsymbol{\tau_t} + \mathbf{R}\boldsymbol{\lambda}, \tag{3.35}$$

$$\mathbf{S}_u\mathbf{Q}^T(\boldsymbol{q})\left(\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{F}_g(\boldsymbol{q})\right) = \mathbf{S}_u\mathbf{Q}^T(\boldsymbol{q})\mathbf{S}^T\boldsymbol{\tau_t}, \tag{3.36}$$

where $\mathbf{S}_c = \begin{bmatrix}\mathbf{I}^{l\times l} & \mathbf{0}^{l\times(n-l)}\end{bmatrix}$ expresses a selection matrix for the constrained part of the system dynamics, and $\mathbf{S}_u = \begin{bmatrix}\mathbf{0}^{(n-l)\times l} & \mathbf{I}^{(n-l)\times(n-l)}\end{bmatrix}$ stands for a selection matrix of the unconstrained part of the system dynamics.

The rigid-joint gravity compensation torque ($\boldsymbol{\tau_g} \in \Re^{n_a}$) can then be defined as [102]

$$\boldsymbol{\tau_g}(\boldsymbol{q}) = (\mathbf{S}_u\mathbf{Q}^T(\boldsymbol{q})\mathbf{S}^T)^{+}\mathbf{S}_u\mathbf{Q}^T(\boldsymbol{q})\boldsymbol{F}_g(\boldsymbol{q}), \tag{3.37}$$

which is constructed based only on the contact points positions, in addition to typical model parameters, and the external force measurement/estimation is not needed.

Compliance Gravity Compensation
From (3.36), (3.37), (3.32), and (3.33) in static condition, i.e. $\dot{\boldsymbol{q}}_a = \ddot{\boldsymbol{q}}_a = \dot{\boldsymbol{\theta}} = \ddot{\boldsymbol{\theta}} = \mathbf{0}$, one can extract the desired motor positions vector as follows [54]

$$\boldsymbol{\theta}_d = \boldsymbol{q}_d + \mathbf{K}_t^{-1} \boldsymbol{\tau}_{\boldsymbol{g}}(\boldsymbol{q}). \tag{3.38}$$

3.3.2 Impedance Control

For a lower-level joint space control, an impedance control with the feed-forward gravity compensation is often used to control the robotic structures. A collocated-based PD position controller [54] is an example of such control scheme, whose corresponding control law is described by

$$\boldsymbol{\tau}_{controller} = \boldsymbol{\tau}_{\boldsymbol{g}}(\boldsymbol{q}_d) + \mathbf{K}_P(\boldsymbol{\theta}_d - \boldsymbol{\theta}) - \mathbf{K}_D \dot{\boldsymbol{\theta}}, \tag{3.39}$$

where $\mathbf{K}_P \in \Re^{k \times k}$ and $\mathbf{K}_D \in \Re^{k \times k}$ are the proportional and derivative gain matrices of the controller, respectively.

3.3.3 Inverse Kinematics Control

In this section, methods for the Inverse Kinematics (IK) control are presented starting from appraoches used for simple robotic strucutres, to the prioritized IK control.

The control requirements, like the robot world posture or the contact point placement are often described in the cartesian-space. To convert the cartesian space tasks into the joint space commands the IK control is among the most commonly adopted methods. The forward kinematics for a rigid-body mechanical system can be computed directly in the form [103][4]

$$\dot{\boldsymbol{x}} = \mathbf{J}\dot{\boldsymbol{q}}, \tag{3.40}$$

where $\dot{\boldsymbol{x}} \in \Re^{|\mathscr{T}_k|}$ describes the desired cartesian-space velocity, $\mathbf{J} \in \Re^{|\mathscr{T}_k| \times n}$ stands for the Jacobian matrix mapping the cartesian-space task to the joint-space solution, and $|\mathscr{T}_k| \in \mathbb{N}$ notes the size of task $\mathscr{T}_k$.

[4]The Inverse Kinematics Control section is fully based on [103].

Direct methods

For the systems with simple kinematics design (3.40) can be solved directly for $\dot{\boldsymbol{q}}$, and the algebraic equations are used to compute the robot commands. However, for more complex designs, computation of the algebraic solution becomes unfeasible and numerical methods have to be adopted. If the size of the task $\mathscr{T}_k$ equals the size of the system state, the IK can be computed through

$$\dot{\boldsymbol{q}} = \mathbf{J}^{-1}\dot{\boldsymbol{x}}, \tag{3.41}$$

where operator $(.)^{-1}$ represents an inverse of a square matrix $(.)$. Solution (3.41) is equivalent to solving the optimization task

$$\underset{\dot{\boldsymbol{q}}}{\text{minimise}} \quad 0.5||\dot{\boldsymbol{x}} - \mathbf{J}\dot{\boldsymbol{q}}||^2 \tag{3.42}$$

On the other hand if the system is redundant for a given task ($\mathscr{T}_k$) (i.e. $|\mathscr{T}_k| < n$), the Moore-Penrose inverse ($(.)^+$) can be used instead; it reads

$$\dot{\boldsymbol{q}} = \mathbf{J}^{+}\dot{\boldsymbol{x}}, \tag{3.43}$$

where

$$\mathbf{J}^{+} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}. \tag{3.44}$$

Solution (3.43) is equivalent to the following optimization task

$$\begin{aligned} &\underset{\dot{\boldsymbol{q}}\in S}{\text{minimise}} && 0.5\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} \\ &\text{with} && S = \dot{\boldsymbol{q}} : \underset{\dot{\boldsymbol{q}}\in\Re^n}{\text{minimise}}\ 0.5||\dot{\boldsymbol{x}} - \mathbf{J}\dot{\boldsymbol{q}}||^2 \end{aligned} \tag{3.45}$$

A singularity in (3.43) arises when the matrix $\mathbf{J}$ is not of a full rank. It corresponds to the situation when the robot actuators are positioned in a manner that they are not capable to affect the motion of at least one element of the operational-space task $\mathscr{T}_k$. To prevent the control scheme (3.43) from generating an infinitely large joint-space velocity commands in the proximity of the singularity, damped pseudo-inverse is used in (3.43); it reads

$$\mathbf{J}^{+} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \boldsymbol{\delta})^{-1},$$

where $\boldsymbol{\delta} \in \Re^{n\times n}$ is a diagonal damping matrix. It is equivalent to solving the following optimization task

$$\begin{aligned} &\underset{\dot{\boldsymbol{q}}\in S}{\text{minimise}} \quad 0.5\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} \\ &\text{with} \qquad S = \dot{\boldsymbol{q}} : \underset{\dot{\boldsymbol{q}}\in\Re^n}{\text{minimise}}\, 0.5||\dot{\boldsymbol{x}} - \mathbf{J}\dot{\boldsymbol{q}}||^2 + 0.5\dot{\boldsymbol{q}}^T\boldsymbol{\delta}\dot{\boldsymbol{q}} \end{aligned}$$

Task Priority

In the complex robotic systems, multiple operational-space tasks may be imposed on the robot simultaneously. Concatenating all of the tasks into one allows to resolve the systems using (3.41) or (3.43). However, in practice some tasks are more important than others. For example, the legged system has to remain balanced in order to be able to perform a manipulation/walking tasks. For that purpose, prioritized IK methods have been developed.

For the set of ith task $\{\mathscr{T}_0, \ldots, \mathscr{T}_i\}$ a weighted pseudo-inverse Jacobian can be used with (3.43) to establish the tasks priority; it reads

$$\mathbf{J}^+ = \boldsymbol{W}\mathbf{J}^T(\mathbf{J}\boldsymbol{W}\mathbf{J}^T + \boldsymbol{\delta})^{-1}, \tag{3.46}$$

where $\boldsymbol{W} \in \Re^{n\times n}$ stands for the symmetric matrix of the task weights, $\mathbf{J} \in \Re^{k\times n}$ symbolizes the matrix of concatenated tasks Jacobians. It corresponds to solving following optimization problem

$$\begin{aligned} &\underset{\dot{\boldsymbol{q}}\in S}{\text{minimise}} \quad 0.5\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} \\ &\text{with} \qquad S = \{\dot{\boldsymbol{q}} : \underset{\dot{\boldsymbol{q}}\in\Re^n}{\text{minimise}}\, 0.5\sum_{j=0}^{i}\sum_{k=j}^{i}(\dot{\boldsymbol{x}}_j - \mathbf{J}_j\dot{\boldsymbol{q}})\boldsymbol{w}_{jk}(\dot{\boldsymbol{x}}_k - \mathbf{J}_k\dot{\boldsymbol{q}}) + 0.5\dot{\boldsymbol{q}}^T\boldsymbol{\delta}\dot{\boldsymbol{q}}\}, \end{aligned}$$

where $\boldsymbol{w}_{jk}$ stands for the part of the matrix $\boldsymbol{W}$ corresponding to the tasks j and k. In this solution all tasks are resolved at once, and thus the weighted IK provides a soft task hierarchy, where the elements of the matrix $\boldsymbol{W}$ describe the tasks importance.

When the strict tasks hierarchy has to be imposed, the null-space projection method [104] can be used to resolve the IK problem. In this method, the vector of joint space trajectories respecting m tasks ($\dot{\boldsymbol{q}}_m$) is computed with a recursive formula [104]

$$\forall i \in \{1,\ldots,m\} : \dot{\boldsymbol{q}}_i = \dot{\boldsymbol{q}}_{i-1} + (\mathbf{J}_i\boldsymbol{P}_{i-1})^+(\dot{\boldsymbol{x}}_i - \mathbf{J}_i\dot{\boldsymbol{q}}_{i-1}), \tag{3.47}$$

where $\mathbf{J}_i \in \Re^{|\mathscr{T}_i| \times n}$ symbolises the ith task Jacobian, $\boldsymbol{P}_{i-1} = \boldsymbol{I} - \mathbf{J}^{+}_{1,\ldots,i-1}\mathbf{J}_{1,\ldots,i-1}$ stands for the projection matrix to the null-space of all the higher-priority tasks, and $\mathbf{J}^{T}_{1,\ldots,i-1} = [\mathbf{J}_1,\ \ldots,\ \mathbf{J}_{i-1}]$. This task is equivalent to solving a recursive set of optimization problems that read

$$\begin{aligned} &\forall i \in \{1,\ldots,m\}: \\ &\qquad \underset{\dot{\boldsymbol{q}} \in S_i}{\text{minimise}} \quad 0.5||\dot{\boldsymbol{x}}_i - \mathbf{J}_i\dot{\boldsymbol{q}}||^2 \\ &\qquad \text{with} \qquad S_i = \{\dot{\boldsymbol{q}} : \underset{S_{i-1}}{\text{minimise}}\ 0.5||\dot{\boldsymbol{x}}_{i-1} - \mathbf{J}_{i-1}\dot{\boldsymbol{q}}||^2\}, \end{aligned}$$

with initial condition $S_0 = \Re^n$, and the final task

$$\begin{aligned} &\underset{\dot{\boldsymbol{q}} \in S_{m+1}}{\text{minimise}} \quad 0.5\dot{\boldsymbol{q}}^T\dot{\boldsymbol{q}} \\ &\text{with} \qquad S_{m+1} = \{\dot{\boldsymbol{q}} : \underset{S_m}{\text{minimise}}\ 0.5||\dot{\boldsymbol{x}}_m - \mathbf{J}_m\dot{\boldsymbol{q}}||^2\}. \end{aligned}$$

Finally, to consider the task constraints the quadratic programming task can be solved directly

$$\begin{aligned} &\underset{\dot{\boldsymbol{q}} \in S_{m+1}}{\text{minimise}} \quad 0.5||\dot{\boldsymbol{q}}||^2 + 0.5||\boldsymbol{w}||^2 \\ &\text{where} \qquad S_{m+1} = \{\dot{\boldsymbol{q}} : \underset{S_m}{\text{minimise}}\ 0.5||\dot{\boldsymbol{x}}_m - \mathbf{J}_m\dot{\boldsymbol{q}}||\}, \\ &\text{subject to} \quad \boldsymbol{A}\dot{\boldsymbol{q}} + \boldsymbol{a} = 0, \\ &\qquad\qquad\quad \boldsymbol{B}\dot{\boldsymbol{q}} - \boldsymbol{b} \leq 0, \\ &\qquad\qquad\quad \boldsymbol{C}\dot{\boldsymbol{q}} - \boldsymbol{c} \leq \boldsymbol{w}, \end{aligned}$$

where $\boldsymbol{A}$, $\boldsymbol{a}$ symbolise the equality constraints, $\boldsymbol{B}$, $\boldsymbol{b}$ note hard inequality constraints, $\boldsymbol{C}$, $\boldsymbol{c}$ represent the soft inequality constraints and $\boldsymbol{w}$ stands for the slack variables.

To provide a position tracking through the IK schemes the task desired velocity $\dot{\boldsymbol{x}}$ is set-up based on the tracking error

$$\dot{\boldsymbol{x}} = \boldsymbol{K}\boldsymbol{e}, \tag{3.48}$$

where $\boldsymbol{K} \in \Re^{|\mathscr{T}|}$ stands for the tunning gain, and $\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{x}$ symbolise the tracking error.

If a general quaternion is defined as $\boldsymbol{\rho} = [w,\ \boldsymbol{v}^T]^T$ where $w \in \Re^1$ denotes the quaternion scalar part, and $\boldsymbol{v} \in \Re^3$ stands for the quaternion vector part, the orientation tracking error ($\boldsymbol{e}_q \in \Re^3$) can be defined as [105]

$$\boldsymbol{e}_q = w_{\text{des}}\boldsymbol{v} - w\boldsymbol{v}_{\text{des}} + \boldsymbol{v}_{\text{des}} \times \boldsymbol{v}, \tag{3.49}$$

where $\boldsymbol{\rho}$, $\boldsymbol{\rho}_{\text{des}}$ symbolise the current/desired orientation, respectively.

3.3.4 Second-order Inverse Kinematics

In this section, extension of the IK control to the second-order kinematics is outlined.

A simple solution to the second-order inverse kinematics problem reads [103][5]

$$\ddot{\boldsymbol{q}} = \mathbf{J}^{+}(\ddot{\boldsymbol{x}} - \dot{\mathbf{J}}\dot{\boldsymbol{q}}),$$

that is equivalent to solving the optimization problem

$$\underset{\ddot{\boldsymbol{q}} \in \Re^n}{\text{minimise}} \quad 0.5(\mathbf{J}^{+}(\ddot{\boldsymbol{x}} - \dot{\mathbf{J}}\dot{\boldsymbol{q}}))^T(\mathbf{J}^{+}(\ddot{\boldsymbol{x}} - \dot{\mathbf{J}}\dot{\boldsymbol{q}})).$$

Similarly, to the first-order kinematics the damped and weighted Jacobian inverse as well as the task prioritization can be used with the problem description being analogous to the first order kinematics.

3.4 Twist-Swing Decomposition

In this section, quaternion decomposition into two orthogonal rotations is discussed, and the swing-twist decomposition method is presented.

Any quaternion ($\boldsymbol{\rho} \in SO^3$) can be decomposed into the twist quaternion ($\boldsymbol{\rho}_{|a}$) describing the rotation around a given axis ($\boldsymbol{a} \in \Re^3$) and the complementary swing quaternion ($\boldsymbol{\rho}_{|Pa}$) comprising of the remaining rotation [106][6]. For each quaternion two – twist-swing and swing-twist – decompositions can be defined. In the former case the twist rotation is applied first, in the latter case – second. For a quaternion ($\boldsymbol{\rho} = [\rho_w, \boldsymbol{\rho}_a^T]^T$, where $\rho_w \in \Re^1$ symbolises the quaternion scalar and $\boldsymbol{\rho}_a \in \Re^3$ refers to the quaternion vector, the twist-swing decomposition reads

$$\begin{aligned} \boldsymbol{\rho}_{a|a} &= \boldsymbol{a}\frac{\boldsymbol{a}^T\boldsymbol{\rho}_a}{||\boldsymbol{a}||\sqrt{\rho_w^2||\boldsymbol{a}||^2 + \boldsymbol{a}^T\boldsymbol{\rho}_a\boldsymbol{a}^T\boldsymbol{\rho}_a}}; \\ \rho_{w|a} &= \frac{||\boldsymbol{a}||^2\rho_w}{||\boldsymbol{a}||\sqrt{\rho_w^2||\boldsymbol{a}||^2 + \boldsymbol{a}^T\boldsymbol{\rho}_a\boldsymbol{a}^T\boldsymbol{\rho}_a}}; \quad . \\ \boldsymbol{\rho}_{|Pa} &= \boldsymbol{\rho}\boldsymbol{\rho}_a^{-1}\frac{1}{||\boldsymbol{\rho}_a||} \end{aligned} \tag{3.50}$$

[5]Second-order Inverse Kinematics section is based on [103].

[6]Twist-Swing Decomposition section is fully based on [106]

The swing-twist decomposition in terms of the twist-swing decomposition reads

$$\begin{aligned} \boldsymbol{\rho}_{|a} &= \boldsymbol{\rho}^{-1}\boldsymbol{\rho}_{|a}^{-1}\frac{1}{||\boldsymbol{\rho}||||\boldsymbol{\rho}_{|a}||}; \\ \boldsymbol{\rho}_{|Pa} &= \boldsymbol{\rho}_{|Pa}^{-1}\frac{1}{||\boldsymbol{\rho}_{|Pa}||}. \end{aligned} \tag{3.51}$$

3.5 CENTAURO robot

In this section the CENTAURO robot hardware is presented, based on [1]. First, the robot kinematic structure is described, then the robot actuation and sensors are briefly laid out, the robot computational units are given, and the robot mobility is discussed at the end.

The platform discussed in this work is a wheeled centaur-like robot composed of a humanoid upper-body mounted on a quadrupedal lower-body, as shown in Fig. 3.4. Additionally, a 3-DoF head structure composed of a yaw-pitch chain and a continuous yaw DoF has been mounted on the upper-body. The upper-body comprises of a dual-arm robotic system with seven DoFs per manipulator relying mostly upon an anthropomorphic design. The shoulder complex is constructed on the basis of a 3-DoF pitch-roll-yaw arrangement implementing extension-flexion, adduction-abduction and humeral roll motions, respectively. It is connected to the forearm through the upper arm and an elbow joint performing a flexion-extension DoF. The forearm apparatus is formed upon a 3-DoF yaw-pitch-yaw configuration replicating the wrist motions required for orienting the end-effector. The length and mass considered for each of the arms are 72.7 cm and around 10 Kg; the robot width along the shoulders as well as the pelvis width are 61 cm.

The pelvis base encompassing the first actuator of the legs is connected to the arms through the torso section composed of a yaw joint. The lower-body consists of four 6-DoF legs with kinematics arrangement selected according to a spider-like configuration, see Fig. 3.4. At the end of a 3-DoF yaw-pitch-pitch leg structure, a pitch-yaw ankle is mounted with a wheel at the end-effector. The robot height varies from 112 to 171 cm depending on the leg configuration.

To exploit the high fidelity torque control characteristics and physical robustness required for interaction with environment, the robot is actuated by SEAs as described in Section 3.2.7. Depending on the expected joint torque requirement five actuators models of an increasing pique torque has been used. The 19/20-bit magnetic encoders measure the absolute link-side joint positions, and the joint torques measurements are provided by the Strain-Gauge or the Deflection-Encoder sensors [107]. Two lower-level controllers have been implemented: the

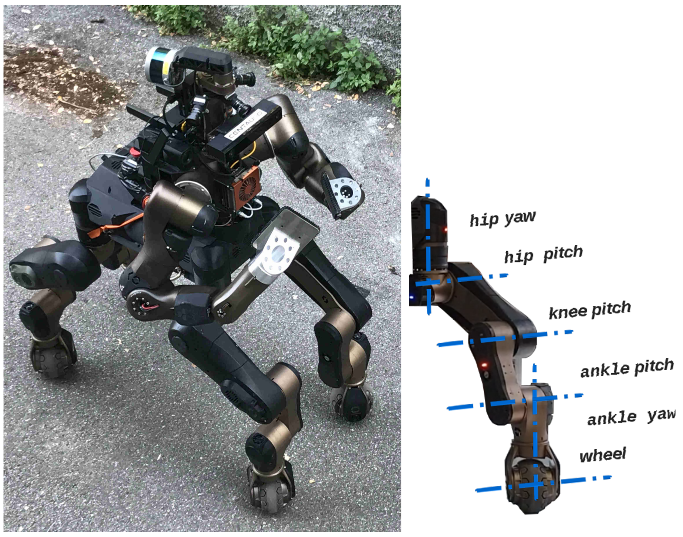


Figure 3.4 On the left: The CENTAURO robot hybrid mobility manipulation platform. On the right: leg kinematic structure of the CENTAURO robot.

position control and impedance/torque control for all joints with exception of the velocity controlled wheels and the velocity controlled continuous head DoF.

The CENTAURO robot battery is placed in the robot pelvis. The robot is equipped with two ZOTAC-EN1070K PCs with high-performance GPUs. One is responsible for the system high-level control and motion planning while the other processes the perception data. Finally, a COM Express conga-TS170 embedded computer with Intel Core i7-6820EQ CPU runs with XENOMAI RT and provides the real-time control layer, and Netgear Nighthawk X10 R900 router ensures the wireless communication with the pilot.

3.5.1 Kinematics Analysis of CENTAURO Robot

The CENTAURO robot lower-body consists of $n_l = 25$ links and $n_a = 24$ actuated DoFs (with $p = 4$ legs of $n_{leg} = 6$ DoFs each). The contact point assumption (Section 3.2.2) constrains the position of the four legs end-effectors, and thus each leg in contact with the ground can be modelled to have an additional 3-DoF spherical joint attached at the end-effector. The overall system kinematics sums up to $n_a + 3f$ DoFs, and the system mobility reads

$$6n_l - 5n_a - 3f = \begin{cases} 21 & f = 3 \\ 18 & f = 4 \end{cases},$$

where $f \in \mathbb{N}$ represents the number of legs in ground contact. Assuming a full floating base state controlled, $15/12$ redundant DoFs remains for $f = 3/4$ legs in ground contact.

During stable motion a leg, in contact with the ground, moves with NSPR assumption (Section 3.2.2), so that the end-effector point in ground contact is constrained to remain still with respect to the ground. The set of constraints can be expressed in the standard first-order kinematics form

$$^{o}\dot{\boldsymbol{x}}_{cp} = \mathbf{J}_{cs}\dot{\boldsymbol{q}} = 0, \tag{3.52}$$

with $\mathbf{J}_{cs} \in \Re^{3f \times n}$ representing constraints Jacobian. Rows associated with i_{th} leg read as

$$\begin{aligned} &\forall i \in \mathbb{F}: \\ &\mathbf{J}_{cs_i} = \begin{bmatrix} \mathbf{J}_{cs_i,\boldsymbol{q}_b} & \mathbf{J}_{cs_i,\boldsymbol{q}_1} & \mathbf{J}_{cs_i,\boldsymbol{q}_2} & \cdots & \mathbf{J}_{cs_i,\boldsymbol{q}_p} \end{bmatrix} \end{aligned} \tag{3.53}$$

where $\mathbf{J}_{cs_i,\boldsymbol{q}_b} \in \Re^{3 \times n_b}$ denotes the part of Jacobian associated with floating base coordinates, and $\mathbf{J}_{cs_i,\boldsymbol{q}_k} \in \Re^{3 \times n_{leg}}$ stands for the part of Jacobian related to the k$^{\mathrm{th}}$ leg DoFs. Then

$$\forall k \in \{1, \ldots, p\} - \{i\} : \mathbf{J}_{cs_i,\boldsymbol{q}_k} \triangleq \mathbf{0}. \tag{3.54}$$

With (3.52), (3.53) and (3.54) one can show that

$$\mathbf{J}_{cs_i,\boldsymbol{q}_b}\dot{\boldsymbol{q}}_b = -\mathbf{J}_{cs_i,\boldsymbol{q}_i}\dot{\boldsymbol{q}}_i,$$

i.e., a ground contact defines three DoFs per leg if the floating base state is imposed. Thus, the CENTAURO robot kinematics allows to control a floating base, 6 DoFs of a free leg and 3 DoFs of a leg in contact with the ground.

3.6 Conclusion

In this chapter, theoretical background that forms a basis for this thesis contribution has been presented, and the hardware used in this work has been outlined. First, the mathematical definition of the non-holonomic systems has been recalled, then the modelling of the robotic structures has been discussed starting from the basic rigid-body point kinematics on the example of the contact point, through modelling of the standard wheeled mobile robots, hybrid legged-wheeled robots and legged robots, to modelling of the Series Elastic Actuators. Finally, control methods in robotics has been described including the gravity compensation, impedance control, and the whole-body inverse kinematics. At the end the CENTAURO robot hardware has been outlined, and the robot mobility has been discussed.

Chapter 4

Simulation of the CENTAURO robot

Simulators, as a necessary mean for the development of dynamical systems, permit the risk-free evaluation and tuning of new control methods before their implementation on the real systems.[1] As a result, simulation tools can significantly accelerate the system control and software development and reduce the maintenance time and cost of potential damages during the early stage of testing. Hence, it is essential to exploit a comprehensive framework comprising all individual elements of the real platform. The increasing attention paid to robotics during the past four decades motivated the development of a large number of simulation tools for this class of dynamic systems. Furthermore, the communication of a higher-level architecture with a robot and/or simulator requires an operating platform capable of incorporating different control architectures while interacting with various devices. To this end, middleware software layers have been developed and exploited as an important component of the simulation, interfacing and control of robotics systems.

This chapter presents a simulator framework for a SEA (Section 3.2.7) actuated CENTAURO robot (see Section 3.5) where the ROS middleware software and Gazebo simulator are exploited. While the passive dynamics imposed by the inclusion of SEA has significant effect on the system behaviour, it is usually neglected since the dynamics associated with motors is not embedded in the Gazebo simulator by default. As a result, integration of the flexible element into joints needs to be done indirectly. In this chapter, possible solutions are explored and the development of a custom control plugin consisting of the dynamics brought by the passive compliance is proposed. Moreover, the customisation of the control plugin enables the capability of managing ROS topics in a way that a computationally efficient

[1]Work included in this chapter has been presented at the ICINCO 2016 conference with the publication titled "A Compliant Actuation Dynamics Gazebo-ROS Plugin for Effective Simulation of Soft Robotics Systems: Application to CENTAURO Robot" by Malgorzata Kamedula, Navvab Kashiri, Darwin G. Caldwell, and Nikos G. Tsagarakis.

communication can be achieved. The robot linkage representation is structured in such a way so that the robot is split into five sections, and the system can operate user-defined section(s) independently.

The rest of the chapter is organised as follows: Section 4.1 delineates the architecture of the simulator, including the modular description of the robot linkage structure, and the proposed compliant dynamics/control plugin. Section 4.2 demonstrates the simulation results validating the accuracy of the approach propounded in this work, while Section 4.3 presents the conclusion.

4.1 Simulator Design

The ROS middleware has been exploited for the development of the CENTAURO robot simulation platform. It is due to the modularity of this software, and the variety of its supported programming languages, e.g. python, C/C++ and Matlab, that results in a broad usage by the robotics community. The robot simulator is built upon the Gazebo software due to its extensibility through various plugins, broad customisation options, flexible choice of the physics engine and integration with the ROS middleware; thanks to its open-sources architecture. To integrate the simulator environment with the Matlab® software, the Robotics System Toolbox™ is used. A pack of bridge files is generated to manage the specific messages and services corresponding to the robot. Table 4.1 summarizes the simulation software requirements[2].

Table 4.1 The simulator software requirements.

Dynamic Simulator	Gazebo 5 or higher
Middleware	ROS Indigo or higher
Operation System	Ubuntu 14.04 or higher
Optional software	Matlab 2015a or higher
Programming language	C/C++, Python, Java, Matlab

[2]Simulations in this work are performed using a laptop with a processor of Intel® Core™ i5-5200U 4×2.2 GHz, 8 GB RAM, and a Graphics card of GeForce 920M.

4.1.1 URDF Model

Robot Linkage Representation

The Unified Robot Description Format (URDF) representation of the robot is structured according to the robot description (presented in Chapter 3.5), see Fig. 4.1, and the Gazebo simulator associated with the robot linkages is accordingly generated. The robot structure is represented by means of a tree topology with a base link at the pelvis centroid. The tree is constructed on the basis of five branches, four of which are assigned to legs, while the fifth branch establishes the upper-body by connecting the torso to three descendant branches associated with the arms and the robot head.

System Modularity

As the development of control algorithms can be facilitated by an initial implementation on a section of the robot, the robot is structured by six separate sections: pelvis, torso, left and right arms, legs and head. As a result each section can be included or excluded in simulator and control scheme at will. Nevertheless, when an ancestor of a section does not spawn, the descendant branches are disabled automatically; for instance, head can be included provided that torso is not excluded. Furthermore, while the pelvis is connected to a floating-based system by default, it turns to a stationary base when a user grounds the pelvis at will or disables the legs, and the torso can be loaded as a revolute or a fixed joint. The active sections of the structure are therefore selected by setting up the main launch file when assigning *true/false* arguments for different sections, see Fig. 4.2.

Furthermore, developed simulator supports a few visualisation and collision models that differ in the complexity of an underlying geometry, and thus a computational power they require to render. Each implemented visualisation model is automatically available as a collision model, and vice versa. Currently three models are available: a full mesh files, a simplified mesh files[3] and convex hulls, see Fig. 4.3. Finally, a few different end-effectors have been implemented and integrated with the CENTAURO robot simulator; it includes a simulation with no end-effector attached, a stick end-effector and three different robotic hands: SoftHand [108], HERI hand [109] and Schunk Hand[4]. End-effectors for left and right arms are decoupled, and any of the implemented end-effectors can be attached to each hand;

[3]The simplified lightweight mesh files are supported thanks to the MeshLab script provided by University of Bonn.

[4]Schunk Hand simulation has been developed and provided by the University of Bonn.

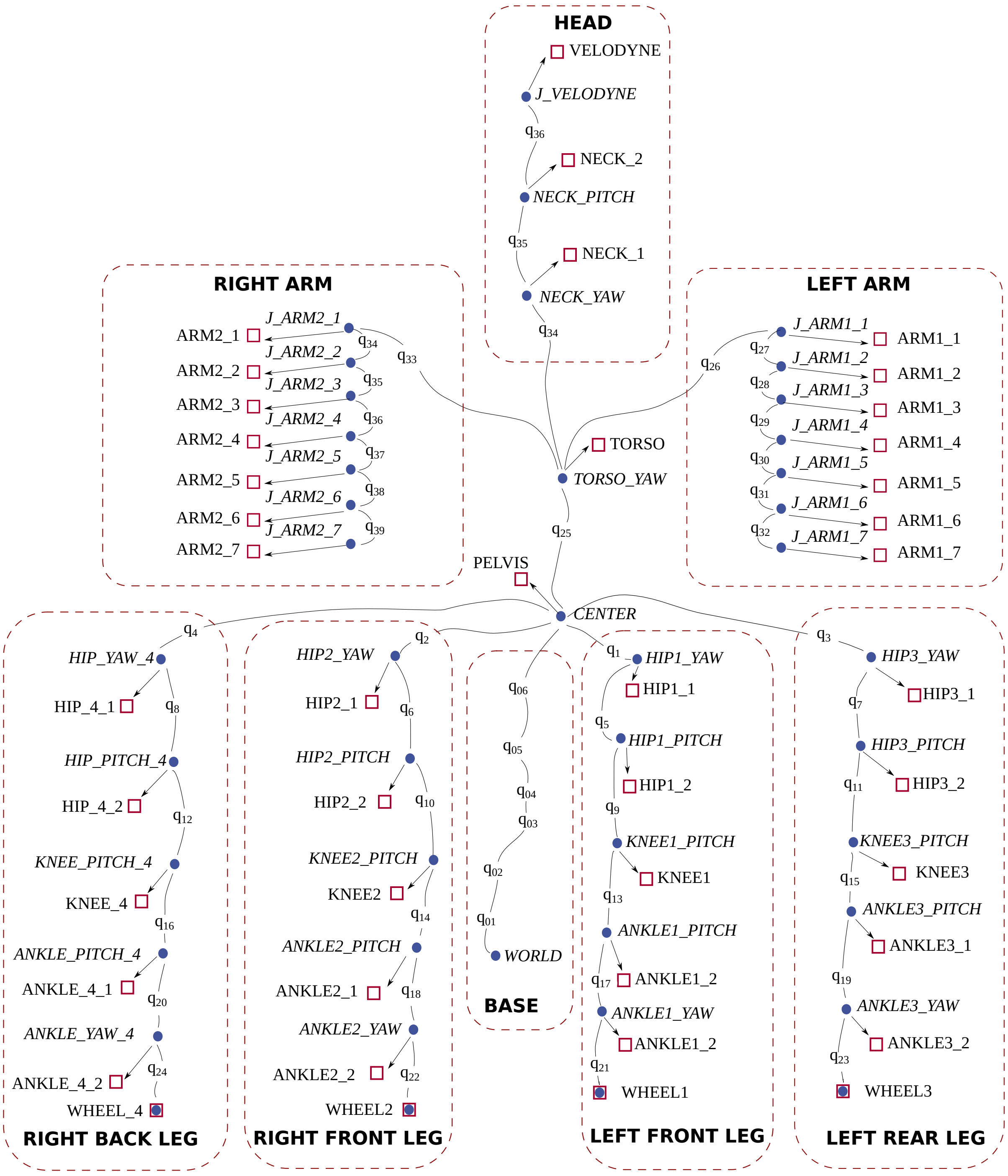
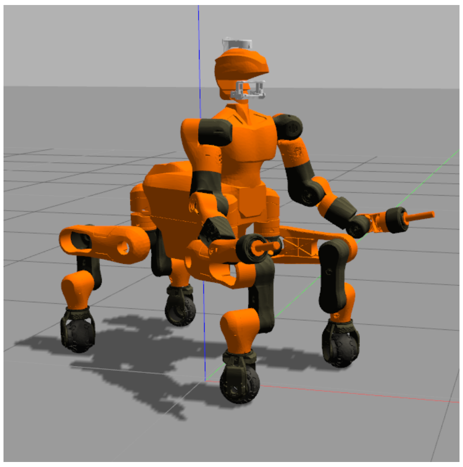
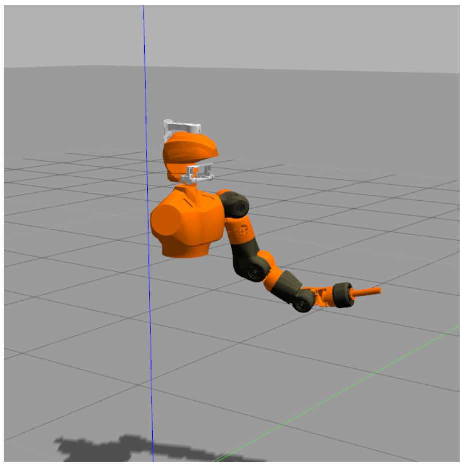


Figure 4.1 URDF file structure for the CENTAURO robot.

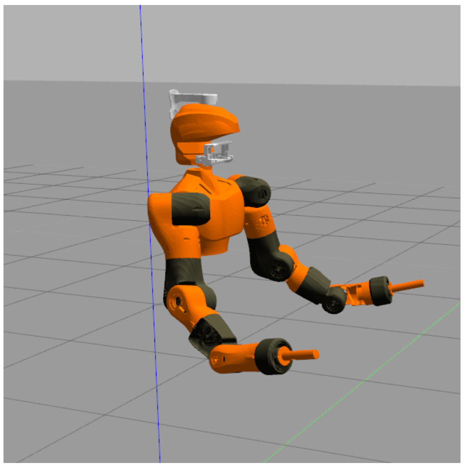
roslaunch centauro_gazebo centauro_world.launch ...



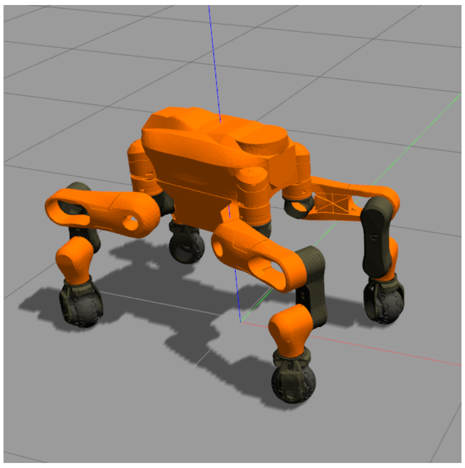
...



... legs:=false
torso:=false
right_arm:=false



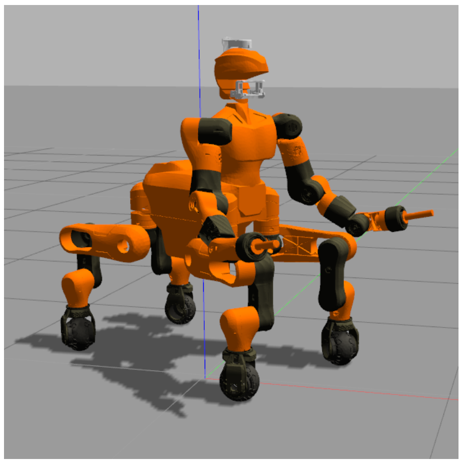
... legs:=false



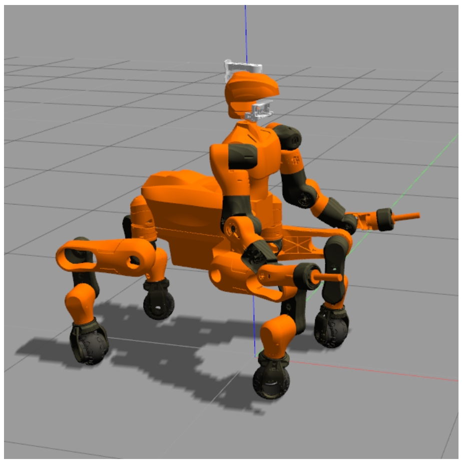
... torso:=false

Figure 4.2 Examples of the CENTAURO robot simulator with different parts of the robot rendered.

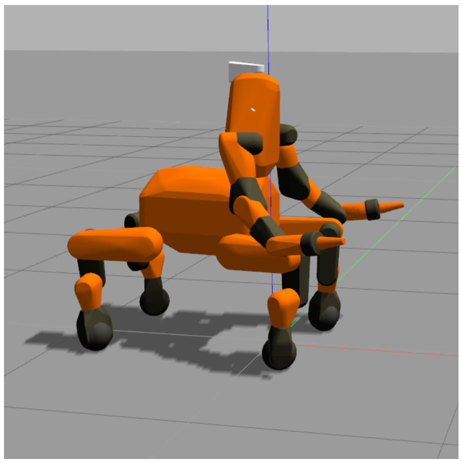
roslaunch centauro_gazebo centauro_world.launch ...



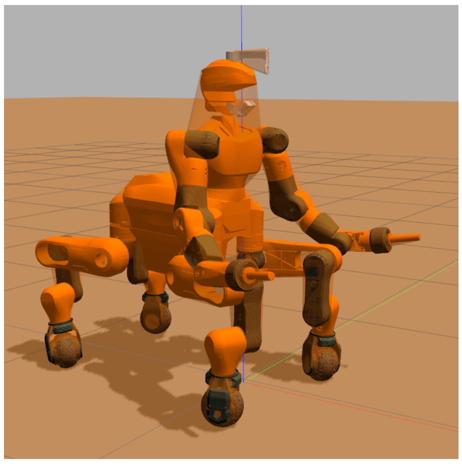
... visual_model:=mesh



... visual_model:=simplified



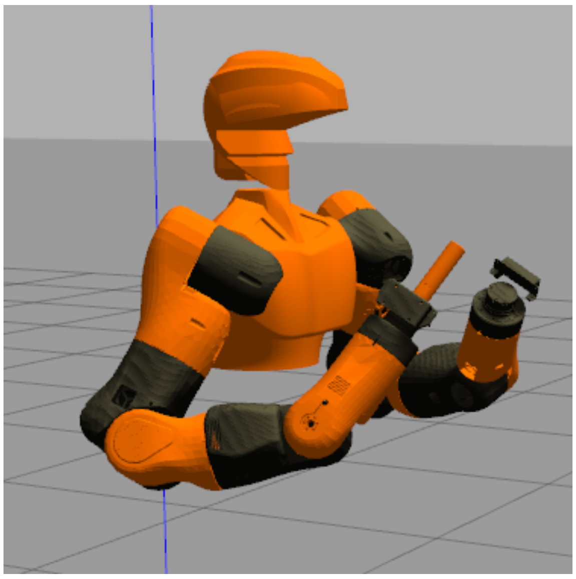
... visual_model:=convex_hull



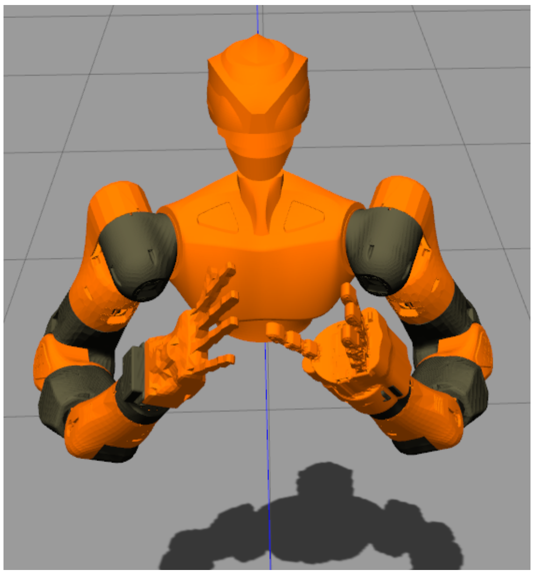
... collision_model:=convex_hull

Figure 4.3 Examples of the CENTAURO robot simulator running with different visualisation and collision models.

```
roslaunch centauro_gazebo centauro_world.launch head:=false legs:=false
```



```
... left_end_effector:=none
    right_end_effector:=stick
```



```
... left_end_effector:=heri
    right_end_effector:=soft_hand
```

Figure 4.4 Examples of the CENTAURO robot upper-body simulator initialized with different end-effectors. On the left: The CENTAURO robot upper-body without an end-effector attached to the left arm, and a stick end-effector attached at the end of the right arm. On the right: The CENTAURO robot upper-body with HERI hand attached at the end of the left arm and SoftHand attached at the end of the right arm.

Fig. 4.4 shows the CENTAURO robot upper-body simulation initialized with different left and right arm end-effectors.

4.1.2 Control Plugin

Multiple Joints Controller

The “ros_control” packages provide the controller interfaces/managers handling the connection between the ROS platform and the Gazebo software (or the real hardware). Given a robot with n_a actuated DoFs, when one uses the default “ros_controllers” packages, the number of threads concerning the lower-level controller is $2n_a$ as each DoF demands a couple of threads for sending and receiving data. Such an approach results in many threads when a robot possesses a large number of DoFs such as humanoids, quadrupeds and centaur-like robots. However, it is burdensome for a system to coordinate executions of numerous small threads; as the large number of threads aggravates system performance and elevates hardware requirements. To address such a potential problem, a control plugin dedicating only two threads to the lower-level of multiple joint controllers of the robot has been developed, while guaranteeing the user-friendly and independent control of individual joints, besides ensuring the synchronisation of data sent/received to/from all DoFs.

The control plugin is structured with two non-real-time ROS topics: the "command", and the "state". One thread manages the data transmitted on the former, that includes three input variables to be subscribed to the controller associated with each DOF. The other thread takes care of the data published to the latter, that comprises a set of output variables. The input variables are reference position and velocity values as well as a centralized torque command, and the output signals are motor-side position and torque states. Moreover, the plugin incorporates five constant gains for each DOF to be set through the ROS service and/or through the ROS parameter server during the controller initialisation [5].

Furthermore, to facilitate the tuning and debugging stages which requires full state feedback of joints, a *debugging mode* is defined to operate in another thread. For a user-defined group of joints, a set of additional data is subscribed to another ROS topic, named "debug". It includes the motor velocities, transmission torques, and position references, as well as position/velocity values of links when received at the controller. The group of joints whose additional data is to be monitored is assigned by means of the ROS parameter server and/or a ROS service entitled "update_print". To set/read the input/output states corresponding to different joints, a unique number is assigned to each joint and is provided for the user by a ROS service "get_joint_names". It is also used for allocating the set of joints whose data is read in *debugging mode*.

The data flow in the system is displayed at Figure 4.5 showing rqt_graph for the simulation. It can be seen that the simulator operates two types of control plugins associated with wheels and compliant joints, named "wheel_controller" and "flexiblejoint_controller", respectively. Thus, support for the multiple joints through the developed ros_control controller decreases the number of threads required to control the 42 DoFs CENTAURO robot from 84 to 4 in the *normal operation mode* and 6 in the *debugging mode*, while providing the additional motor-side state feedback.

[5]Although the control law (3.39) includes only two gains, the implementation of more sophisticated control schemes such as [110] requires a larger number of gains.

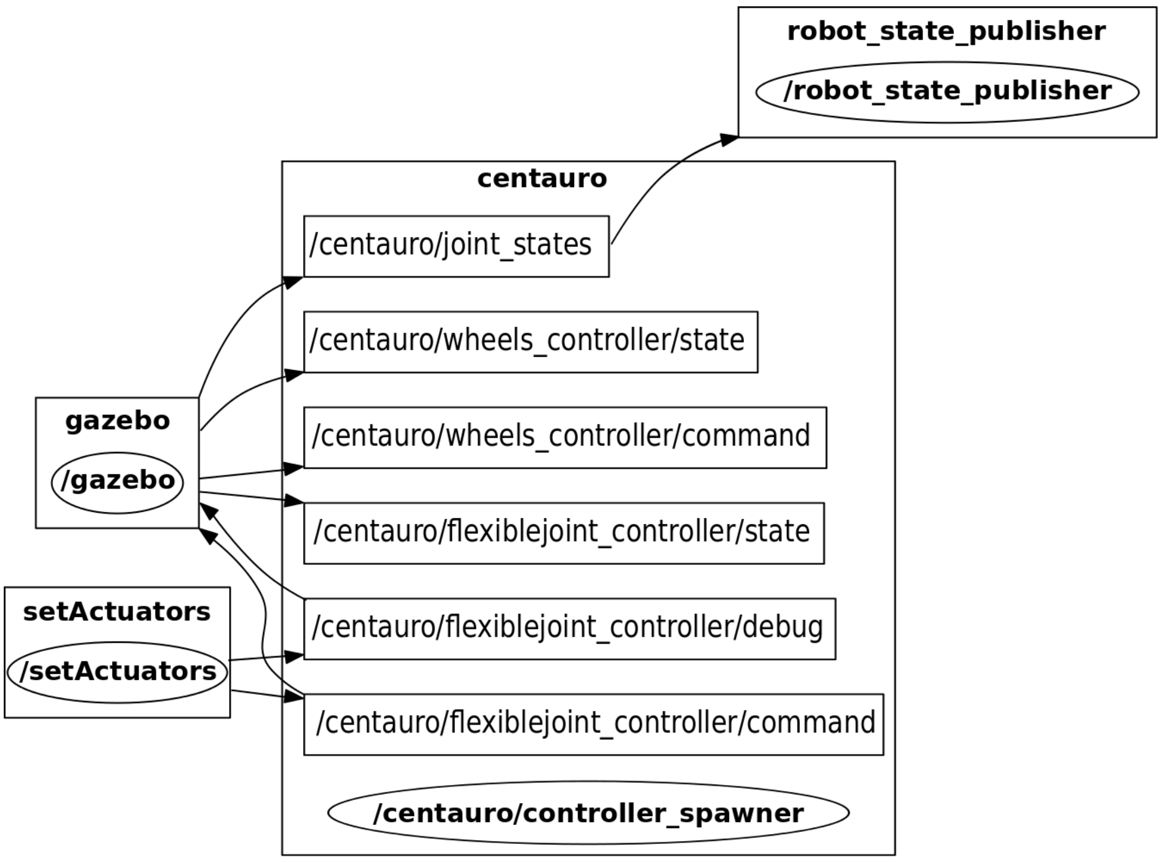

robot_state_publisher
/robot_state_publisher
centauro
/centauro/joint_states
/centauro/wheels_controller/state
/centauro/wheels_controller/command
/centauro/flexiblejoint_controller/state
/centauro/flexiblejoint_controller/debug
/centauro/flexiblejoint_controller/command
/centauro/controller_spawner
gazebo
/gazebo
setActuators
/setActuators

Figure 4.5 rqt_graph of a full simulation depicting system data flow.

Series Elastic Actuator Module

SEAs powers the CENTAURO robot links, while the inclusion of compliance into drive units has not been taken into account in the Gazebo software explicitly. Possible solutions addressing this deficiency are listed as follows

1. Addition of an extra DOF per joint replicating the motor-side dynamics while adding spring–damper forces to link-side joints;
2. Use of a ros_control transmission interface;
3. Employment of a Gazebo model plugin;
4. Definition of an independent ROS node simulating compliant actuators;
5. Incorporation of compliant system dynamics into a real-time controller module on the basis of “ControllerBase” plugin.

Solution 1, however, doubles the number of DOFs that increases the computational burden of the system to a great extent when the robot possesses a large number of DOFs. As for solutions 2 and 3, they cannot directly propagate the motor-side states to the controller while maintaining the minimal control functionality. Solution 4 imposes an unknown delay to the system due to the inclusion of a non-real-time node. It contravenes the synchronisation of motor-side and link-side states, that results in instability of the dynamical system. Hence, solution 5 is selected for the implementation of the compliant actuator as the control scheme is implemented in a real-time module, and the synchronisation of full state feedbacks is therefore guaranteed.

SEA Module Development

The implementation of the SEA dynamics in the control plugin requires a digital form of the corresponding equations. To this end, (3.32) and (3.33) need to be discretised. Since the afore-said equations are related to a set of linear time-invariant systems, they can be expressed by

$$\ddot{\boldsymbol{\theta}}[n] = \mathbf{B}_m^{-1}(\boldsymbol{\tau}_m[n] - \mathbf{D}_m\dot{\boldsymbol{\theta}}[n] - \boldsymbol{\tau}_t[n]), \tag{4.1}$$

$$\boldsymbol{\tau}_t[n] = \mathbf{K}_t(\boldsymbol{\theta}[n] - \boldsymbol{q}[n]) + \mathbf{D}_t(\dot{\boldsymbol{\theta}}[n] - \dot{\boldsymbol{q}}[n]), \tag{4.2}$$

when the Euler method is used. Similarly, the control law, e.g. (3.39), can also be expressed in a discrete form from which the motor torque $\boldsymbol{\tau}_m$ can be derived. The motor-side angular

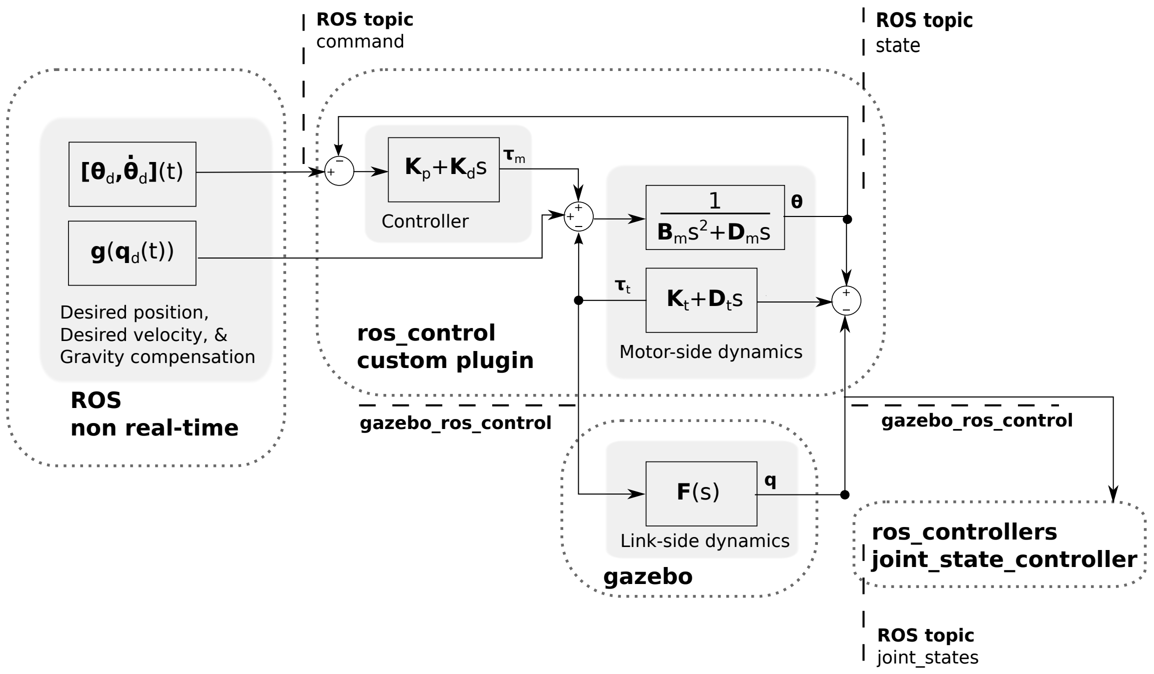


Figure 4.6 Structure of SEA module model and its integration.

accelerations $\ddot{\boldsymbol{\theta}}$ can therefore be computed from (4.1) with the transmission torque $\boldsymbol{\tau}_t$ given by (4.2). The motor-side positions and velocities are thus calculated from the integrations of the acceleration. In this work, the Euler integration method is exploited due to the simplicity of implementation.

From (4.2), it can be observed that the accuracy of the simulation results depends on the both motor-side and link-side data. These readings therefore need to be synchronised in time whereas they are provided by two different sources; the link feedback is given by Gazebo meanwhile the motor-side data is generated in the ROS control plugin. The data flow between the afore-stated sources needs to be designed in such a way so that delays in the data flow that can disturb the synchronisation of the readings are avoided. Hence, the "ros_control" plugin time is coordinated with the Gazebo time. Depending on the stiffness of the passive compliance element, the dynamics of the system can evolve in different frequencies. Discrete representation of such a dynamic system may then cause numerical instability if the sampling frequency f_s used for the discretisation is too low to include the major dynamics variations. Since the energy stored in a discrete system within one sample time is presumed to be constant, the spring motion can converge to a stable state provided that the sampling time is sufficiently small to support this assumption. The simulation of dynamical systems with higher impedance then requires smaller time steps.

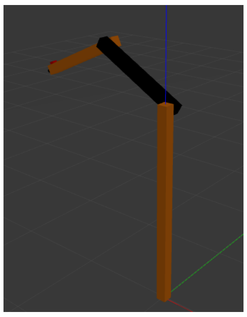


Figure 4.7 Screen-shot from the rrbot gazebo simulation.

The functionality of the simulator in terms of stability and accuracy can therefore depend on the sampling frequency, while it is essential to set the control loop frequency equal to that of the real hardware, so that the results extracted from simulations are comparable with that from experiments on the robot. On the other hand, the controller and the actuator dynamics are implemented at the same thread, and a change in the system sampling frequency f_s affects the controller frequency f_c, while the later may need to be set lower than the former. To this end, the control law setting the motor torque is revised in a way that the motor torque $\boldsymbol{\tau}_m$ is updated at a user-defined frequency f_c. It is expressed as follows

$$\boldsymbol{\tau}_m[n] = \begin{cases} \boldsymbol{\tau}_m[n-1] & \Leftarrow t\,\mathrm{mod}(\frac{1}{f_c}) > \varepsilon \\ \boldsymbol{\tau}_{controller}[n] & \Leftarrow t\,\mathrm{mod}(\frac{1}{f_c}) < \varepsilon \end{cases} \tag{4.3}$$

where t is a simulation time, ε is a small constant, and the operator $a\,\mathrm{mod}(b)$ gives the remainder after division of a by b.

Fig. 4.6 depicts the structure of the simulator with the proposed control plugin, in which the data flow and the role of each part of the simulation framework are also denoted. The linkage dynamics (3.31) is computed by the Gazebo software, and the real-time "gazebo_ros_control" bridge sends the link states to the control plugin and the "joint_state_-controller" from "ros_controllers" packages. The "joint_state_controller" publishes the data to a non-real-time ROS topic named "joint_states" for higher-level control schemes. The real-time "gazebo_ros_control" bridge sends the input torques, i.e. the transmission torques $\boldsymbol{\tau}_t$, to Gazebo from the "ros_control" plugin consisting of motor-side dynamics (3.32), transmission unit (3.33), and the controller (3.39). The desired motor positions and the gravity compensation torques are sent to the control plugin through the non-real-time ROS topic "command", while the ROS topic "state" published the motor states.

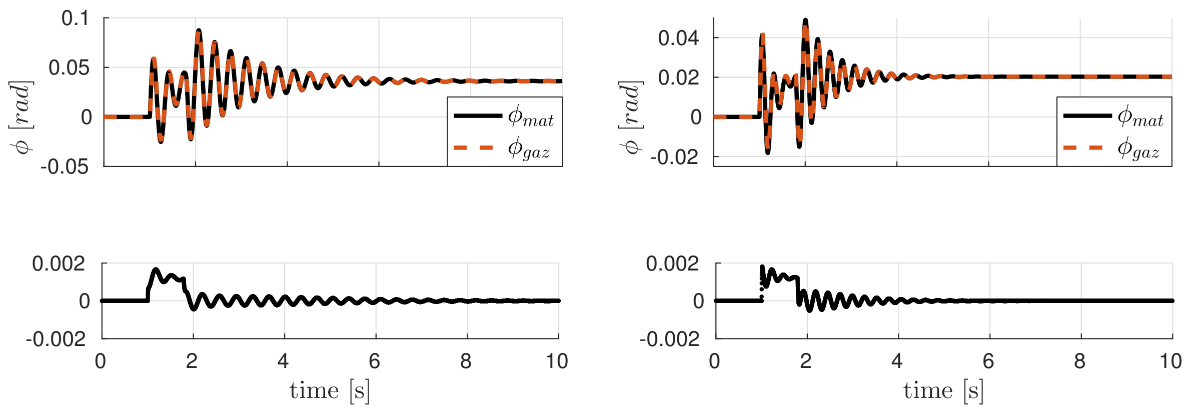


Figure 4.8 Time history of transmission displacement for a given step reference: low stiffness on the left, default stiffness on the right. Bottom plots show the errors between two simulations.

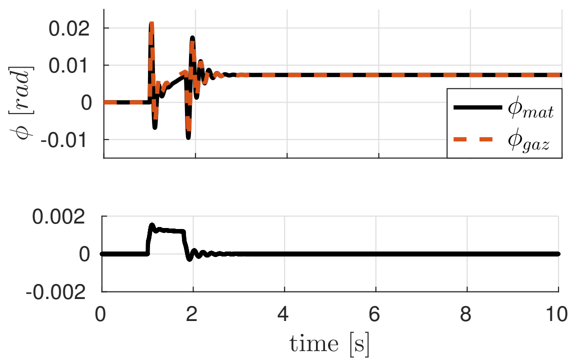


Figure 4.9 Time history of transmission displacement for a given step reference for the high stiffness set-up. Bottom plot shows the errors between two simulations.

4.2 Results

4.2.1 Comparison with Matlab

To validate the above implementation, a comparison between a Gazebo-ROS simulation and a Matlab simulation is presented. To this end, simulation of the last joint-link of the rrbot manipulator shown at Fig. 4.7, available in “gazebo_ros_demos” repository, when powered by a SEA is carried out. The controller gains are set to $K_p = 1000$ and $K_d = 0$ with a saturation limit of $\tau_{max} = 33$ N, and the SEA transmission parameters are $K_t = 180$ Nm/rad and $D_t = 0.5$ Nms/rad. The motor-side inertia and damping reflected to the link-side through the gearing system are $B_m = 0.0742$ Kg/m^3 and $D_m = 24.768$ Nms/rad, where the motor damping value includes both the physical damping and the back-EMF effect. The Matlab

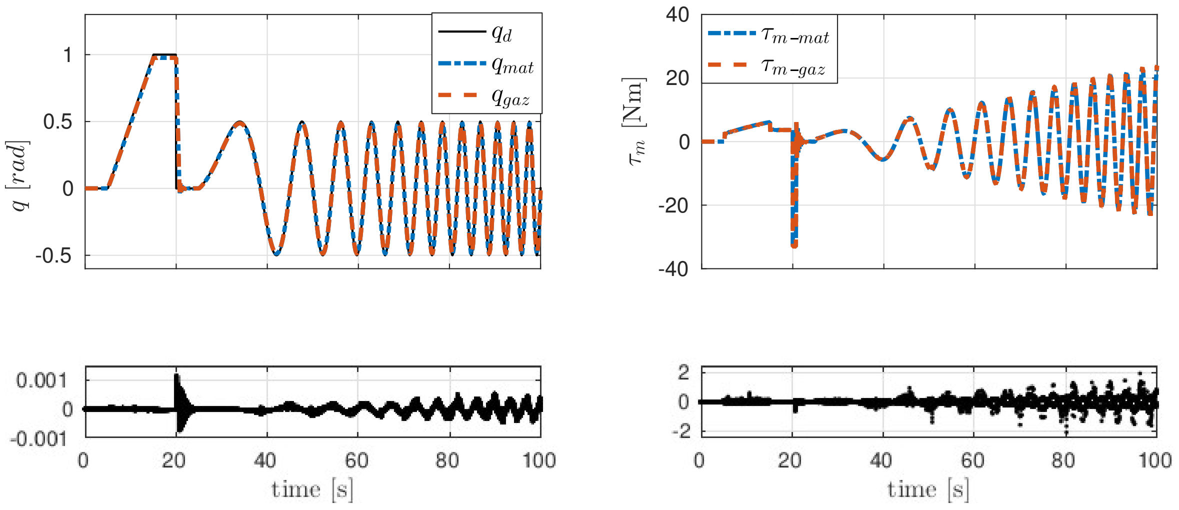


Figure 4.10 Results from the test comparing the proposed SEA dynamics implementation with the continuous Matlab simulation. On the left side there are the link positions and velocities, on the right there are the motor torques. Bottom plots show the errors between two simulations.

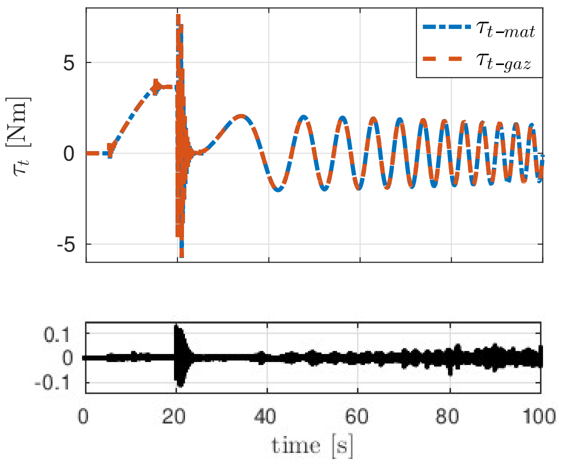


Figure 4.11 Results from the test comparing the proposed SEA dynamics implementation with the continuous Matlab simulation the transmission torques. Bottom plot shows the errors between two simulations.

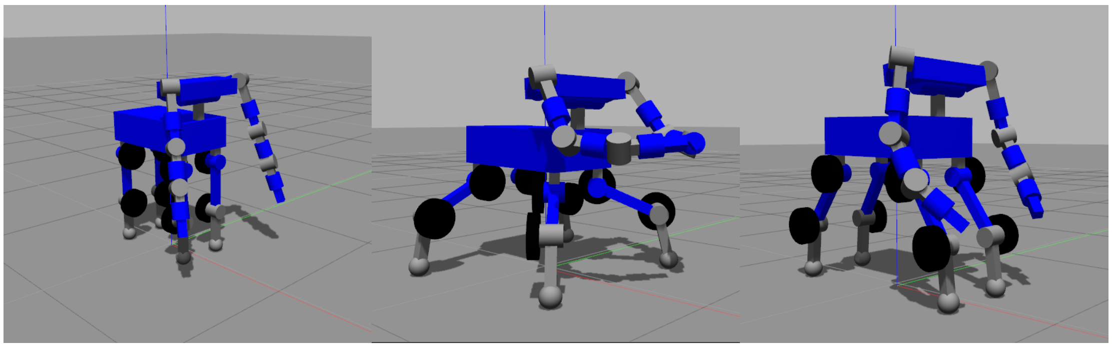


Figure 4.12 Evolution of CENTAURO robot posture. On the left there is the initial position, in the middle there is the spider-like posture, on the right is the mammal-like pose.

simulation was done with a continuous PD controller when using a fixed-step solver at 1 kHz, and the Gazebo simulation was executed using the Bullet physics engine with $f_c = f_s = 1$ kHz. Fig. 4.8 and Fig. 4.9 illustrate the evolution of the transmission displacement when a one radian step is sent as the reference trajectory for the above-said transmission impedance values, in addition to that for two other transmission impedance sets with $K_t = 100$ and 500 Nm/rad when $D_t = 0.3$ and 1.5 Nms/rad, respectively. The second simulation comparison is executed when the reference trajectory is composed of a 0.1 rad/s ramp, a negative step of 1 rad and a chirp signal described by $0.5\sin(0.02\pi t + 0.004\pi t^2)$. Fig. 4.10 and Fig. 4.11 demonstrate the evolution of positions and torques over time when using two simulators with default stiffness/damping values. It shows that the ROS-Gazebo simulator reproduces the Matlab simulator results with 99.9% matching in link position data when the corresponding maximum error and the normalized root mean squared error (NRMSE) are 1.2e-3 and 4.5e-7 rad, respectively. The transmission torque and motor torque may instantaneously show differences up to 0.12 and 2.11 Nm, although the corresponding NRMSE values of 3.6e-5 and 5.3e-4 Nm expresses that such a high error occurs only in a few moments.

4.2.2 Whole-body Simulation

A simulation of the whole robot is presented in this section, when the robot moves from an initial position to a spider-like posture providing larger support polygon and then a mammal-like pose more suitable for dynamic walking tasks. This simulation has been performed on the prototype CENTAURO robot kinematics with the 2-DoFs yaw-pitch torso, two 7-DoFs arms with the same kinematic arrengment as in the final the CENTAURO robot described in Section 3.5, and four 3-DoF yaw-pitch-pitch legs with two sets of wheels attached at the CENTAURO robot hip and knees, see Fig. 4.12. The simulation was executed at $f_c = 1$ and

$f_s = 2$ kHz using the Bullet physics engine. Simulation results of the shoulder and elbow joints of one arm and all joints of one leg are illustrated in Fig. 4.13, Fig. 4.14, Fig. 4.15 and Fig. 4.16. It can be seen that joints which are highly load by the gravity exhibit non-negligible steady state error resulting from low impedance gains of the controller. For instance, the hip yaw joint is not under the gravitational torque, and therefore the position error can converge to zero, while the pitch joints present considerable errors.

4.3 Conclusion

This chapter presented a simulator for the CENTAURO robot powered by SEAs, utilising a Gazebo-ROS framework. While the incorporation of passive elements into the robot actuators using conventional approaches requires the inclusion of extra DoFs, the proposed approach implements the passive dynamics on the control plugin so that the control scheme can have access to both motor-side and link-side readings simultaneously. To achieve this, the "gazebo_ros_control" bridge is focused on, and a custom control plugin is designed in such a way that the dynamics of motor-side and link-side are synchronised. A simulation comparison with Matlab approving the accuracy of the introduced architecture is demonstrated. Finally, the proposed control plugin is exploited in the simulation of the prototype CENTAURO robot possessing 36 DoFs. In the end, it should be noted that the approach propounded in this work can be employed for the incorporation of the dynamics of different actuators such as variable impedance actuators (VIAs) or clutch-based drive units into various robots. The simulation framework presented in this chapter has been used in the development of controllers proposed in Chapter 7 and Chapter 8 for the initial simulation validation.

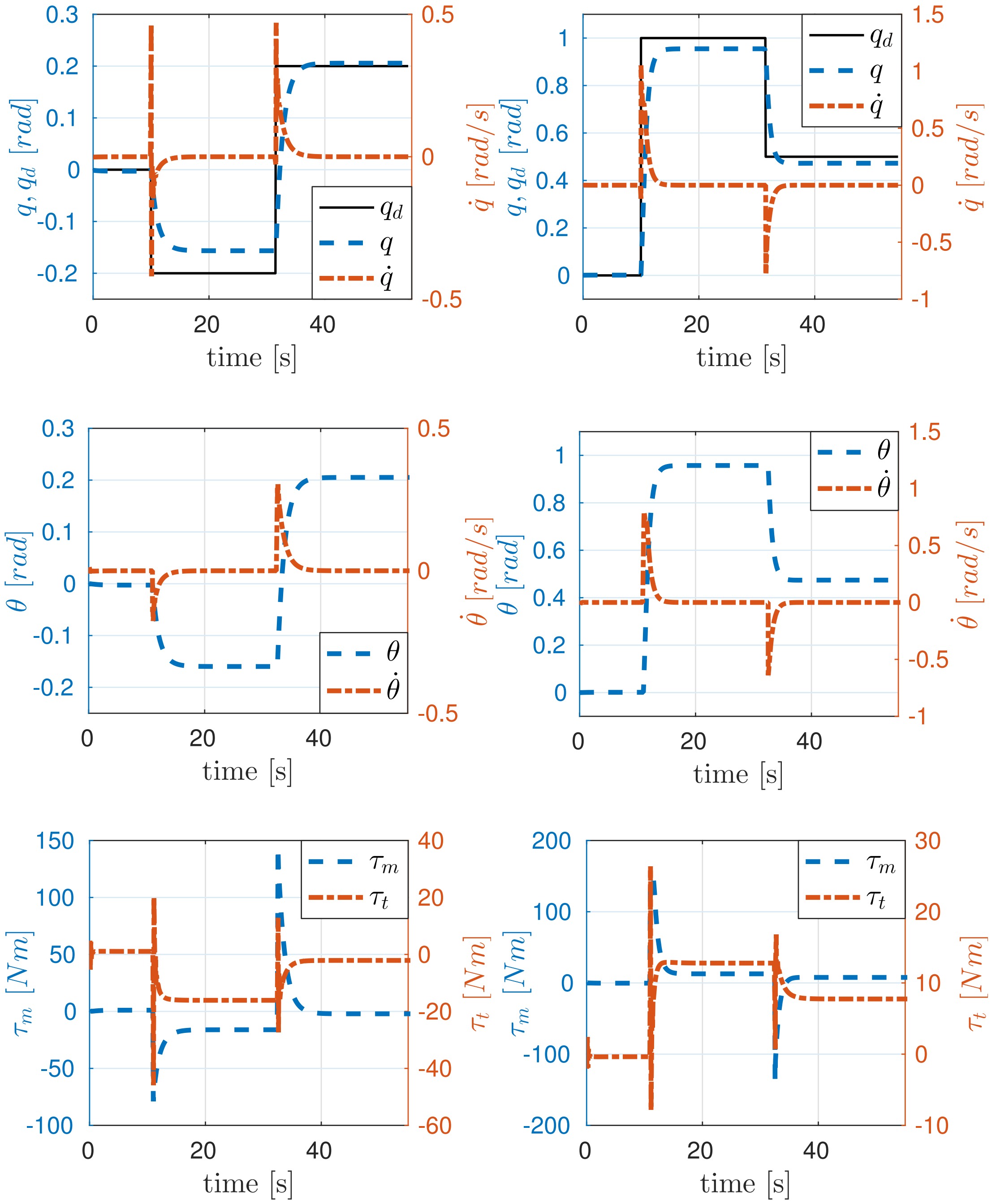


Figure 4.13 The whole-body simulation results for the left arm: the reference and actual link positions, and link velocities on top; the motor positions and velocities in the middle; the motor and transmission torques at the bottom – shoulder joint on the left; elbow joint on the right.

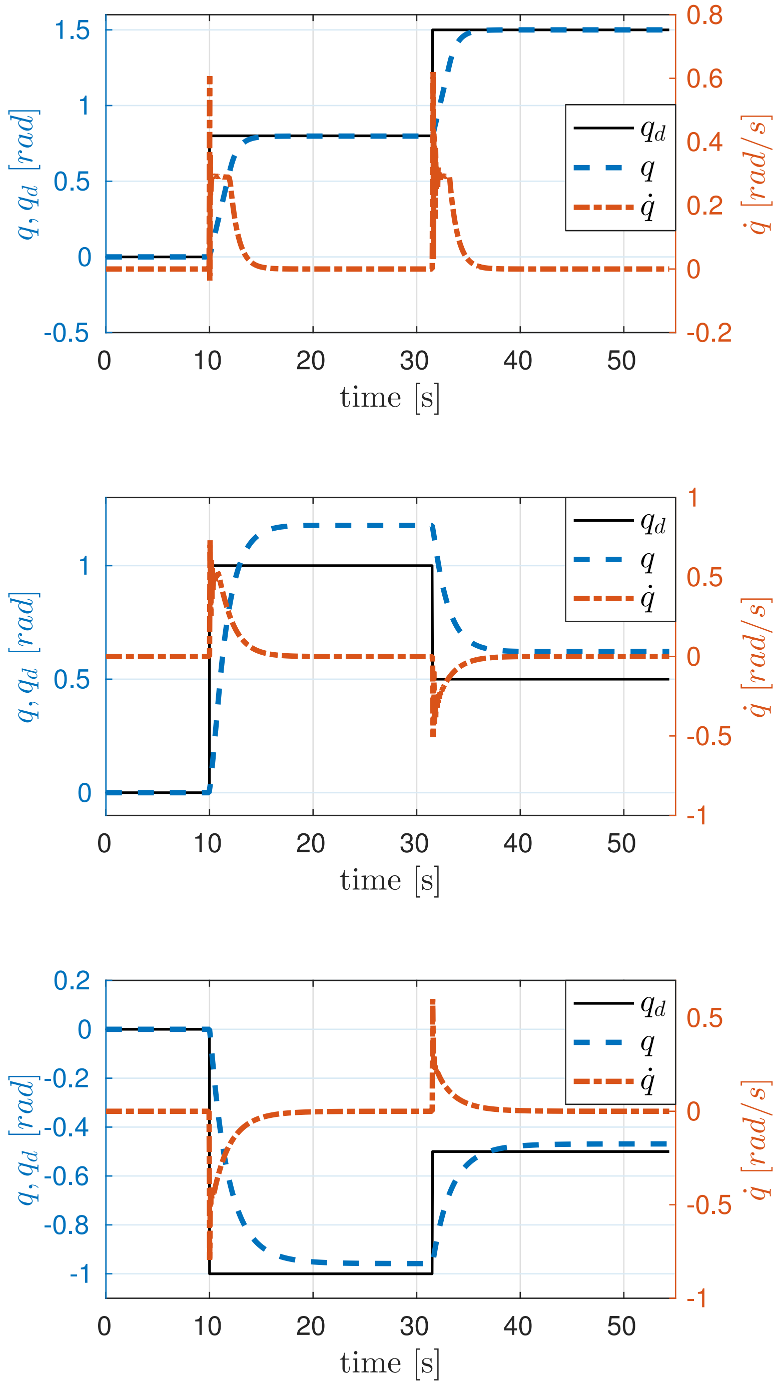


Figure 4.14 The whole-body simulation results for the left front leg: the reference and actual link positions, and link velocities – hip yaw joint on the top; hip pitch joint in the middle; knee pitch joint at the bottom.

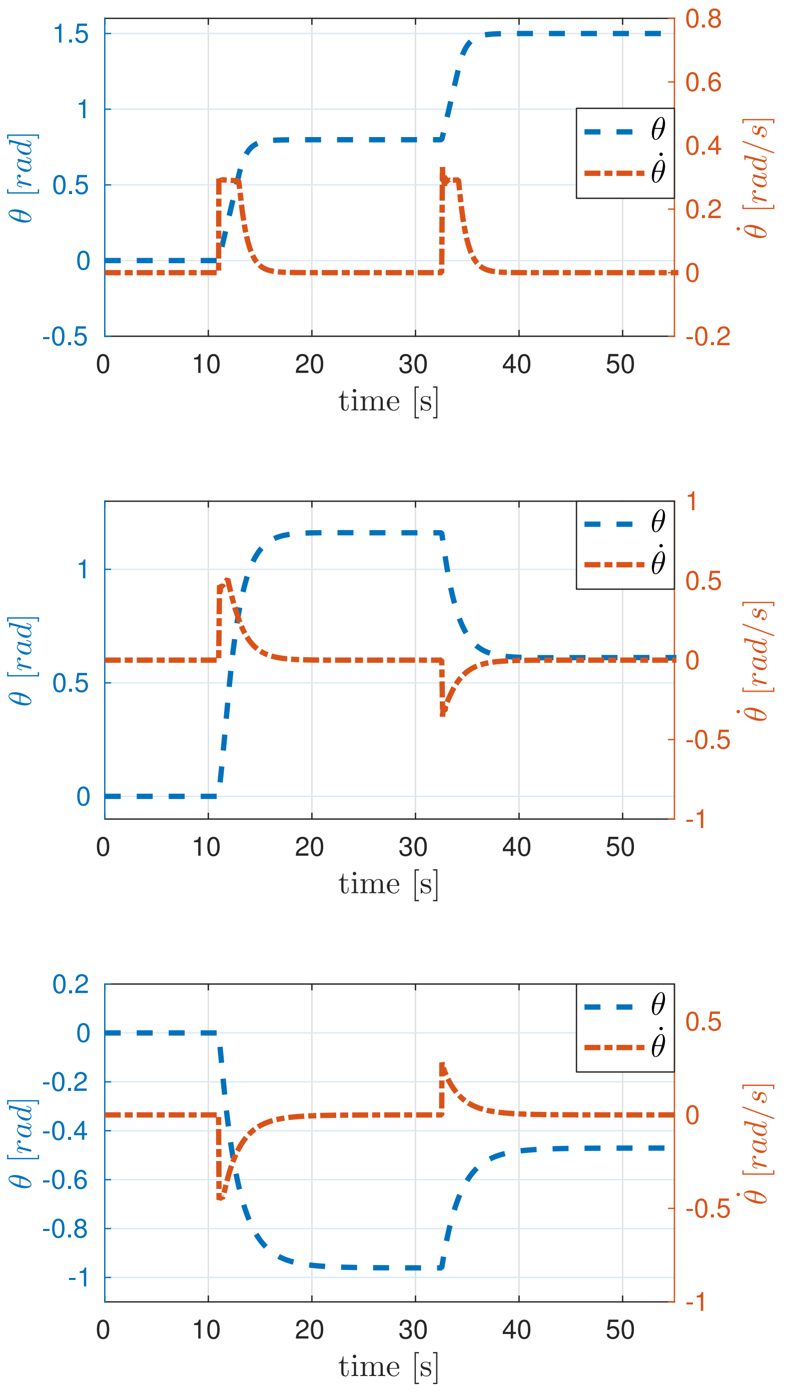


Figure 4.15 The whole-body simulation results for the left front leg: the motor positions and velocities – hip yaw joint on the top; hip pitch joint in the middle; knee pitch joint at the bottom.

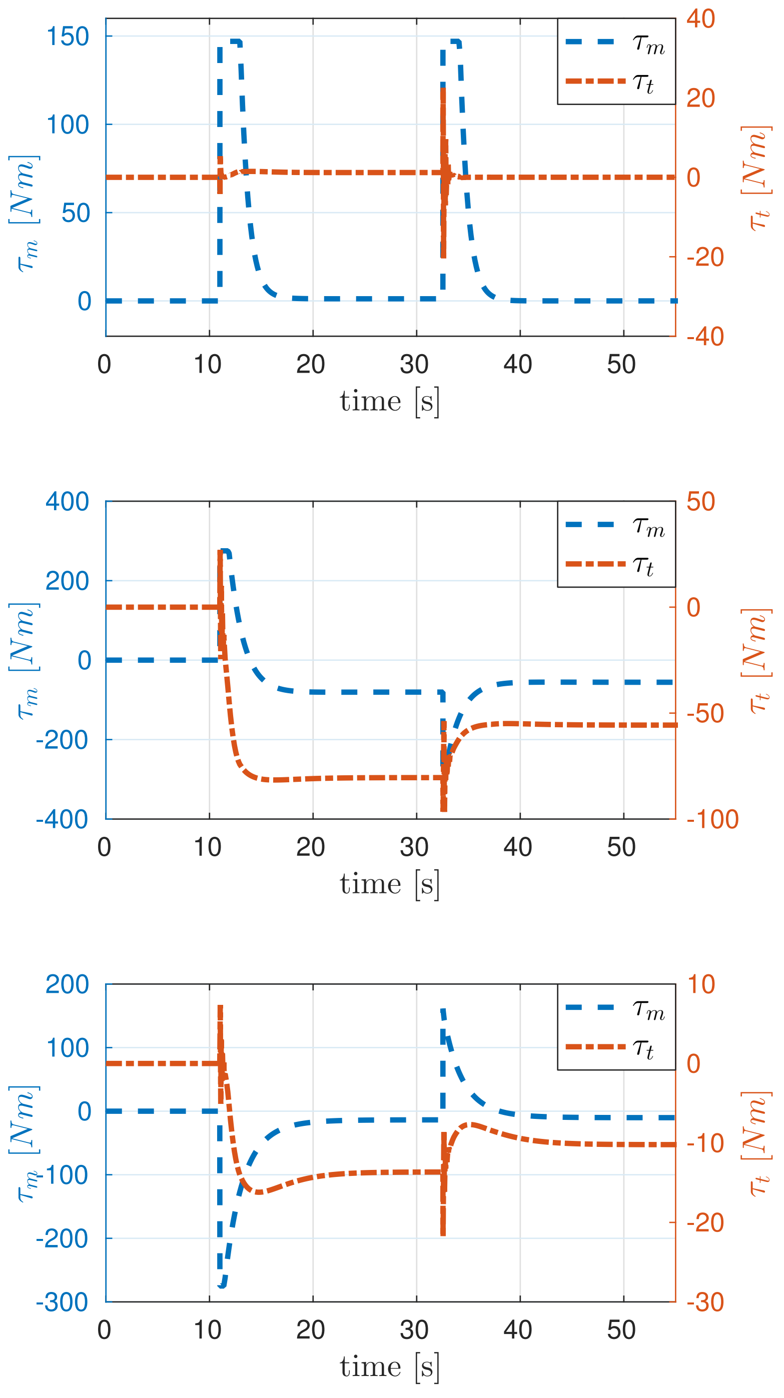


Figure 4.16 The whole-body simulation results for the left front leg: the motor and transmission torques – hip yaw joint on the top; hip pitch joint in the middle; knee pitch joint at the bottom.

Chapter 5

Simplifying Operations in Locomotion - Framework for Efficient Research

A functional robotic platform is a complex synthesis of different cooperating components. Besides the robotic hardware, one of the key elements of the successful control application is a reliable, flexible software layer. In this chapter, a software for a research in locomotion has been developed.

Simplifying Operations in Locomotion - Framework for Efficient Research (SOL-FER) has been developed as a part of this thesis to facilitate research in locomotion. It provides a way to quickly try out ideas by maximizing the flexibility and reconfigurability of the developed modules. It is achieved through the reconfigurable YAML files and a simple end-user API. An abstraction layers for the robot kinematics and dynamics are provided, and a communication layer that consists of the reconfigurable feedbacks/commands has been developed. This communication layer is automatically loaded on the plugin initialization. The configurable feedbacks/commands allow, for example, to run multiple cooperating plugins simultaneously or to switch between the online/offline implementation of the module without any source code modification, and thus without a need to recompile the code. Furthermore, the software provides a unified interface for the non real-time ROS and the real-time XBotCore middlewares. Thanks to the automatically generated plugins, the software can be moved from the simulation to the hardware with one parameter change in the configuration file.

A Robot Points interface has been developed to simplify a definition of the locomotion problems by providing a common interface for different robot characteristics and corresponding Jacobians. With these elements defined through the common interface, a template class has been developed to manage the elements together while ensuring access to individual points if required.

This chapter is organized as follows: First, core components are presented in Section 5.1, the middleware support and algorithms implementation through the SOL-FER are discussed in Section 5.2, and a short summary is given in Section 5.3. Furthermore, the configuarition examples and listings for the main interfaces are cited in Appendix B.

5.1 Core components

In this section, essential elements that create the SOL-FER are described. It inculdes the hardware and kineamatic abstraction layer, and the mapping system described in Section 5.1.1. The abstraction layer for robot dynamics is presented in Section 5.1.2. Then, the concept of the **Robot Points** is proposed in Section 5.1.3; and **Handler** is introduced in Section 5.1.4 that also includes practical examples of the usage of the **Robot Points/Handler** structure.

5.1.1 Robot Structure

A key component of the developed system is a **Robot** structure that contains description of the robot and its state. Furthermore, it coordinates the model and state updates. The **Robot** kinematics is read from the URDF file, and the SOL-FER configuration file describes the robot communication layer. Additionaly, the Semantic Robot Description Format (SRDF) file can be provided to specify a robot subgroups and predefined joint-space states.

Robot Description

The robot is described with the rigid-body kinematic model provided by the RBDL library [111], actuation type and state, gravity vector and state size. Furthermore, mappings between the robot links, joints, state and actuators are provided.

Robot State

Elements of the robot state are split into two categories, the joint-space data and the **Robot Points** that describe the higher-level characteristics of the robot, see Section 5.1.3. The former is implemented through the **State** class that provides the position, velocity, acceleration and torques interfaces by default; more interfaces can be added to each **State** online. The **Robot** structure provides following joint-space states: the current link-side and motor-side states, the desired state and the joint-space lower/upper limits. The **Robot Points** in the **Robot** consist of the CoM, CoP, contact points positions and reaction forces.

Mapping system

Integration of various components is necessary in a functional robotic framework. While the DoFs order in the SOL-FER corresponds to the DoFs order in the RBDL model, the external components often adopt different conventions. To simplify integration of different elements with the SOL-FER, **Bidirectional Map** class that allows to select and sort group of DoFs between two conventions has been implemented. To improve flexibility of the developed components, a few commonly used bidirectional maps are generated automatically at the **Robot** initialization. It includes mappings between the full robot state and

- the RBDL model[1];
- the actuated DoFs;
- the default state feedback[2];
- the lower-level controller[3].

Furthermore, since the order of DoFs in the RBDL model is used as a defualt for the **SOL-FER plugins**, mappings between the robot state and different RBDL models can be initialized from the external URDF files specified in the configuration file.[4] This mechanism is useful to integrate **SOL-FER plugins** that work with only part of the robot (e.g. arms for manipulation plugins or lower body for the locomotion plugins). Finally, an API is provided to add the **Bidirectional Maps** online.

Besides the **Bidirectional Map**, **Unidirectional Map** is implemented to allow user to select a group of the robot DoFs (e.g. wheels, left arm). **Unidirectional Map** can be predefined in the SRDF file through the group tag or added online.

5.1.2 Dynamic Model

Dynamic Model provides an abstraction layer for the robot dynamics. It computes three components of the robot dynamics: the robot bias force, generalized inertia matrix and the inverted inertia matrix. Two dynamic models have been implemented, a full constrained robot dynamics (3.22) and unconstrained dynamics computed with the QR decomposition of the constraints Jacobian (3.36).

[1]It is an identity map, it has been implemented to maintain the consistency in the interfaces.

[2]specific implementation depends on the middleware interface

[3]See footnote [2]

[4]In Appendix B in Code B.1 the example SOL-FER configuration that defines the external RBDL maps is given.

Since computation of the robot dynamics is numerically expensive, to decrease the numerical cost of the integrated control scheme, a subscription-based update system has been implemented. It allows to share the robot dynamic model between the different parts of the program without recomputing it.

At the plugin initialization, each object that relies on the robot dynamics declares how many times per loop each component of the robot dynamics is going to be called; these declarations increase the **call counter**. The **Update Manager** counts the calls from the last model update and upon the update request, it checks if the call counter has been exhausted. As a result, it updates only elements that have been cleared since the last update; it also prevents computations of the robot dynamics components that are not used by the plugin. Thanks to this system, each component can be programmed as an independent module that updates the dynamic model without consideration given to the order of the objects in the final plugin. As a result, the components can be easily rearranged in/added to/removed from the plugin.

5.1.3 Robot Points

Robot Point is an interface to simplify a definition of the locomotion problems by providing a common interface for frequently used variables. It relies on the assumption that most of the control problems can be described through a vector of values and a mapping matrix between two spaces. E.g., rigid-body point can be described by the current position or tracking error and the Jacobian mapping the element from the cartesian-space to the joint-space. The **Robot Points** interface provides the basic algebraic operations between the different **Robot Points** and update method.[5] Through the **Robot Points** following elements are implemented

- Rigid Point Position,
- Rigid Frame Orientation,
- Rigid Body Spatial State,
- Linear/Angular/Spatial Velocity,
- CoM,
- CoP,
- SPV,

[5]The interface code is quoted in Appendix B at Code B.3.

- Constant Point,
- Point and Rolling Contacts,
- Point Difference,
- Point Norm.

5.1.4 Handler

With all elements defined through the **Robot Point** interface, a template class **Handler** has been implemented. It provides methods to handle **Robot Points** updates as a group. It generates a concatenated vector of **Robot Points** values and a concatenated matrix of **Robot Points** mappings, and it ensures access to all individual **Robot Points** through the custom iterators.

Together, **Robot Points** and **Handler**, create simple, yet powerful tandem that can, in an easy, intuitive way, describe variety of robotics motion control problems. Thanks to an extensive use of templates and simple interfaces, elements implemented through **Robot Points** are reusable, and integrate naturally with the software previously developed with the **Robot Points/Handler** pattern. For example, a simple one task inverse kinematics problem (3.43) comes to Code 5.1[6]. A more complex example is given at Code 5.2, where a pseudo-code for implementation of the IK for the robot CoM tracking task and the position tracking task of the robot hands relative to the torso is shown in Code 5.2. Note that the update methods are the same in the two examples, and the reference task in Code 5.2 only adds the task reference number. The declaration in Code 5.2 replaces the **Robot Points** with the **Handlers**, and the **Linear Point** (Code 5.1) is replaced by the **Robot Point** (Code 5.2) to support variety of tasks besides position tracking of a point on a rigid-body. The basic code of the task initialization is the same in the both examples; however, in Code 5.2 dynamic number of linear point tracking tasks is generated. Finally, note that in Code 5.2 the size of the '_tasks' **Handler** is bigger than the number of tasks as it also contains the support **Robot Points** for the relative tracking task. The order of the points in the '_tasks' **Handler** ensures that all lower-level **Robot Points** are updated before the higher-level, dependent ones. This chaining capability of the **Handler** update method allows to further expand the reusability of the implemented **Robot Points**. In Appendix B, in Code B.2, additional example is given for the program that computes the euclidean norm between the robot hands and the robot reference point.

[6]The C++ pseudo-code in the examples accurately represents SOL-FER API.

```
protected: // variables declaration
       // Robot Point that computes the position and Jacobian of the point on the rigid-body
       LinearPoint _task;
       // Constant Robot Point that implements the 'set' method to set the state by hand
       Constant _reference;
       // Minus Robot Point that subtracts two Robot Points
       Minus _ik;
public:
// memory allocation, initialize inverse kinematics for the end-effector tracking
function init(Robot& robot, string pointName = "EndEffectorName"){
       // create the end-effector position point
       _task = LinearPoint(pointName, robot);

       // create a reference point of a size to the task '_task'
       RobotPoint _reference = Constant(_task.size());
       // set the reference jacobian to zero for the Minus robot point update method
       _reference.set(Matrix::Zero(_reference.rows(), _reference.cols()));

       // initialize the tracking error computation
       RobotPoint _ik = Minus(_task, _reference);
}

function update(){ // control loop
      // update current end-effector state
      _task.update(true); // true means update jacobian

      // update an inverse kinematics
      _ik.update(); // jacobian_update=true
      _q = inverse(_ik.getJacobian())*_ik.getState();
}

      // update the reference
function reference(const Vector& x_des){ _reference.set(x_des);}
```

Listing 5.1 Pseudo-code for the **Robot Points**, **Handler** implementation of the IK scheme for the point position tracking, with example initialization for and user specified end-effector.

```
protected: // variables declaration
    // Handler of Robot Points to support any IK task
    Handler<RobotPoint> _tasks;
    // Handler of references, it is specialized for Constant Robot
     ↪  Point to have an access to the set method
    Handler<Constant> _reference;
    // Handler of the Minus points to compute the tracking errors
    Handler<Minus> _ik;
public:
// memory allocation, initialize inverse kinematics for a given Robot,
 ↪  and requested end-effectors
function init(Robot& robot, str points="hands", str root="torso"){
// initialize the COM tracking task with the COM embedded in the Robot
    _tasks.add(robot.COM());
    // initialize reference and task
    _reference.add(Robot.COM.size());
    _ik.add(Minus(_tasks[-1],_reference[-1]);
    // add the Linear Point of torso
    _tasks.add(LinearPoint(root));
    // functionined the task for each point in the Unidirectional Map
     ↪  requested specified the the user
    for (auto& point: Robot.getLinks(points)){
        // add Linear Points
        _tasks.add(LinearPoint(point));
        // compute a state of the relative point
        _tasks.add(Minus(_tasks[-1], _tasks[0]));
        // initialize reference and task
        _reference.add(Constant(_tasks[-1].size()));
        _ik.add(Minus(_tasks[-1],_reference[-1]));}
    // init the reference Jacobians
    for (auto& point: _reference) point.set(Matrix::Zero(point.rows(),
     ↪  point.cols()));}
function update(){ // control loop
    // update all the support points
    _tasks.update(true); // jacobian_update=true
    // update an inverse kinematics
    _ik.update(true); //  jacobian_update=true
    _q = inverse(_ik.getJacobian())*_ik.getState();}
function reference(Vector& x_des, int i){//reference update in task i
        _reference[i].set(x_des);} // access Constant set method
```

Listing 5.2 Pseudo-code for the **Robot Points**, **Handler** implementation of the IK scheme for multiple aggregated tasks. Example for the IK: CoM tracking and the position of the ’n’ hands with respect to the torso.

Dynamic Points

The **Robot Point** interface well fits the robot first-order kinematics that is described by the linear equations in the form $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. However, it is not sufficient to describe the second-order kinematic and dynamic problems that read $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$. To support the higher-order robot description, the child **Dynamic Point** interface to the **Robot Points** interface has been introduced. **Dynamic Point** has an additional offset member; and it provides an interface for the following elements

- linear, angular and spatial acceleration for the rigid-point/body,
- point/body force, torque and wrench,
- SPV acceleration.

5.2 Plugins

The SOL-FER adopts a plugin system that exploits two software layers to provide the support for different middlewares. The first layer consists of the **components** that are middleware independent implementations of specific algorithms/programs. Each component has to inherit from the **Base** class and adhere to the provided interface[7]. The second layer consists of **SOL-FER plugins** that generate the middleware dependent components including the communication layer. They also coordinate the kinematic updates of the **Robot** instance. Dynamically loaded plugins for all supported middlewares are auto-generated from the plugin template specialization that is required for each **Component**. In Appendix B in Code B.14, an example of the template specialization for the joint-space controller module is given.

Remainder of this section is organized as follows: First, the SOL-FER configuarition files are described in Section 5.2.1, then a few exmples that highlight felxibility of the developed **Plugin/Component** structure are shown in Section 5.2.2, and the **shared plugin** is introduced in Section 5.2.3.

5.2.1 Configuration File

Plugin loads the communication layer and the **Robot** instance based on the URDF/SRDF files defined in the configuration file. The communication layer depends on the middleware, and, after ROS, adopts a strict publisher-subscribes pattern. Each connection between two plugins is refereed to as a **pipe**.

[7]as shown at Code B.13 in Appendix B

Library of pipes

The communication layer for the robot current/desired **States** is specified in the configuration file. To that end, a configuration for each **pipe** has to be given in the configuration file creating a library of predefined **pipes** the plugins can load from. A **pipe** configuration consists of: a type of information a **pipe** transfers, affected DoFs, **State** interfaces the **pipe** publishes for/subscribes to, robot **State** the **pipe** should be attached to and the **Bidirectional Map** that should be used for mapping.

One of two information types can be passed through the auto-generated **pipe**: joint-space values (see Code B.4 in Appendix B for an example configuration) or the state of the robot rigid-body (see Code B.5 in Appendix B for an example configuration). Other types of **pipes** are not supported for the auto-generation. If plugin specific higher-level communication not supported through the configuration file is required, it can be defined in the plugin template specialization **_initCallbacks** method (see Code B.14 in Appendix B).

The affected DoFs are described by providing the information, which robot chain should be considered (all, SRDF group tags), and what type of DoFs is affected (e.g. all, actuated, body, **Bidirectional Map** name). If the body type is used, a link name has to be provided. Finally, the 'function' tag describes **State** a **pipe** is attached to, where options are 'state' for the robot current state and 'reference' for the robot desired state. When the **pipe** provides a state of the rigid-body, a convention used to describe it has to be specified (Code B.5, Appendix B).

Finally, a middleware specific configuration is required (see Code B.6 in Appendix B). For example, ROS requires type and topic of a message, and – due to the callback nature of the ROS communication layer – an information if given pipe has to be initialised before the component initial conditions can be set. On the other hand, XBotCore needs the software layer the pipe communicates with and the **pipe** name. Some more specific tags may be required depending on the **pipe** loaded; Table 5.1 and Table 5.2 provide overview over implemented **pipes** that can be auto-generated for ROS and XBotCore, respectively.

Plugin Configuration

Plugins are loaded based on the configuration specified in the SOL-FER configuration file; a minimal example is given at Code B.7. Each plugin configuration is identified by a unique name ('joint_online' in the example). For XBotCore plugins this name has to correspond with identifier hardcoded in a template specialization required for each component, as the middleware does not provide a solution to pass additional arguments on the plugin start-up.

ROS			
type	input	output	comments
join-space	NRT	lower-level	ros_control, impedance controller
join-space	NRT	lower-level	ros_control, velocity controller
join-space	lower-level	NRT	ros_control, state feedback
rigid-body	lower-level	NRT	simulated floating base state
join-space	lower-level	NRT	simulated contact forces
rigid-body	NRT	NRT	pass **Robot Point** state
joint-space	NRT	NRT	pass joint-space state in ROS

Table 5.1 Overview over ROS pipes implemented for the auto-generation in the SOL-FER plugin.

XBotCore			
type	input	output	comments
join-space	RT	lower-level	desired state
join-space	lower-level	RT	state feedback
rigid-body	lower-level	RT	imu readings
joint-space	RT	RT	shared memory
rigid-body	RT	RT	**Robot Point** state, shared memory
rigid-body	RT	NRT	rigid-body state
joint-space	NRT	RT	from ROS topic
joint-space	RT	NRT	publish to ROS topic

Table 5.2 Overview over XBotCore pipes implemented for the auto-generation in the SOL-FER plugin.

In ROS, by default the same configuration corresponding to the hardcoded plugin identifier is loaded. However, the user can load a plugin with any configuration by passing an optional command line argument ’n’ when launching the plugin. The command to run the ROS plugins read

```
rosrun plugins plugin -p plugin_identifier -n config_name
```

A plugin configuration has to specify a layer the plugin operates in, what determines types of **pipes** loaded, and a secondary configuration file[8] that describes parameters dependent on the software layer. These consist of control loop frequency and filters tuning. A plugin configuration also specifies a plugin mode, where the ’full’ mode operates with the full communication layer, while the ’idle’ mode prevents the lower-level commands from being send to the robot. In the ’idle’ mode plugin still receives the system state, but it does not control the robot. The ’idle’ mode is useful to collect the data, debug the code with the real robot state or to check a plugin for the memory allocations. A plugin configuration also contains declarations if plugin relies on the robot kinematics/dynamics, and if it modifies the robot state internally. These flags are used to coordinate updates on **Robot** instances. Finally, the ’controller’ and ’robot’ parameters define the plugin communication layer. The ’controller’ tag consists of list of **pipes** the plugin publishes to, or a name of a predefined set of pipes that can be declared in the ’controllers’ tag[9]. The ’robot’ tag specifies one of predefined configurations to be loaded. These ’robot’ configurations describe the list of subscribers the **Robot** receives the feedbacks from, and if the robot actuation models and contacts are required by the plugin[10]. Similarly to the publishers list in the ’controller’ tag, the lists of subscribers can also be predefined in the ’feedbacks’ tag [11]. In the end, the configuration for each plugin is passed to the **Component** constructor, and so any custom tags can be added to tune/set-up the **Component**.

Secondary Configuration Files

With an extensive use of the libraries of the predefined elements in the configuration files – the library of pipes, publishers/subscribers layers and the **Robot** configurations – range of the different plugins arrangements can be achieved with a minimal change in the configuration file. However, often only a small change in a given predefined configuration is required for a desired configuration. Rather than maintaining multiple full configuration files with a one/two

[8]See Section 5.2.1 for details on the secondary configuration files.
[9]See Code B.9 in Appendix B for an example of the predefined pipeline set-up.
[10]An example of the online and offline ’robot’ configurations is provided in Code B.8 in Appendix B
[11]See Code B.9 in Appendix B for an example of the predefined pipeline set-up.

```yaml
modules:
        joint_online: # plugin name/configuration id
            layer: NRT # layer the plugins runs at
            robot: joint_space # which robot configuration should be
             ↪  loaded
            controller: direct # which controllers should be loaded
            mode: full # if 'full' all controllers are loaded, if
             ↪  'idle' the desired states are not send to the
             ↪  lower-level controller
            kinematics: false # does it require kinematic update
            dynamics: false # does it require dynamic update
            model_change: false # does it modify the robot state
             ↪  internally
```

Listing 5.3 Example of the plugin configuration for the joint-space controller.

lines difference, a secondary configuration files have been introduced. The secondary files are merged with the full configuration file with the priority given to the secondary file tags when the two overlap. Thus, the secondary configuration files allow to overwrite any element of the original configuration file and add tags not present in the original file. Secondary files are specified for each plugin independently by an optional 'secondary_file' tag in the plugin configuration.[12]

Two sets of secondary files are used by default in the SOL-FER. These files load the configuration specific for the software layer the plugin operates in (i.e., control loop frequency) and the configuration specific for the hardware/simulation (i.e., sensors calibration). These files can be expanded to add plugin configuration specific for the software layer or the hardware/simulation (e.g., filter tunning).

Furthermore, the secondary files simplify integration of the SOL-FER with the XBotCore. This mechanism allows to add only one parameter to the XBotCore configuration file – a path to the SOL-FER configuration file – to run multiple XBotCore plugins. Otherwise, starting multiple XBotCore plugins with contradicting parameters would require specifying and maintaining a SOL-FER configuration file for each XBotCore plugin independently. Examples of the practical application of the secondary files for the manipulation and state estimation plugins are given in Appendix B in Code B.11 and Code B.12, respectively.

[12]See Code B.10 in Appendix B for an example of the a plugin configuration with a secondary file.

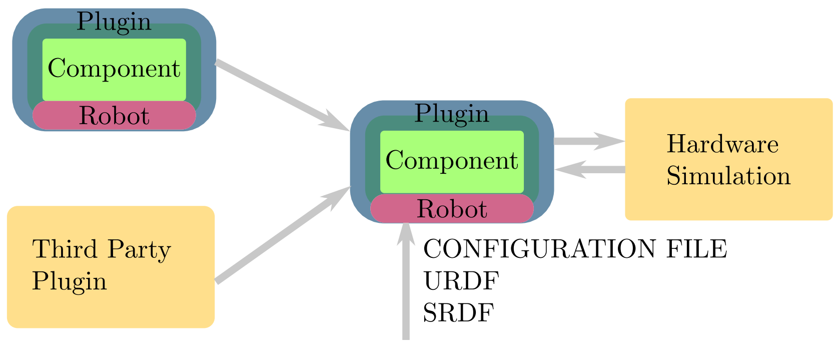


Figure 5.1 Scheme that visualises the **Plugin–Component** structure and the **Plugin** interaction with the middleware. Pink areas show loaded **Robot** configuration, blue areas indicate the **Plugin** space which green interiors represent the **Component** space. Yellow areas symbolise the third party elements.

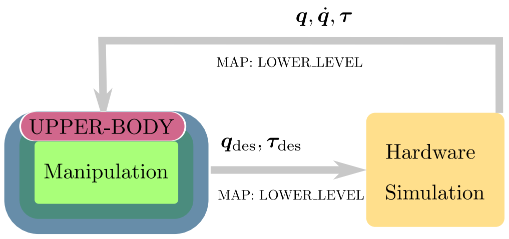

$\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}$
MAP: LOWER_LEVEL
UPPER-BODY
Manipulation
$\boldsymbol{q}_{\text{des}}, \boldsymbol{\tau}_{\text{des}}$
MAP: LOWER_LEVEL
Hardware
Simulation

Figure 5.2 An example of the plugin communicating with the lower-level control. Pink area marks loaded **Robot** configuration, blue area indicates the **Plugin** space which green interior represents **Component** space. Yellow area symbolises the third party element - hardware/simulator.

5.2.2 Examples

Fig. 5.1 visualises the two layers **Plugin–Component** structure, and Fig. 5.2 presents an example of the manipulation plugin controlling the robot upper-body, where the mapping between the upper-body and the lower-level controller is generated automatically in the plugin layer. The automatic map generation based on the robot kinematics specified in the configuration file means the same plugin can be used to control different parts of the robot without any modifications in the **Component/Plugin** source code (e.g. gravity compensation, joint-space control, IK, Inverse Dynamics (ID)). Furthermore, definition of the communication layer through the configuration files allows to switch between the online and offline plugin implementation without any modification of the source code, see Fig. 5.3 for a concept scheme.[13].

[13]In Appendix B in Code B.8 the corresponding robot/feedbacks configurations are given

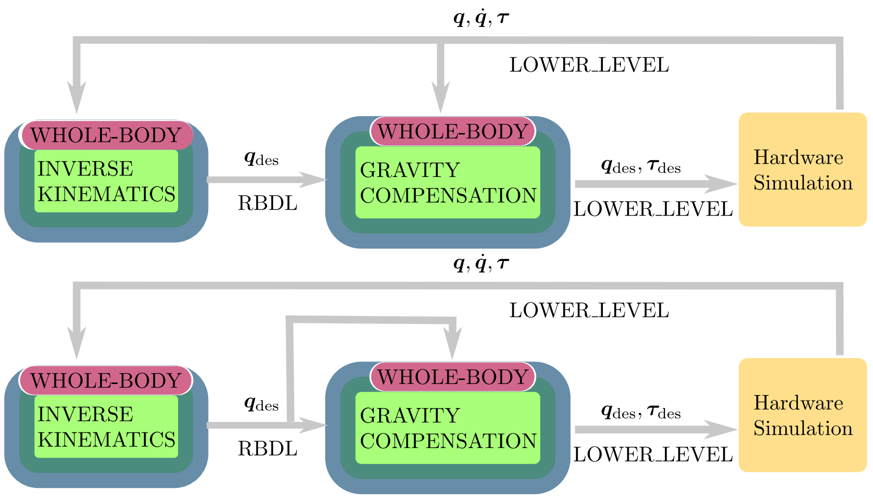


Figure 5.3 An example of the configuration for the online/offline plugin configuration. Pink areas mark loaded **Robot** configurations, blue areas indicate the **Plugin** space which green interiors represent **Component** space. Yellow areas symbolise the third party elements - hardware/simulator.

Shared control

For a complex robots with multiple kinematic chains, only part of the robot structure may be required to solve a given task. In these cases, to speed-up the computations, a simplified, partial robot model is typically used. Thanks to the flexible communication layer and the mapping system (see Section 5.1.1), multiple plugins that control specific parts of the robot structure can cooperate without interference. Fig. 5.4 shows an example of the locomotion and manipulation plugins working simultaneously controlling the robot upper-body and lower-body structures, respectively. In this example, the gravity compensation plugin that operates on the whole-body model maps partial – lower-body and upper-body – solutions to the full robot state and sends the whole-body command to the firmware. Note, that in this example any component[14] that loads the whole-body model could be used to merge the partial commands.[15] That is because the pipes are combined at the auto-generated plugin layer. However, although not necessary, the merging component is highly recommended to – for safety reasons – have only one plugin that sends commands directly to the lower-level. Depending on the communication layer design in the underlying middleware two plugins sending partial state commands to the lower-level may or may not be supported.

[14]including an empty component that does not implement any algorithms

[15]To be more specific, a full whole-body model is not a requirement. Any model that includes all DoFs controlled by other plugins is sufficient. Otherwise, the DoFs not represented in the final model would not be parsed to the lower-level.

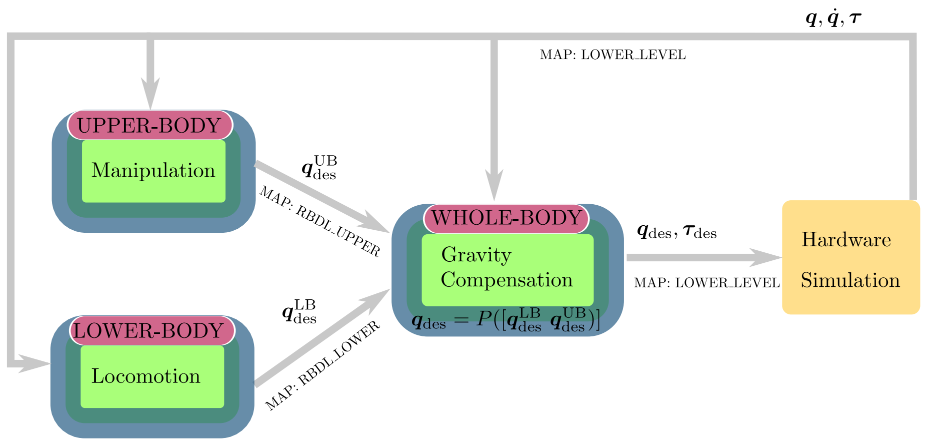


Figure 5.4 An example of the shared control structure. Pink areas mark loaded **Robot** configurations, blue areas indicate the **Plugin** space which green interiors represent the **Component** space. $P()$ represents the permutation operation, and yellow area symbolises a third party element - hardware/simulator.

5.2.3 Shared plugin

With the basic ROS/XBotCore plugins, elements share the states through the **pipes**, but each plugin runs as an independent, self-sufficient program. In practice, when running multiple plugins, the same robot model/feedback configuration is often used multiple times. In such cases, number of updates on the robot kinematics/dynamics could be reduced if the plugins would share the same **Robot** instance. To that end, a **shared plugin**, for both ROS and XBotCore middlewares, have been developed to coordinate allocation and updates of the **Robot** instances for the **Components**. It implements a mechanism to automatically detect that multiple plugins configurations require compatible **Robot** instances. It automatically creates correct **Robot** instances and provides the **Components** with references.

Shared plugin also provides a shared space – a middleware independent communication layer between the **Components** that allows to share the information immediately without relying on the middleware communication mechanism that may introduce delays. To that end, new types of **pipes** for the plugins to communicate the joint-space and **Robot Points** states through the shared space have been developed. These shared **pipes** are created automatically, when the **pipe** publisher is created to communicate with the middleware, i.e., all information broadcasted outside the **shared plugin** is also broadcasted to the shared space. When the subscriber is created for the **Component** inside the **shared plugin**, priority is given to the shared space **pipes**; the middleware **pipes** are generated if no matching shared publisher exists. Other, custom shared **pipes** can be initialized in the **_initCallbacks** method of the

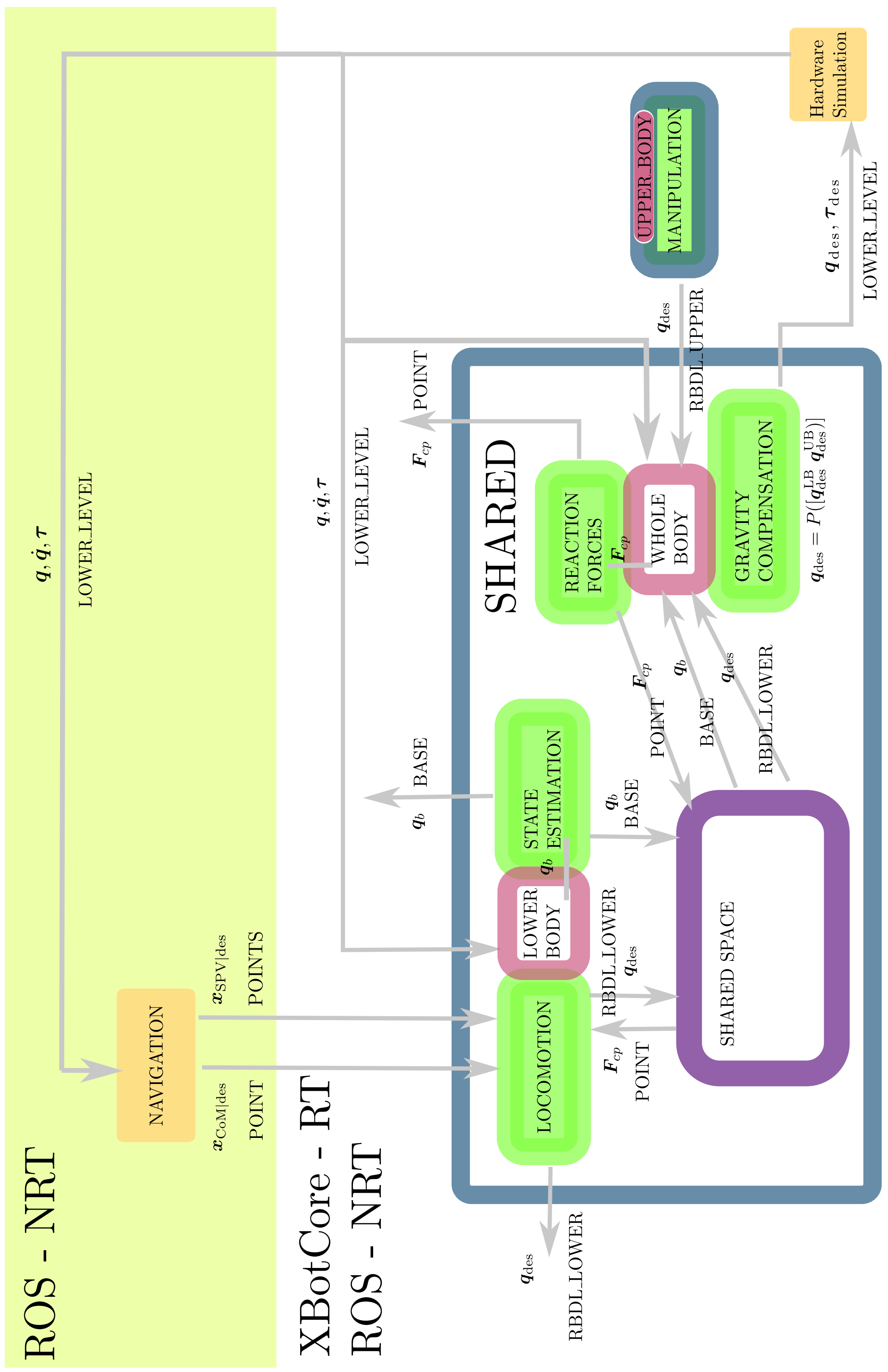


Figure 5.5 An example of the full control scheme implemented with the SOL-FER including cooperation between the RT and NRT layers. Pink areas show loaded **Robot** configuration, blue areas indicate the **Plugin** space, and green areas represent the **Component** space. Yellow areas symbolise the third party elements, and purple area denotes the **shared plugin** shared space.

plugin template. Finally, the **shared plugin** provides a mechanism in XBotCore to load the same **Plugin/Component** with different configurations.

The **shared plugin** configuration does not adhere to the minimum plugin configuration described in Section 5.2.1. The only required tags are 'layer' and a list of plugins to be loaded. Custom plugin configuration may be specified using the double colon as a separator; otherwise, a default configuration is used.[16]. Note that the same plugin can be loaded multiple times simultaneously with different, or even the same, configurations.

Fig. 5.5 shows an example control scheme built from the **shared plugin** cooperating with a SOL-FER **plugin**, a higher-level third party component and a hardware/simulation. In the **shared plugin**, four basic **plugins** cooperate: the state estimation and the whole-body locomotion control use the same lower-body robot model, and the estimation of the reaction forces and the gravity compensation share the full whole-body model.

On **shared plugin** update, the **Robot** lower-body kinematics is updated, and the state estimation **Component** computes the estimated robot floating base position. The state estimation plugin updates the state of the lower-body **Robot** it is attached to. Then, it sends the estimated floating base position to the shared space and to the middleware. The whole-body **Robot** floating base state is read from the shared space, and the kinematic and dynamic models are updated. The next step consists of update of the reaction forces **Component** that estimates and updates the reaction forces in the whole-body **Robot**, and which sends the estimated reaction forces to the shared space and the middleware space. Next, the lower-body **Robot** kinematics is updated again, since the state estimation **Component** modified its current state internally. Moreover, the estimated reaction forces are read from the shared space and updated in the lower-body **Robot**. Then, the locomotion **Component** computes the lower-body joint-space commands to track the desired CoM and SPV positions received from the third party navigation plugin operating in the non-real-time ROS space. The computed joint-space command is sent to the shared-space and the middleware space. Finally, the gravity compensation **Component** is computed. Since, the whole-body **Robot** state has not been changed since the last update, there is no update of the whole-body **Robot** kinematics. The desired lower-body command is read from the shared space, and the desired upper-body command is read from the middleware. The whole-body gravity compensation is computed based on the contact points positions, and whole-body position, velocity and torque commands are send to the lower-level simulation/hardware. The upper-body reference is defined by the external SOL-FER manipulation plugin that operates on the static upper-body

[16]In Appendix B in Code B.15, an example of the **shared plugin** configuration to load 5 different plugins is given

model. In XBotCore, the manipulation plugin is ensured to be updated before the **shared plugin**. Otherwise, since in ROS the order of the plugins updates cannot be secured, the last known desired upper-body joint-space command is used. In the **shared plugin**, order of the **Components** updates is always ensured.

These control scheme has been tested in non-real-time ROS with CENTAURO robot simulation and in real-time XBotCore with CENTAURO robot hardware. These control scheme has been used in the experiments shown in the Chapter 8. In real-time implementation, the higher-level locomotion CoM and SPV commands have been send from the non-real-time ROS layer.

5.3 Summary

In this chapter, Simplifying Operations in Locomotion - Framework for Efficient Research (SOL-FER), a software for the locomotion research has been introduced. This framework has been designed to provide a way to quickly try out ideas by maximizing the flexibility and reconfigurability of the developed modules. An abstraction layers for the robot kinematics and dynamics have been introduced, and the communication layer that consists of the reconfigurable feedbacks/commands has been described. These reconfigurable feedbacks/commands allow, for example, to run multiple cooperating plugins simultaneously or to switch between the online and offline implementations of a module without any source code modification, and thus without a need to recompile the code. Furthermore, the **shared plugin** that decreases the computational burdain of the multiple cooperating plugins by sharing the same robot abstraction layers have been proposed. Finally, the **Robot Points/Handler** structure have been introduced to simplify the implementation of the locomotion control schemes. The proposed framework has been used in the development of the controllers presented in Chapter 7 and Chapter 8 in the initial simulations with the CENTAURO robot (Chapter 4) as well as in experiments presented in Chapter 7 and Chapter 8.

The proposed framework can be extended by automatically sharing the robot dynamic model between plugins. Similar system to the mechanism currently implemented for the robot hardware/kinematics layer could be used. Furthermore, the update management mechanism implemented in the dynamic model could be used throughout the framework to decrease the computational burden of implemented modules. An algorithm to detect that different robot configurations are compatible to share the same kinematic model would further decrease the computational load of the multiple cooperating plugins. Finally, a plugin execution scheduler to support modules operating on different frequencies would greatly enhance the framework capabilities.

Chapter 6

Model of a Support Polygon Vertex in the Hybrid Legged-Wheeled Robots

As reported in [80], a first-order kinematics scheme with steering reference based on the ICR alone causes the support polygon of some of the hybrid legged-wheeled robots to diverge. In this chapter, a new kinematic model for the SPV of a general HLW robot is developed and analysed to investiagte this behaviour.[1] To that end, the torus wheel geometry rather than standard cylindrical and spherical geometries is considered to describe the support polygon of a legged-wheeled robot. Then, its first and second-order derivatives are computed under the wheel rolling assumption. Furthermore, this new model is applied to a general legged-wheeled robot as well as a standard mobile platform, and a non-holonomic constraint for legged-wheeled robots is discussed with respect to standard non-holonomic wheeled mobile robots.

The chapter is organized as follows: First, torus wheel model is introduced in Section 6.1, and the SPV velocity is computed in Section 6.2. Obtained result is discussed and analysed in Section 6.3, and the developed model is computed at the acceleration level in Section 6.4. Finally, a conclusion is given in Section 6.5

[1]The work included in this chapter has been presented at the IROS 2018 conference with the publication titled "On the kinematics of wheeled motion control of a hybrid wheeled-legged centauro robot" by Malgorzata Kamedula, Navvab Kashiri, and Nikos G. Tsagarakis. An extension of this publication has been submitted to Robotics and Autonomous Systems with the submission titled “Wheeled Motion Kinematics and Control of a Hybrid Mobility CENTAURO robot” by the same authors which is currently under revision. The second-order kinematics model has been published in the IEEE Robotics and Automation Letters article titled “Reactive Support Polygon Adaptation for the Hybrid Legged-Wheeled CENTAURO robot” by Malgorzata Kamedula and Nikos G. Tsagarakis.

6.1 Wheel-ground contact model

In this section, a wheel geometry is discussed, and the ground contact model is studied, so that an expression for the SPV in terms of the robot state and the ground normal is derived.

A wheel-ground contact is a complex function of the geometry and elastic properties of the wheel as well as the ground [112]. To discuss the system kinematics, in this work, a simplified wheel-ground contact model is considered in which the contact area is reduced to a single point that depends only on the ground and wheel geometries. The former is an extrinsic parameter and no specific properties can be presumed; however, for the sake of simplicity, it is assumed that the ground normal is constant in the proximity of a contact point. The latter, as an intrinsic characteristic, can be modelled with more details.

A common simplification is to consider the wheel as a sphere or a cylinder that provides a sufficient approximation for systems with fixed camber angle. However, the cylindrical geometry does not allow to model a non-zero camber angle without taking into account deformations, while the spherical geometry implies the wheel centre and the contact point are collinear along the ground normal. It is shown at the Fig. 6.1, on the left, where blue line represents the torus and green line a sphere model. In this work, we propose to model the wheel as a torus to more closely represent the kinematics of the ground-wheel contact for a non-zero camber angle. Particularly, the proposed model is compatible with the geometry of the wheel used in the CENTAURO robot shown at Fig. 6.1. The top photograph of the Fig. 6.2 shows the CENTAURO robot in a configuration with the zero camber angles, and so with the wheels perpendicular to the ground. The bottom photograph on the Fig. 6.2 presents the CENTAURO robot adopting the configuration with the non-zero camber angles, and so with the wheels at an angle with respect to the ground surface. Even though the zero camber

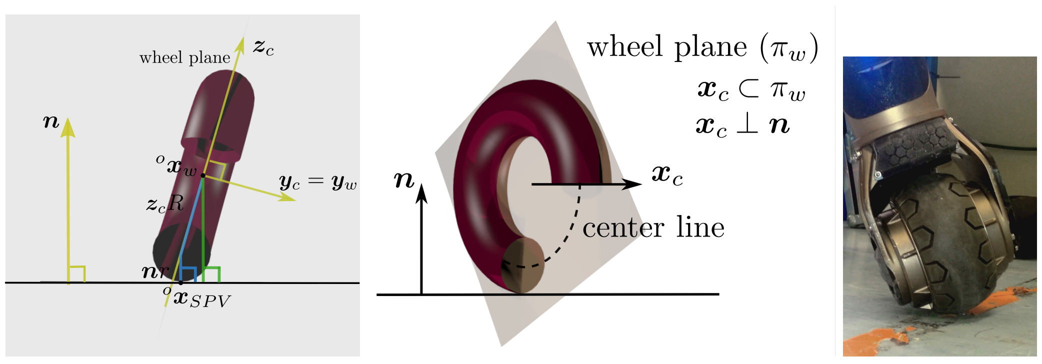


Figure 6.1 On the left: A contact point placement in sphere and torus models for a non-zero camber angle; in the middle: Torus model with wheel plane, wheel axis, and ground-wheel frame axes; on the right: Photo of the CENTAURO robot wheel.

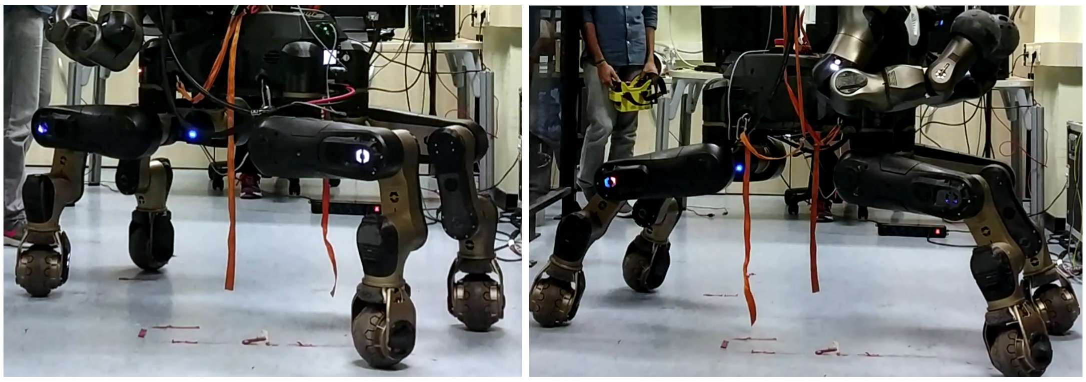


Figure 6.2 On the left: CENTAURO robot with all legs in the zero camber angle position. On the right: CENTAURO robot with all legs in the non-zero camber angle configuration.

angle assumption is common in the modelling of the hybrid legged-wheeled platforms, when this motion is permissible by the robot mechanical design, maintaining the zero camber angle throughout the motion cannot be safely ensured on uneven terrains. When the robot adopts a non-zero camber angle position, the motion controllers developed with the zero-camber angle assumption cannot ensure the stable motion. Furthermore, consideration of the non-zero camber angle position allows the robot to extend its leg workspace with comparison to the standard zero camber angle controllers.

Let us define the 'wheel plane' that is constructed upon the torus centre line, i.e. it divides the wheel in half with the plane normal being parallel to the wheel axis ($\boldsymbol{y}_w \in \Re^3$). Then the position vector from the wheel centre to the SPV (${}^w\boldsymbol{x}_{SPV} \in \Re^3$) is, as shown in Fig. 6.1,

$$
{}^w\boldsymbol{x}_{SPV} = -\boldsymbol{z}_c R - \boldsymbol{n} r, \tag{6.1}
$$

where $R \in \Re^+$ represents the torus major radius, $r \in \Re^+$ stands for a torus minor radius, $\boldsymbol{n} \in \Re^3$ symbolises the unit vector along the ground normal, and $\boldsymbol{z}_c \in \Re^3$ denotes a unit vector that lies in the cross-section of a wheel plain and a plane constructed with the wheel axis and ground normal.

The vector $\boldsymbol{z}_c$ in (6.1) is not fixed in any of the robot frames. It is convenient to define a new frame at the wheel centre, referred as wheel-ground frame and marked by $\mathscr{F}_c$ in this work (Fig. 6.3). Its y-axis ($\boldsymbol{y}_c \in \Re^3$) coincides with wheel axis ($\boldsymbol{y}_c \triangleq \boldsymbol{y}_w$), and the x-axis ($\boldsymbol{x}_c \in \Re^3$) is perpendicular to the ground normal

$$
\boldsymbol{x}_c \triangleq \frac{\boldsymbol{y}_c \times \boldsymbol{n}}{|\boldsymbol{y}_c \times \boldsymbol{n}|}. \tag{6.2}
$$

The z-axis is defined by a right-hand-side rule, and it coincides with the vector $\boldsymbol{z}_c$ in (6.1),

$$\boldsymbol{z}_c \triangleq \frac{\boldsymbol{x}_c \times \boldsymbol{y}_c}{|\boldsymbol{x}_c \times \boldsymbol{y}_c|}. \tag{6.3}$$

Equations (6.2) and (6.3) can be expressed, as functions of robot state ($\boldsymbol{q}$) and ground normal ($\boldsymbol{n}$). Using linear algebra operations, it can be shown that

$$\boldsymbol{x}_c(\boldsymbol{q},\boldsymbol{n}) = \widetilde{\boldsymbol{y}}_w(\boldsymbol{q})\boldsymbol{n}\frac{1}{||\widetilde{\boldsymbol{y}}_w(\boldsymbol{q})\boldsymbol{n}||},$$

$$\boldsymbol{z}_c(\boldsymbol{q},\boldsymbol{n}) = (\boldsymbol{n} - \boldsymbol{\alpha}(\boldsymbol{q},\boldsymbol{n}))\chi(\boldsymbol{q},\boldsymbol{n}), \tag{6.4}$$

where the tilde symbol $\widetilde{(.)}$ represents a vector $(.)$ skew matrix, and the auxiliary variables $\boldsymbol{\alpha} \in \Re^3$ and $\chi \in \Re^1$ read

$$\boldsymbol{\alpha}(\boldsymbol{q},\boldsymbol{n}) = \boldsymbol{y}_w(\boldsymbol{q})\boldsymbol{n}^T\boldsymbol{y}_w(\boldsymbol{q}), \tag{6.5}$$

$$\chi(\boldsymbol{q},\boldsymbol{n}) = \frac{1}{||\boldsymbol{n} - \boldsymbol{\alpha}(\boldsymbol{q},\boldsymbol{n})||}. \tag{6.6}$$

6.2 Derivative of the SPV

A SPV position, expressed in (6.1), describes the robot characteristic that is not directly associated to any point of the robot body. On the other hand, NSPR constraint (3.5) applies a condition on the wheel point that is in contact with the ground, i.e. the point rigidly attached to the robot body. At a given time ($t \in \Re^1$) the vector from the world origin to the constraint end-effector point – ${}^o\boldsymbol{x}_{cp}$ – coincides with the vector from the world origin to the SPV – ${}^o\boldsymbol{x}_{SPV} \in \Re^3$ –

$${}^o\boldsymbol{x}_{SPV}|_t = {}^o\boldsymbol{x}_{cp}|_t. \tag{6.7}$$

However, their time derivatives differ (${}^o\dot{\boldsymbol{x}}_{SPV}|_t \neq {}^o\dot{\boldsymbol{x}}_{cp}|_t$). To examine a set of feasible SPV motions under the NSPR condition, the time derivatives of ${}^o\boldsymbol{x}_{SPV}$ and ${}^o\boldsymbol{x}_{cp}$ are computed next.

6.2.1 Non-sliding Pure Rolling Constraint

First, let us discuss the NSPR condition (3.5) that constrains velocity of the point on the wheel in a ground contact relative to the ground. Using rigid body transformation, one can show (3.2)

$${}^o\boldsymbol{x}_{cp} = {}^o\boldsymbol{x}_r + {}^r\boldsymbol{x}_w + {}^w\boldsymbol{x}_{cp}, \tag{6.8}$$

where ${}^{o}\boldsymbol{x}_r \in \Re^3$ stands for the vector from the world origin to the robot reference point and defines the world posture of the robot, ${}^{r}\boldsymbol{x}_w \in \Re^3$ symbolises the vector from the robot reference point to the wheel centre that describes internal state of the robot, and ${}^{w}\boldsymbol{x}_{cp} \in \Re^3$ denotes the vector from the wheel centre to the wheel point in contact with the ground. Vectors ${}^{o}\boldsymbol{x}_r$ and ${}^{r}\boldsymbol{x}_w$ and corresponding derivatives are commonly evaluated variables in robot kinematic analysis that can be computed with a standard rigid body kinematics. Time derivative of (6.8) reads (3.3)

$$ {}^{o}\dot{\boldsymbol{x}}_{cp} = {}^{o}\dot{\boldsymbol{x}}_r + {}^{r}\dot{\boldsymbol{x}}_w + {}^{w}\dot{\boldsymbol{x}}_{cp}. \tag{6.9} $$

The vector ${}^{w}\boldsymbol{x}_{cp}$ belongs to a robot wheel – a rigid body – and therefore its derivative is defined as (3.5)

$$ {}^{w}\dot{\boldsymbol{x}}_{cp} = \boldsymbol{\omega}_w \times {}^{w}\boldsymbol{x}_{cp}, \tag{6.10} $$

where $\boldsymbol{\omega}_w \in \Re^3$ symbolizes wheel angular velocity relative to the reference frame. (6.10) expressed by a skew-matrix operator reads

$$ {}^{w}\dot{\boldsymbol{x}}_{cp} = -{}^{w}\tilde{\boldsymbol{x}}_{cp}\boldsymbol{\omega}_w. \tag{6.11} $$

Furthermore, wheel angular velocity can be written in a general first-order kinematic form

$$ \boldsymbol{\omega}_w = \mathbf{J}_w\dot{\boldsymbol{q}} + \boldsymbol{\omega}_m, \tag{6.12} $$

where $\mathbf{J}_w \in \Re^{3\times n}$ denotes the wheel orientation Jacobian that is a known variable within a standard kinematic model, and $\boldsymbol{\omega}_m \in \Re^3$ represents all potential reference frame angular velocity components dependent on external factors that are not modelled in this work such as ground curvature or external non-inertial reference frame.

SPV derivative

Now, consider the vector from the world origin to the SPV defined as

$$ {}^{o}\boldsymbol{x}_{SPV} = {}^{o}\boldsymbol{x}_r + {}^{r}\boldsymbol{x}_w + {}^{w}\boldsymbol{x}_{SPV}, \tag{6.13} $$

where ${}^{w}\boldsymbol{x}_{SPV}$ represents the vector from the wheel centre to the SPV as described by (6.1). The time derivative of (6.13) reads

$$ {}^{o}\dot{\boldsymbol{x}}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_r + {}^{r}\dot{\boldsymbol{x}}_w + {}^{w}\dot{\boldsymbol{x}}_{SPV}, \tag{6.14} $$

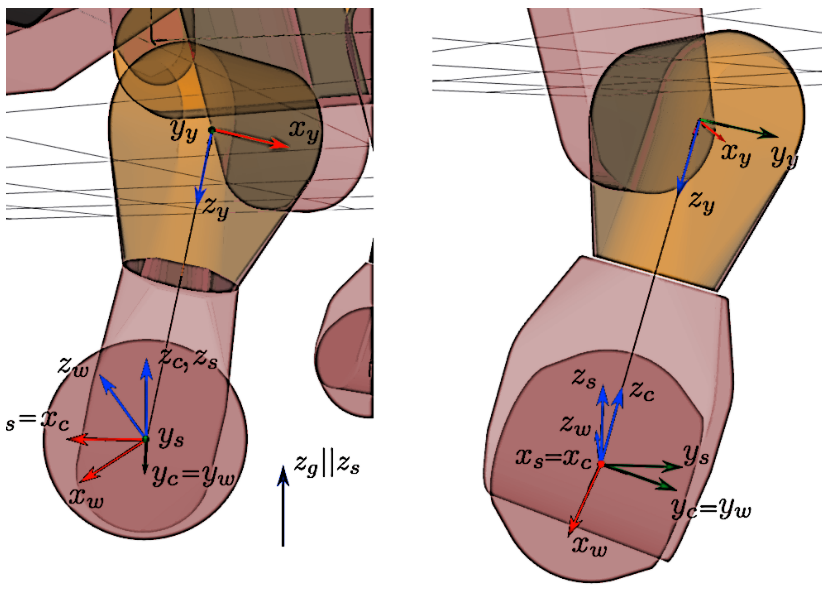


Figure 6.3 The wheel-ground and steering frames in the legged-wheeled structure. On the left: the view aligned with the steering frame y-axis, on the right: the view aligned with the steering frame x-axis.

where only ${}^{w}\dot{\boldsymbol{x}}_{SPV}$ remains unknown. From (6.1) it reads

$$ {}^{w}\dot{\boldsymbol{x}}_{SPV} = -\dot{\boldsymbol{z}}_c R - \dot{\boldsymbol{n}} r. \tag{6.15}$$

The derivative of the ground normal depends on the ground properties. We assume a flat ground in this thesis, and thus the derivative depends only on the reference frame angular velocity ($\boldsymbol{\omega}_n \in \Re^3$)

$$\dot{\boldsymbol{n}} = \boldsymbol{\omega}_n \times \boldsymbol{n}.$$

It can be further decomposed as follows

$$\dot{\boldsymbol{n}} = (\mathbf{J}_n \dot{\boldsymbol{q}} + \boldsymbol{\omega}_m) \times \boldsymbol{n}, \tag{6.16}$$

where $\mathbf{J}_n \in \Re^{3\times n}$ represents the rotation Jacobian from the reference to the inertial frame.

The remaining term of (6.15) – the derivative of the wheel-ground frame z-axis (6.4) – reads

$$\dot{\boldsymbol{z}}_c = (\boldsymbol{n} - \boldsymbol{\alpha})\,\dot{\chi} + (\dot{\boldsymbol{n}} - \dot{\boldsymbol{\alpha}})\,\chi. \tag{6.17}$$

From (6.5) and (6.6) one can see that $\boldsymbol{z}_c$ depends on the ground normal and wheel axis – $\boldsymbol{y}_w$ – that by definition is constant in the wheel reference frame. The wheel axis time derivative reads

$$\dot{\boldsymbol{y}}_w = \boldsymbol{\omega}_w \times \boldsymbol{y}_w = -\widetilde{\boldsymbol{y}}_w \boldsymbol{\omega}_w. \tag{6.18}$$

Then, taking into account (6.16) and (6.18), it can be shown that derivative of (6.5) reads

$$\dot{\boldsymbol{\alpha}}(\boldsymbol{q},\boldsymbol{n}) = -\boldsymbol{\gamma}(\boldsymbol{q},\boldsymbol{n})\boldsymbol{\omega}_w + \boldsymbol{y}_w \boldsymbol{y}_w^T \dot{\boldsymbol{n}}, \tag{6.19}$$

where

$$\boldsymbol{\gamma}(\boldsymbol{q},\boldsymbol{n}) = \widetilde{\boldsymbol{\gamma}}_s + \boldsymbol{y}_w \boldsymbol{n}^T \widetilde{\boldsymbol{y}}_w,$$

$$\boldsymbol{\gamma}_s(\boldsymbol{q},\boldsymbol{n}) = \boldsymbol{y}_w \boldsymbol{n}^T \boldsymbol{y}_w.$$

A remaining term in (6.17) – $\dot{\chi}$ – represents a derivative of a cross product normalisation used in (6.3) to ensure a unit length of a frame axis. To compute $\dot{\chi}$ an euclidean norm in (6.6) is expressed as a vector dot product that reads

$$\chi(\boldsymbol{q},\boldsymbol{n}) = \frac{1}{\sqrt{(\boldsymbol{n}-\boldsymbol{\alpha}(\boldsymbol{q},\boldsymbol{n}))^T(\boldsymbol{n}-\boldsymbol{\alpha}(\boldsymbol{q},\boldsymbol{n}))}}.$$

Recall that the ground normal and the wheel axis have been defined as unit vectors, and thus $||\boldsymbol{n}|| = ||\boldsymbol{y}_w|| = 1$. The above equation simplifies to

$$\chi(\boldsymbol{q},\boldsymbol{n}) = \frac{1}{\sqrt{1-\boldsymbol{\alpha}^T\boldsymbol{n}}}, \tag{6.20}$$

and its derivative reads

$$\dot{\chi}(\boldsymbol{q},\boldsymbol{n}) = -0.5\chi^3 \frac{d}{dt}(1-\boldsymbol{\alpha}^T\boldsymbol{n}).$$

With the derivative of the wheel axis computed as in (6.18), the above equation results in

$$\dot{\chi}(\boldsymbol{q},\boldsymbol{n}) = -0.5\chi^3 \boldsymbol{n}^T \boldsymbol{\gamma}\boldsymbol{\omega}_w + \chi^3 \boldsymbol{\alpha}^T \dot{\boldsymbol{n}}. \tag{6.21}$$

Finally, from (6.17), taking into account (6.4), (6.19) and (6.21), the formula of (6.15) in terms of the robot state – $\boldsymbol{y}_w(\boldsymbol{q})$, $\mathbf{J}_{\theta_w}(\boldsymbol{q})$, $\boldsymbol{\omega}_w(\dot{\boldsymbol{q}})$ – and the ground normal – $\boldsymbol{n}$, $\dot{\boldsymbol{n}}$ – takes form

$${}^w\dot{\boldsymbol{x}}_{SPV} = (\boldsymbol{\zeta} - \boldsymbol{I})\,\boldsymbol{\gamma}\boldsymbol{\omega}_w \chi R - (\boldsymbol{\psi} R + \boldsymbol{I} r)\dot{\boldsymbol{n}}, \tag{6.22}$$

where

$$\boldsymbol{\zeta} = 0.5\chi^2(\boldsymbol{n}-\boldsymbol{\alpha})\boldsymbol{n}^T,$$

$$\boldsymbol{\psi} = (\boldsymbol{I} - \boldsymbol{y}_w \boldsymbol{y}_w^T)(\boldsymbol{I} + \boldsymbol{n}\boldsymbol{\alpha}^T \chi^3),$$

and $\boldsymbol{I} \in \Re^{3\times 3}$ stands for an identity matrix.

6.2.2 Constraint SPV

In stable wheeled motion, the NSPR condition is ensured and only a limited subset of possible solutions for (6.14) is of an interest for the motion scheme design. To determine a subspace of feasible motions, the derivative of a SPV is resolved under the NSPR condition. (6.22) from (3.5), (6.9) and (6.14) reads

$$^{o}\dot{\boldsymbol{x}}_{SPV} = {}^{w}\dot{\boldsymbol{x}}_{SPV} - {}^{w}\dot{\boldsymbol{x}}_{cp}. \tag{6.23}$$

Given that from (6.7), (6.8) and (6.13) one gets

$$^{w}\boldsymbol{x}_{cp|t} = {}^{w}\boldsymbol{x}_{SPV|t}, \tag{6.24}$$

(6.11) at any given time reads

$$^{w}\dot{\boldsymbol{x}}_{cp|t} = -{}^{w}\widetilde{\boldsymbol{x}}_{SPV}\boldsymbol{\omega}_w. \tag{6.25}$$

As a result, when the NSPR condition is satisfied, under (6.22), (6.23) and (6.25), (6.14) can be written as

$$\begin{aligned} ^{o}\dot{\boldsymbol{x}}_{SPV|t} = \boldsymbol{\zeta}\boldsymbol{\gamma}(\boldsymbol{q},\boldsymbol{n})\boldsymbol{\omega}_w \chi R - \widetilde{\boldsymbol{n}}\boldsymbol{\omega}_w(\chi R + r) \\ - \boldsymbol{y}_w \boldsymbol{n}^T \widetilde{\boldsymbol{y}}_w \boldsymbol{\omega}_w \chi R - (\boldsymbol{\psi} R + \boldsymbol{I} r)\dot{\boldsymbol{n}} \end{aligned} \tag{6.26}$$

that depends solely on the robot state, the ground normal and $\boldsymbol{\omega}_m$ that gathers unmodelled extrinsic factors.

Note that the (6.26) gives the value of the first-order SPV derivative under given assumption and not an expression for the SPV function. If one wish to compute higher-order derivatives of SPV, (6.22) and (6.14) should be considered directly.

6.3 Result Analysis

The expression in (6.26) describes an instantaneous SPV motion under the NSPR condition. To further discuss the obtained result, (6.26) is expressed with respect to the ground frame, noted with $\mathscr{F}_n$ in this work[2]. It is an inertial frame, thus $\mathbf{J}_n = 0$, and – for a flat ground –

[2]Reacall that the variables expressed in a specific frame are indicated by a superscript consisting of the frame symbol in a parentheses, e.g. $x^{(n)}$.

$\boldsymbol{\omega}_m = 0$, $\dot{\boldsymbol{n}} = \mathbf{0}$. The ground frame is defined so that its z-axis, $\boldsymbol{z}_n$, coincide with the ground normal, while the corresponding x-axis, $\boldsymbol{x}_n$, is an arbitrary reference vector that lies on the ground plane. Then, the ground normal is a constant value that reads $\boldsymbol{n}^{(n)} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, the wheel axis is $\boldsymbol{y}_w^{(n)} = \begin{bmatrix} y_x^{(n)} & y_y^{(n)} & y_z^{(n)} \end{bmatrix}^T$, and (6.20) becomes

$$\chi^{(n)} = \frac{1}{\sqrt{1 - y_z^{2(n)}}}.$$

Equation (6.26) in the ground frame reads

$$^{o}\dot{\boldsymbol{x}}_{SPV|t}^{(n)} = \boldsymbol{A}^{(n)} \mathbf{J}_w^{(n)} \dot{\boldsymbol{q}} \tag{6.27}$$

where[3]

$$\boldsymbol{A}^{(n)} = \begin{bmatrix} -y_x y_y \chi^3 R & (1 + y_x^2 \chi^2)\chi R + r & 0 \\ -(1 + y_y^2 \chi^2)\chi R - r & y_x y_y \chi^3 R & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Equation (6.27) implies that SPV cannot move along the ground normal if the NSPR assumption is satisfied as that would correspond to raising a leg above or penetrating the ground. Thus a subspace of feasible motions is reduced from a full 3-D space to a two dimensional subspace of the ground plane. This constraint can be also expressed in a frame independent manner

$$^{o}\dot{\boldsymbol{x}}_{SPV}^{T} \boldsymbol{n} = 0. \tag{6.28}$$

(6.27) also shows that the wheel rotation around the ground normal is an internal motion that has no direct influence on the SPV.

Furthermore, to gain more insight into the contact point motion in terms of rolling, steering and camber angles that are often used to describe a wheel orientation, a new frame is introduced, called steering frame and marked by $\mathscr{F}_s$ in this work (Fig. 6.3). It is defined in a way that its z-axis ($\boldsymbol{z}_s \in \Re^3$) coincides with $\boldsymbol{z}_n$, and the x-axis ($\boldsymbol{x}_s \in \Re^3$) coincides with $\boldsymbol{x}_c$. The ground normal is constant and equal to $\boldsymbol{n}^{(s)} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, what on a flat ground

[3]The equation is expressed in the ground frame, and so all the variables. The frame superscripts have been neglected for readability.

implies $\boldsymbol{\omega}_n || \boldsymbol{n}$; $\dot{\boldsymbol{n}} = \boldsymbol{\omega}_n \times \boldsymbol{n} = 0$ therefore holds. The wheel axis becomes

$$\boldsymbol{y}_w^{(s)} = \begin{bmatrix} 0 \\ y_y^{(s)} \\ y_z^{(s)} \end{bmatrix}, \ \chi^{(s)} = \frac{1}{\sqrt{1 - y_z^{2(s)}}},$$

and (6.26) reads

$$^{o}\dot{\boldsymbol{x}}_{SPV|t}^{(s)} = \begin{bmatrix} 0 & \chi^{(s)}R + r & 0 \\ -r & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\nu} \\ \dot{\beta} \end{bmatrix} \tag{6.29}$$

with

$$\boldsymbol{\omega}_w^{(s)} = \mathbf{J}_w^{(s)} \dot{\boldsymbol{q}} = \begin{bmatrix} \dot{\varphi} \\ \dot{\nu} \\ \dot{\beta} \end{bmatrix}.$$

The x-coordinate of the wheel angular velocity ($\varphi \in \Re^1$) signifies change in the wheel camber angle, the z-coordinate ($\beta \in \Re^1$) indicates rotation around the ground normal, and y-coordinate ($\nu \in \Re^1$) directly maps to the wheel rolling motion. The term $\chi R + r$ represents the wheel effective radius, i.e the radius the wheel is rolling around given current camber angle.

Note that the z-axes of both the ground frame and the steering frame are parallel to the ground normal, and thus the rotation around the ground normal by β describes the transformation from the ground frame to the steering frame.

Equation (6.29) decouples impact of the wheel angular velocity components on the SPV motion. In the real system, the range of camber angle motion is limited with the wheel geometry and joint limits while the wheel camber angle defines the wheel grip on the ground. In practical applications, the camber angle value is designed to satisfy these condition and is not available for SPV tracking. As a result, regardless the system kinematics, maximum one dimension remains to track a contact point on a 2D ground plane. With constrained camber angle ($\varphi = \text{const}$) the system in (6.29) – and so in (6.26) – is subject to a non-holonomic constraint.

Note, that in derivation of (6.26) only a wheel geometry has been assumed and no assumptions on the system kinematics has been made. As a result, (6.26) describes the support polygon in a general legged-wheeled robot for which the wheel model holds, regardless a specific wheel assembly. The system kinematics in (6.26) is expressed by $\boldsymbol{\omega}_w$ which Jacobian structure may limit the support polygon motion.

6.3.1 Non-holonomy of legged-wheeled and all-steerable mobile robots

In this section, the non-holonomy in (6.29) for the system with defined camber angle and the SPVs tracking is laid out. It is shown, that for the planar base motion, (6.29) takes the same form as kinematics equation of a standard non-holonomic mobile robot with only steerable wheels.[4]

As mention previously in this section, β describes a rotation from ground to steering frame. This transformation reads

$$ {}^{o}\dot{\boldsymbol{x}}^{(n)}_{SPV|t} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{o}\dot{\boldsymbol{x}}^{(s)}_{SPV|t}. \tag{6.30} $$

To compare a legged-wheeled robot motion with a motion of standard wheeled mobile robots, consider a simple case when the desired base velocity is composed of the robot base planar coordinates, and the robot moves with a constant support polygon, i.e wheel contact points and the robot base move as a rigid body, it reads

$$ {}^{o}\dot{\boldsymbol{x}}^{(n)}_{SPV,\text{des}} = \begin{bmatrix} {}^{o}\dot{x}_{SPV,\text{des}} \\ {}^{o}\dot{y}_{SPV,\text{des}} \\ {}^{o}\dot{z}_{SPV,\text{des}} \end{bmatrix} = \begin{bmatrix} {}^{o}\dot{x}_{b,\text{des}} \\ {}^{o}\dot{y}_{b,\text{des}} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega_{z,\text{des}} \end{bmatrix} \times \begin{bmatrix} {}^{b}x_{SPV,\text{des}} \\ {}^{b}y_{SPV,\text{des}} \\ {}^{b}z_{SPV,\text{des}} \end{bmatrix}, \tag{6.31} $$

where $\dot{x}_{b,\text{des}}, \dot{y}_{b,\text{des}} \in \Re^1$ represent the vector coordinates of the desired base linear velocity, $\omega_{z,\text{des}} \in \Re^1$ stands for the desired angular base velocity around the ground normal and ${}^{b}x_{SPV,\text{des}}, {}^{b}y_{SPV,\text{des}}, {}^{b}z_{SPV,\text{des}} \in \Re^1$ symbolise the coordinates of the position vector from the robot base to the SPV. In this case, taking into account (6.30) and (6.31), (6.29) takes, for

[4]This type of mobile robots is refereed to as 'all-steerable' platform in this work.

each leg, a form

$$\begin{bmatrix} \cos(\underline{\beta}) & \sin(\underline{\beta}) \\ -\sin(\underline{\beta}) & \cos(\underline{\beta}) \end{bmatrix} \begin{bmatrix} {}^{o}\dot{x}_{SPV,\text{des}} \\ {}^{o}\dot{x}_{SPV,\text{des}} \end{bmatrix} = \begin{bmatrix} R_{\chi}\underline{\dot{v}} \\ -r\dot{\varphi} \end{bmatrix}, \tag{6.32}$$

where underline $\underline{(.)}$ highlights the unknown variables and $R_{\chi} = \chi^{(s)}R + r$. With the camber angle motion – $\dot{\varphi}$ – defined, only one variable remains – $\dot{v}$ – and SPV can be tracked along the wheel rolling direction only. However, the solution exists if β takes a value that the desired SPV motion lies along the wheel rolling motion. According to (6.29), $\dot{\beta}$ is an internal DoF that does not directly influence the SPV.

For $\varphi = 0$, (6.31) and (6.32) accurately represent the well-known kinematics of the all-steerable non-holonomic platforms as given in [72, 74, 75]. For these robots, violation of the non-holonomic constraint prohibits the wheels motion without sliding. However, for hybrid legged-wheeled platforms the relative motion between the robot base and SPVs (${}^{b}\boldsymbol{x}_{SPV} \in \Re^3$) is possible. For the CENTAURO platform with 6-DoF legs any relative motion is feasible, and the SPV velocity (6.31) for a system following the planar base motion expands to

$${}^{o}\dot{\boldsymbol{x}}^{(n)}_{SPV,LW} = \begin{bmatrix} {}^{o}\dot{x}_{SPV,\text{des}} \\ {}^{o}\dot{y}_{SPV,\text{des}} \\ {}^{o}\dot{z}_{SPV,\text{des}} \end{bmatrix} + \begin{bmatrix} {}^{b}\dot{x}_{SPV} \\ {}^{b}\dot{y}_{SPV} \\ {}^{b}\dot{z}_{SPV} \end{bmatrix}, \tag{6.33}$$

where ${}^{o}\dot{\boldsymbol{x}}_{SPV,LW} \in \Re^3$ represents the velocity of the SPV in the HLW robots, and (6.30) reads

$$\begin{bmatrix} \cos(\underline{\beta}) & \sin(\underline{\beta}) \\ -\sin(\underline{\beta}) & \cos(\underline{\beta}) \end{bmatrix} \begin{bmatrix} {}^{o}\dot{x}_{SPV,\text{des}} + \underline{{}^{b}\dot{x}_{SPV}} \\ {}^{o}\dot{y}_{SPV,\text{des}} + \underline{{}^{b}\dot{y}_{SPV}} \end{bmatrix} = \begin{bmatrix} R_{\chi}\underline{\dot{v}} \\ -r\dot{\varphi} \end{bmatrix}. \tag{6.34}$$

In contrary to standard platforms, the solution to (6.34) exists – the desired base motion is achievable with a relative motion between the robot base and the support polygon. This indicates the system is holonomic for omnidirectional driving unless a desired support polygon velocity is imposed (${}^{b}\boldsymbol{x}_{SPV} = {}^{b}\boldsymbol{x}_{SPV,\text{des}}$) together with the camber angle ($\dot{\boldsymbol{\varphi}} = \dot{\boldsymbol{\varphi}}_{\text{des}}$). In such situations, (6.34) can be expressed as (6.32) with a constant offset. Thus (6.34) is subject to the same non-holonomic constraint, when both the support polygon and camber angles are tracked.

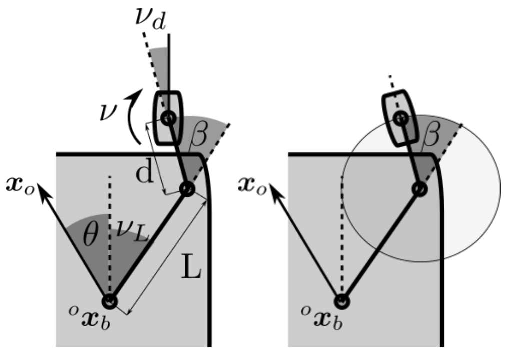


Figure 6.4 On the left – schematic of the conventional wheel assembly, on the right – trace of the caster wheel contact point with respect to the robot base.

Table 6.1 Support polygon evolution of the standard wheeled platforms.

Wheel Type	Fixed	Steerable	Caster
Parameters (Fig. 6.4)	$\beta = 0$ $d = 0$	$\nu_d = 0$ $d = 0$	$\nu_d = 0$
Auxiliary Variables	$\nu_F = \nu_L + \nu_d + \theta_{\vert n}$ $k = \sin(\nu_F)\dot{x}_b - \cos(\nu_F)\dot{y}_b$	$\nu_S = \nu_L + \beta + \theta_{\vert n}$ $k = \sin(\nu_S)\dot{x}_b - \cos(\nu_S)\dot{y}_b$	$\nu_C = \nu_L + \beta + \theta_{\vert n}$ $k = \sin(\nu_C)\dot{x}_b - \cos(\nu_C)\dot{y}_b$
NSPR condition[1] $\dot{\nu} =$	$\frac{L\sin(\nu_d)\dot{\theta}_{\vert n} + \sin(\nu_F)\dot{y}_b + \cos(\nu_F)\dot{x}_b}{R+r}$	$\frac{L\sin(\beta)\dot{\theta}_{\vert n} + \sin(\nu_S)\dot{y}_b + \cos(\nu_S)\dot{x}_b}{R+r}$	$\frac{L\sin(\beta)\dot{\theta}_{\vert n} + \sin(\nu_C)\dot{y}_b + \cos(\nu_C)\dot{x}_b}{R+r}$
$k =$	$L\cos(\nu_d)\dot{\theta}_{\vert n}$	$L\cos(\beta)\dot{\theta}_{\vert n}$	$L\cos(\beta)\dot{\theta}_{\vert n} + d\left(\dot{\theta}_{\vert n} + \dot{\beta}\right)$
Support Polygon (6.36) ${}^{b}\dot{\boldsymbol{x}}_{SPV}^{(b)}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -d\sin(\nu_L + \beta) \\ d\cos(\nu_L + \beta) \\ 0 \end{bmatrix}$

[1] The unilateral condition for all standard mobile platforms imposes $\dot{z}_b = 0$.

6.3.2 Support polygon analysis for standard wheeled mobile robots

In this section, (6.29) is applied to analyse the evolution of the support polygon for well-studied conventional wheel assemblies: fixed, steering and caster wheels. To this end, the kinematics of the generalized standard wheel assembly, as given at Fig. 6.4, has been considered first; (6.29) results to

$$ {}^{o}\dot{\boldsymbol{x}}^{(s)}_{SPV|t} = \begin{bmatrix} (R+r)\dot{v} \\ 0 \\ 0 \end{bmatrix}. $$

In the next step, the feasible values of $\dot{v}$ are computed from the NSPR constraint for each wheel type and applied to (6.35). Finally, to achieve more clear-cut result, the robot world posture is eliminated from (6.35) by moving the vector origin to the robot base with a rigid-body transformation that reads

$$ {}^{b}\dot{\boldsymbol{x}}^{(b)}_{SPV} = {}^{b}\boldsymbol{R}_{s}\left({}^{o}\dot{\boldsymbol{x}}_{SPV} - {}^{o}\dot{\boldsymbol{x}}_{b}\right)^{(s)} - \left(\boldsymbol{\omega}_{b} \times {}^{o}\boldsymbol{x}_{b}\right)^{(b)}, \tag{6.36} $$

where ${}^{o}\boldsymbol{x}_{b}$ represents the current base position, $\boldsymbol{\omega}_{b} \in \Re^{3}$ denotes the angular velocity of the robot base with respect to the inertial frame, and ${}^{b}\boldsymbol{R}_{s} \in \Re^{3\times3}$ expresses the rotation matrix from the steering frame to the base reference frame.

Table 6.1 provides a summary of the above analysis. It shows that the contact point placement is locked for the fixed and steering wheels, while the mobility of the castor wheel imposes the support polygon adjustment on a circular path around the wheel steering axis whose radius equals to the wheel offset (Fig. 6.4). As follows from (6.29) the holonomic omnidirectional driving scheme can be designed for the wheeled platform provided that its support polygon is not constrained and/or the camber angle is not imposed. In case of standard wheeled platforms, that applies only to the systems with only castor wheels, whose motion imposes support polygon change. Other standard platforms, with fixed and steering wheels, enforce a constant support polygon where the former excludes omnidirectional driving, and the later introduces non-holonomy [68].

The analysis provided in this section shows that the proposed SPV model can be applied to any wheeled and legged-wheeled robot, and verifies that the developed model yelds the excpected results for the simple platforms.

6.4 Acceleration of the Support Polygon Vertex

In this section, the SPV model developed in Section 6.2 has been extended to the second-order kinematics, and the linear model with the mixed acceleration/velocity state is designed.

To that end, the equation on the SPV velocity is recalled. For simplicity, the inertial frame and flat ground are assumed, i.e., $\dot{\boldsymbol{n}} = \mathbf{0}$,

$$
{}^{o}\dot{\boldsymbol{x}}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_{w} + {}^{w}\dot{\boldsymbol{x}}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_{w} + (\boldsymbol{\zeta} - \boldsymbol{I})\,\boldsymbol{\gamma}\boldsymbol{\omega}_{w}\chi R, \tag{6.37}
$$

Derivative of (6.37) reads

$$
{}^{o}\ddot{\boldsymbol{x}}_{SPV} = {}^{o}\ddot{\boldsymbol{x}}_{w} + {}^{w}\ddot{\boldsymbol{x}}_{SPV}, \tag{6.38}
$$

where

$$
{}^{w}\ddot{\boldsymbol{x}}_{SPV} = \dot{\boldsymbol{\zeta}}\boldsymbol{\gamma}\boldsymbol{\omega}_{w}\chi R + (\boldsymbol{\zeta} - \boldsymbol{I})\,(\dot{\boldsymbol{\gamma}}\boldsymbol{\omega}_{w}\chi R + \boldsymbol{\gamma}\dot{\boldsymbol{\omega}}_{w}\chi R + \boldsymbol{\gamma}\boldsymbol{\omega}_{w}\dot{\chi} R) \tag{6.39}
$$

with

$$
\dot{\boldsymbol{\zeta}} = 2\boldsymbol{\zeta}\frac{\dot{\chi}}{\chi} + 0.5\chi^{2}\boldsymbol{\gamma}\boldsymbol{\omega}_{w}\boldsymbol{n}^{T}, \tag{6.40}
$$

$$
\frac{d\boldsymbol{\gamma}\boldsymbol{\omega}_{w}}{dt} = \boldsymbol{\gamma}\dot{\boldsymbol{\omega}}_{w} + \widetilde{\boldsymbol{\omega}}_{w}\widetilde{\boldsymbol{\alpha}}\boldsymbol{\omega}_{w} - 2\widetilde{\boldsymbol{\omega}}_{w}\boldsymbol{y}_{w}\boldsymbol{n}^{T}\widetilde{\boldsymbol{\omega}}_{w}\boldsymbol{y}_{w} - \boldsymbol{y}_{w}\boldsymbol{n}^{T}\widetilde{\boldsymbol{\omega}}_{w}\widetilde{\boldsymbol{\omega}}_{w}\boldsymbol{y}_{w}, \tag{6.41}
$$

$$
\dot{\chi} = -\chi^{3}\boldsymbol{n}^{T}\widetilde{\boldsymbol{\alpha}}\boldsymbol{\omega}_{w}. \tag{6.42}
$$

To compute the SPV acceleration under the rolling constraint, the rolling constraint at the acceleration level is derived. From (3.5) and (6.1), the non-sliding pure rolling condition implies the wheel centre has to move with the velocity

$$
{}^{o}\dot{\boldsymbol{x}}_{w} = -\boldsymbol{\omega}_{w} \times {}^{w}\boldsymbol{x}_{SPV} = \boldsymbol{\omega}_{w} \times ((\boldsymbol{n} - \boldsymbol{\alpha})\chi R + \boldsymbol{n}r). \tag{6.43}
$$

At the acceleration level, (6.43) reads (3.8)

$$
{}^{o}\ddot{\boldsymbol{x}}_{w} = -\dot{\boldsymbol{\omega}}_{w} \times {}^{w}\boldsymbol{x}_{SPV} - \boldsymbol{\omega}_{w} \times {}^{w}\dot{\boldsymbol{x}}_{SPV}. \tag{6.44}
$$

Introducing (6.44) to (6.38), the derivative of (6.37) under the rolling assumption reads

$$
\ddot{\boldsymbol{x}}_{cp} = \mathbf{J}\dot{\boldsymbol{\omega}}_{w} + \dot{\mathbf{J}}\boldsymbol{\omega}_{w}, \tag{6.45}
$$

where

$$
\mathbf{J} = (\boldsymbol{\zeta} - \boldsymbol{I})\,\boldsymbol{\gamma}\chi R + {}^{w}\widetilde{\boldsymbol{x}}_{cp}
$$

$$\dot{\mathbf{J}} = (\chi^2 \boldsymbol{n}^T \boldsymbol{\alpha}\boldsymbol{\omega}_w (2\boldsymbol{I} - 3\boldsymbol{\zeta})\boldsymbol{\gamma} + 2\boldsymbol{\zeta}\widetilde{\boldsymbol{\omega}}_w\boldsymbol{\gamma} + \widetilde{\boldsymbol{\Phi}} - \widetilde{\boldsymbol{\Gamma}} - \boldsymbol{y}_w\boldsymbol{n}^T\widetilde{\boldsymbol{\omega}}_w\widetilde{\boldsymbol{y}}_w)\chi R,$$

$$\boldsymbol{\Phi} = \boldsymbol{\zeta}\boldsymbol{\gamma}\boldsymbol{\omega}_w, \quad \boldsymbol{\Gamma} = \boldsymbol{y}_w\boldsymbol{n}^T\widetilde{\boldsymbol{\omega}}_w\boldsymbol{y}_w.$$

6.4.1 Properties of the SPV acceleration

To delineated properties of (6.45), the wheel angular velocity has been decomposed into the two linear subspaces: the ground normal space ($_{|n}$) and its complement of the ground normal null-space ($_{|Pn}$); that reads $\boldsymbol{\omega}_w = \boldsymbol{\omega}_{w|n} + \boldsymbol{\omega}_{w|Pn}$. From (6.45), the following properties can be proofed directly. The SPV acceleration

- is zero along the ground normal, $\boldsymbol{n}^{T\,o}\ddot{\boldsymbol{x}}_{cp} = \mathbf{0}$;
- is independent on the wheel angular acceleration around the ground normal, i.e., $\mathbf{J}\dot{\boldsymbol{\omega}}_{w|n} = \mathbf{0}$;
- depends linearly on the wheel angular velocity along the ground normal, i.e., $\dot{\mathbf{J}}(\boldsymbol{y}_w, \boldsymbol{n}, \boldsymbol{\omega}_{w|n})\boldsymbol{\omega}_w = \mathbf{0}$ and $\dot{\mathbf{J}}(\boldsymbol{y}_w, \boldsymbol{n}, \boldsymbol{\omega}_{w|Pn})\boldsymbol{\omega}_w \neq \mathbf{0}$.

Thus a linear model on the SPVs acceleration may be defined in terms of the wheels acceleration/velocity

$$^{o}\ddot{\boldsymbol{x}}_{SPV} = \mathbf{J}_c \begin{bmatrix} \dot{\boldsymbol{\omega}}_{w|Pn} \\ \boldsymbol{\omega}_{w|n} \end{bmatrix} + \boldsymbol{A}_c, \tag{6.46}$$

where $\mathbf{J}_c = \begin{bmatrix} \mathbf{J} & \dot{\mathbf{J}}(\boldsymbol{\omega}_{w|Pn}) \end{bmatrix}$, $\boldsymbol{A}_c = \dot{\mathbf{J}}(\boldsymbol{\omega}_{w|Pn})\boldsymbol{\omega}_{w|Pn}$. (6.46) is a linear model for the SPV motion that can be used to solve the non-holonomic constraint.

6.5 Conclusion

In this chapter, a new model of the SPV in the HLW robots that does not make any assumptions on the robot kinematics or the wheel camber angle has been proposed. To this end, the wheel has been modelled as a torus rather than a standard sphere. Implications of the developed model on the motion control of a legged-wheeled and a standard wheeled mobile platform has been discussed, and a non-holonomy in these two types of robots has been outlined. Finally, the second-order kinematic model has been derived, and its properties have been delineated. It has been shown that during the pure-rolling motion, the support polygon acceleration of the legged-wheeled robot is linearly dependent on the mixed wheel angular velocity/acceleration state.

The SPV model developed in the chapter forms a basis for the robust omnidirectional driving scheme proposed for the CENTAURO robot in Chapter 7 and the reactive support polygon adaptation scheme described in Chapter 8.

Chapter 7

Robust Omnidirectional Driving Scheme

Even though the non-holonomy takes the same form for standard all-steerable platforms and legged-wheeled robots (Section 6.3.1), the nature of this constraint is different. For standard all-steerable platforms the non-holonomy for omnidirectional driving comes from the system design, while in the legged-wheeled robots it is imposed by the control requirements. This may occur to respect the workspace boundaries or to keep the system balanced. To tackle the non-holonomic constraint in the control of the legged-wheeled robot, in this chapter, an omnidirectional driving scheme is proposed for the CENTAURO robot.[1]

First, in Section 7.1 the notion of the constant support polygon is considered in different ground conditions, then the desired SPV velocity that satisfies the rolling constraint (6.28) for omnidirectional driving with the support polygon regulation is designed. The kinematic model developed in Section 6 forms the basis for the design of a first-order IK control scheme to regulate the robot posture and SPVs presented in Section 7.2. To ensure the SPV convergence in Section 7.3, a steering strategy that accounts for the non-holonomic constraint is proposed, and a damping injection method is developed to handle structural singularity in the non-holonomic constraint. Finally, the proposed omnidirectional driving scheme is verified in experiments on the CENTAURO robot in Section 7.4, and a conclusion is given in Section 7.5.

[1]The work included in this chapter has been presented at the IROS 2018 conference with the publication titled "On the kinematics of wheeled motion control of a hybrid wheeled-legged centauro robot" by Malgorzata Kamedula, Navvab Kashiri, and Nikos G. Tsagarakis. An extension of this publication has been submitted to Robotics and Autonomous Systems with the submission titled “Wheeled Motion Kinematics and Control of a Hybrid Mobility CENTAURO robot” by the same authors which is currently under revision.

7.1 SPV Velocity Shaping

Tracking the SPV with respect to the inertial frame implies a time-varying SPV reference to ensure a long distance stable motion. A more practical solution is to track the support polygon with respect to the robot reference point. The robot reference point then moves as it belongs to the same rigid-body with the SPVs, and thus the constant support polygon reference remains feasible throughout the motion on a flat ground. Knowing the velocity of the reference point on a rigid-body (${}^{o}\boldsymbol{x}_r$), and the position vector between the points (${}^{r}\boldsymbol{x}_{SPV} \in \Re^3$), the velocity of a second point such as a SPV(${}^{o}\boldsymbol{x}_{SPV}$) reads (3.3)

$$ {}^{o}\dot{\boldsymbol{x}}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_r + {}^{o}\boldsymbol{\omega}_r \times {}^{r}\boldsymbol{x}_{SPV}. \tag{7.1}$$

However, (6.28) constrains the SPV to remain on the ground plane, while it cannot be ensured simultaneously with (7.1) for an arbitrary floating base reference. To comply with any robot world posture command, the desired base reference has been decomposed to the planar and spatial motions; the planar coordinates reads

$$\begin{aligned} {}^{o}\dot{\boldsymbol{x}}_{r|\boldsymbol{n}} &= (\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T)\,{}^{o}\dot{\boldsymbol{x}}_r, \\ {}^{o}\boldsymbol{\omega}_{r|\boldsymbol{n}} &= \boldsymbol{n}\boldsymbol{n}^T\,{}^{o}\boldsymbol{\omega}_r. \end{aligned} \tag{7.2}$$

The SPV velocity for the system with constant support polygon is then described by applying (7.1) to the planar motion (7.2) only; this reads

$$ {}^{o}\dot{\boldsymbol{x}}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_{r|\boldsymbol{n}} + {}^{o}\boldsymbol{\omega}_{r|\boldsymbol{n}} \times {}^{r}\boldsymbol{x}_{SPV,\mathrm{des}}. \tag{7.3}$$

On the other hand, (7.1) for a spatial – 3D – motion reads

$$ {}^{o}\dot{\boldsymbol{x}}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_r + {}^{o}\boldsymbol{\omega}_r \times {}^{r}\boldsymbol{x}_{SPV} + {}^{r}\dot{\boldsymbol{x}}_{SPV}, \tag{7.4}$$

where vector ${}^{r}\dot{\boldsymbol{x}}_{SPV} \in \Re^3$ represents the robot internal motion that compensates for the non-planar commands. Such SPV motions ensure the constant support polygon command remains feasible throughout the robot motion on a surface, where the constant non-planar floating-base coordinates can be consistently defined for all SPVs.

However, even if this condition is met, the system balance is not ensured during the robot motion on an uneven terrain, where a relative motion of the gravity vector w.r.t. the ground normal may move the CoM out of the support polygon. To address this, the constant support polygon requirement, expressed in (7.1), may be redefined to follow the planar coordinates w.r.t the gravity vector

$$\begin{aligned} {}^{o}\dot{\boldsymbol{x}}_{r|\boldsymbol{g}} &= (\boldsymbol{I} - \boldsymbol{g}\boldsymbol{g}^{T})^{o}\dot{\boldsymbol{x}}_{r}, \\ {}^{o}\boldsymbol{\omega}_{r|\boldsymbol{g}} &= \boldsymbol{g}\boldsymbol{g}^{T\,o}\boldsymbol{\omega}_{r}, \end{aligned} \tag{7.5}$$

where $\boldsymbol{g} \in \Re^{3}$ symbolises the unit vector along the gravity direction. The rigid-body equation then reads

$${}^{o}\dot{\boldsymbol{x}}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_{r|\boldsymbol{g}} + {}^{o}\boldsymbol{\omega}_{r|\boldsymbol{g}} \times {}^{r|\boldsymbol{g}}\boldsymbol{x}_{SPV,\text{des}} + \boldsymbol{g}\dot{x}_{r|\boldsymbol{g}}, \tag{7.6}$$

where ${}^{r|\boldsymbol{g}}\boldsymbol{x}_{SPV,\text{des}}$ is the desired SPV position in the null-space of the gravity vector, and $\dot{x}_{r|\boldsymbol{g}} \in \Re^{1}$ is a scalar value that represents SPV motion in the gravity direction to comply with (6.28); for the spatial – 3D – motion (7.4) holds. Definition of the constant support polygon with respect to the gravity direction provides a direct control over the system stability. However, with a change in a ground inclination a system configuration automatically adjusts, and as a result a constant support polygon reference may become unfeasible with terrain change due to the joint/workspace limits.

Finally, a similar derivation may be provided for a system with a relative base-SPV motion. It yields

$${}^{o}\dot{\boldsymbol{x}}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_{r|\boldsymbol{g}} + {}^{o}\boldsymbol{\omega}_{r|\boldsymbol{g}} \times {}^{r|\boldsymbol{g}}\boldsymbol{x}_{SPV,\text{des}} + \boldsymbol{g}\dot{x}_{r|\boldsymbol{g}} + \boldsymbol{P}_{g}^{T}\dot{x}_{SPV,\text{des}}, \tag{7.7}$$

where $\dot{x}_{SPV,\text{des}} \in \Re^{2}$ symbolises the desired velocity of a SPV in the null-space of the gravity direction, and $\boldsymbol{P}_{g} \in \Re^{2\times 3}$ stands for the matrix projecting a 3D space into the null-space directions of the gravity vector.

7.2 Inverse Kinematics Whole-Body Control

The CENTAURO robot hybrid legged-wheeled kinematics provides a wide range of end-effector motions what benefits its mobility. However, as a result the robot balance is not ensured within the end-effectors workspace. Direct control over the SPVs placements is then central to traverse more challenging terrains such as stairs or cluttered environments. To address this concern, a non-holonomic first-order kinematics scheme is proposed. To this end, the hierarchical operational space control framework [55] is used in a way that the NSPR constraints are satisfied at all time and the system balance during the whole-body motion is ensured as the robot base and support polygon references are simultaneously tracked.

First, the inverse kinematic resolution scheme is described in Section 7.2.1, and the robot world posture task is outlined in Section 7.2.2. The tasks defined to track SPVs are introduced in Section 7.2.3. The proposed control scheme with the IK task stack is outlined in Section 7.2.4, and an extension to incorporate legged motion is given in Section 7.2.5.

7.2.1 Inverse Kinematics Scheme

The control requirements consisting of the robot world posture and SPVs positions are defined in the task-space. The lower-level joint-space position/velocity references are resolved with a hierarchical IK scheme. In this work, a scheme (3.47) has been implemented, where the desired operational space velocity ($\dot{\boldsymbol{x}}_i$) is defined as the position tracking error (3.48). This scheme directly provides reference for the velocity controlled joints after the whole stack of tasks is evaluated ($\dot{\boldsymbol{q}}_{\text{des}} = \dot{\boldsymbol{q}}_m$), and the position reference is determined through numerical integration

$$\boldsymbol{q}_{\text{des}} = \boldsymbol{q} + \dot{\boldsymbol{q}}_m T,$$

where $T \in \Re^1$ stands for the time step of the IK control loop.

Constraints Task

To ensure the compatibility of the joint-space solutions with the current/desired support state, the set of contact constraints (3.5) is computed with the highest priority in the IK scheme. As a result, generated joint space trajectories lie within the null-space of the constraints Jacobian. Thus, the recursive algorithm in (3.47) is initialized with

$$\dot{\boldsymbol{q}}_1 = -\mathbf{J}_{cp}^{+}\boldsymbol{e}_{cp}, \tag{7.8}$$

where $\boldsymbol{e}_{cp} \triangleq \mathbf{0} \in \Re^{3f}$ represents the constraints task error, $\mathbf{J}_{cp} \in \Re^{3f\times n}$ symbolises the Jacobian of the point on the wheel that is in the contact with the ground, and $f \in \mathbb{N}$ stands for the number of legs in ground contact.

7.2.2 World Posture Task

As discussed in Section 3.2.6, the system static balance is measured through the robot CoM position in the gravity vector null-space, while motion along the gravity direction has no influence on the robot static stability. For a robot with a non-negligible leg weight, as is the case with the CENTAURO robot, the pelvis/base position gives a better indication about the distance between the robot base and the ground than the CoM position along the gravity direction. Therefore, the world posture task is composed of three subtasks defined to control the CoM position in the gravity null-space, pelvis position along the gravity vector, and the pelvis spatial orientation. This task provides a clear indication about the system safety margin to the ground, and the upper-body reach; it also uniquely defines the robot world posture (${}^{o}\boldsymbol{x}_r$).

In the first subtask – a CoM regulation task – the tracking error $\boldsymbol{e}_{r1} \in \Re^2$ reads

$$\boldsymbol{e}_{r1} = \mathbf{P}_g(\boldsymbol{x}_{CoM,\text{des}} - \boldsymbol{x}_{CoM}),$$

where $\boldsymbol{x}_{CoM,\text{des}}, \boldsymbol{x}_{CoM} \in \Re^3$ represent the desired/current CoM coordinates. The task Jacobian ($\mathbf{J}_{r1} \in \Re^{2\times n}$) is defined from the whole-body CoM Jacobian determined using the sum of CoM Jacobians (${}^{j}\mathbf{J}_{CoM} \in \Re^{3\times n}$) of robot parts weighted by their masses ($m_j \in \Re^+$)

$$\mathbf{J}_{r1} = -\mathbf{P}_g \sum_{j=0}^{n_l} \frac{{}^{j}\mathbf{J}_{CoM}}{m_j}.$$

The second subtask is the pelvis height regulation. The tracking error $e_{r2} \in \Re^1$ is defined as

$$e_{r2} = \boldsymbol{g}^T({}^{o}\boldsymbol{x}_{b,\text{des}} - {}^{o}\boldsymbol{x}_b),$$

where ${}^{o}\boldsymbol{x}_b$, ${}^{o}\boldsymbol{x}_{b,\text{des}}$ represents the current/desired pelvis position. The task Jacobian $\mathbf{J}_{r2} \in \Re^{1\times n}$ can be computed from a standard rigid-body kinematics.

The third subtask, the pelvis orientation task is defined in a 3D space, and it has a quaternion as an input, where the task error ($\boldsymbol{e}_{r3} \in \Re^3$) is defined after (3.49). Similar to $\mathbf{J}_{r2}$, the task Jacobian ($\mathbf{J}_{r3} \in \Re^{3\times n}$) can be directly computed from a standard rigid-body kinematics.

Finally, the world posture tracking task is constructed by aggregating the three above tasks, so that the task error ($\boldsymbol{e}_r \in \Re^6$) reads

$$\boldsymbol{e}_r = \begin{bmatrix} \boldsymbol{e}_{r1}^T & e_{r2}^T & \boldsymbol{e}_{r3}^T \end{bmatrix}^T, \tag{7.9}$$

and the task Jacobian is defined as

$$\mathbf{J}_r = \begin{bmatrix} \mathbf{J}_{r1}^T & \mathbf{J}_{r2}^T & \mathbf{J}_{r3}^T \end{bmatrix}^T. \tag{7.10}$$

7.2.3 Support Polygon Regulation

As discussed in Section 3.5.1, the contact point constraint for the CENTAURO robot may be solved with a desired base motion for any non-singular state with a 3-DoF redundancy at each leg in contact with the ground. However, based on this assumption alone, no conclusion on the support polygon can be drawn. To guarantee correct tracking of the SPV position, additional tasks are applied based on (6.29).

Camber Angle Regulation

First, consider a wheel camber angle (φ) which measures the angle between the ground normal and the wheel plane that is a complementary acute angle to the angle from the ground vector to the wheel axis (Section 3.2.2). In other words, the desired rotation is equal to the rotation between the ground normal and the vector that is orthogonal to the rotation axis and lies in the wheel plane.

The rotation axis is defined first. Since by definition in (6.2) the vector $\boldsymbol{x}_c$ is orthogonal to the wheel axis and the ground normal, the vector $\boldsymbol{x}_c$ indicates the camber angle rotation axis.

To find the remaining vector, note that the vector $\boldsymbol{z}_c$ is defined in a way that it is orthogonal to the wheel axis (6.3), and as a result, the vector $\boldsymbol{z}_c$ lies in the wheel plane. Moreover, by definition (6.3), the vector $\boldsymbol{z}_c$ is orthogonal to the rotation axis $\boldsymbol{x}_c$. In consequence, the vector $\boldsymbol{z}_c$ lies on both the camber angle plane of rotation and the wheel plane. The camber angle reads

$$\varphi = \tan^{-1}\left(\frac{(\boldsymbol{n} \times \boldsymbol{z}_c)^T \boldsymbol{x}_c}{\boldsymbol{n}^T \boldsymbol{z}_c}\right), \tag{7.11}$$

which is a function dependent on the ground normal and robot state only.

The set of feasible camber angle values is constrained by the wheel geometry and joint limits, whereas the camber angle value determines the wheel grip on the ground. As a dynamic parameter, in this work, the camber angle value is not resolved on the kinematics level and it is considered a higher-level task. An execution of the desired camber angle is ensured through a 1-dimensional regulation task, whose error reads

$$e_\varphi = \varphi_{\text{des}} - \varphi, \tag{7.12}$$

where $\varphi_{\text{des}} \in \Re^1$ represents the task reference. The task Jacobian can be extracted from the time derivative of (7.11).

Steering Angle Regulation

The result from (6.29) implies that with constrained camber angles, the system becomes non-holonomic for an omnidirectional driving. Compliance with the non-holonomic constraint is ensured by a higher-level scheme proposed in Section 7.3 that provides reference for the steering task, which controls the wheel world rotation around the ground normal. The wheel orientation is expressed through the vector $\boldsymbol{x}_c$ that is orthogonal to the ground normal and lies in the wheel plane. The current steering angle (β) is then computed as a function of the robot state and the ground normal only

$$\beta = \tan^{-1}\left(\frac{(\boldsymbol{x}_n \times \boldsymbol{x}_c)^T \boldsymbol{n}}{\boldsymbol{x}_n^T \boldsymbol{x}_c}\right). \tag{7.13}$$

The task error ($e_\beta \in \Re^1$) reads

$$e_\beta = \beta_{\text{des}} - \beta, \tag{7.14}$$

where $\beta_{\text{des}} \in \Re^1$ stands for desired steering angle. The task Jacobian is determined from a derivative of (7.13).

Support polygon regulation

To define the SPV tracking task based on (7.7), recall that according to (6.29), the SPV is not constrained solely along $\boldsymbol{x}_s$. Tracking of the SPV is set up to follow the world posture of the robot as defined in Section 7.2.2. The task error is constructed by mapping the SPV tracking error on the rolling direction as

$$e_{SPV} = \boldsymbol{x}_s^T\left({}^{o}\boldsymbol{x}_r + {}^{r|g}\boldsymbol{x}_{SPV,\text{des}} - {}^{o}\boldsymbol{x}_{SPV}\right). \tag{7.15}$$

The task Jacobian is extracted from a derivative of (7.15), where a desired SPV velocity (${}^{o}\dot{\boldsymbol{x}}_{SPV,\text{des}} \in \Re^3$) is computed as in (7.7). The task error reads

$$\dot{e}_{SPV} = {}^{o}\dot{\boldsymbol{x}}_{SPV,\text{des}} - {}^{o}\dot{\boldsymbol{x}}_{SPV}. \tag{7.16}$$

The desired SPV position is given with respect to the robot base planar coordinates, and it can be expressed in the world frame (${}^{r|g}\boldsymbol{x}_{SPV,\text{des}}$) by the twist-swing decomposition (Section 3.4) of the reference frame rotation around the ground normal.

7.2.4 Tasks Arrangement

The proposed control scheme, depicted at Fig. 7.1, consists of six IK tasks arranged in a manner that the constraint task (7.8) is set at the top priority. It is followed by the steering tracking task constructed upon (7.14). Then, the robot base tracking task is applied

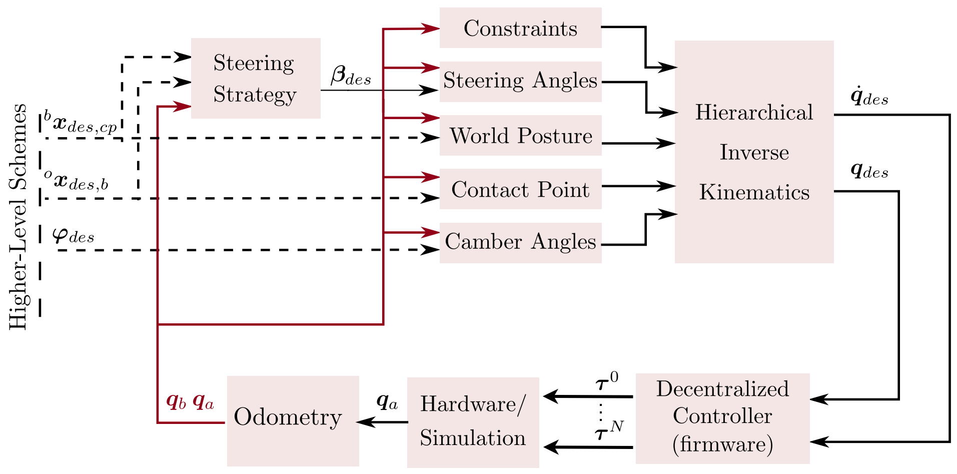


Figure 7.1 Block diagram of a designed motion control scheme.

as described in Section 7.2.2. Next, the support polygon regulation task using (7.15) and (7.16) is pushed into the stack. The final task is the camber angle task defined by (7.12).

The steering, SPV, and camber angle tracking tasks are p-dimensional tasks that gather the respective 1-dimensional tasks defined for each leg end-effector.

7.2.5 Wheeled-Legged Motion Control

The control scheme described in the Section 7.2.4 is sufficient to ensure the continuous, long-distance wheeled motion. However, in certain scenarios it is beneficial to intentionally break the ground-wheel contact to adopt stepping motion. To integrate such scenarios into the proposed control scheme, the SPV tracking task defined in (7.15) and (7.16), on the basis of the 2-dimensional SPV position, is extended to 3-dimensions. The SPV position input is set as

$$ {}^{r|\boldsymbol{g}}\boldsymbol{x}_{SPV,\text{des}} = \boldsymbol{g} x_{\boldsymbol{g},\text{des}} + \boldsymbol{P}_g^T \boldsymbol{x}_{SPV,\text{des}}, \tag{7.17} $$

where $x_{\boldsymbol{g},\text{des}} \in \Re^1$ is the desired SPV position in the gravity direction. It therefore allows for violation of the condition (6.28) holding the ground contact. The SPV tracking error becomes

$$ \boldsymbol{e}_{SPV} = [\boldsymbol{x}_s \quad \boldsymbol{y}_s \quad \boldsymbol{z}_s\ ]^T \, ({}^{o}\boldsymbol{x}_r + {}^{r|\boldsymbol{g}}\boldsymbol{x}_{SPV,\text{des}} - {}^{o}\boldsymbol{x}_{SPV}), \tag{7.18} $$

and the task Jacobian can be extracted from

$$ \dot{\boldsymbol{e}}_{SPV} = \boldsymbol{T}({}^{o}\dot{\boldsymbol{x}}_r + \boldsymbol{\omega}_{r|\boldsymbol{g}} \times {}^{r|\boldsymbol{g}}\boldsymbol{x}_{SPV,\text{des}} - {}^{o}\dot{\boldsymbol{x}}_{SPV}), $$

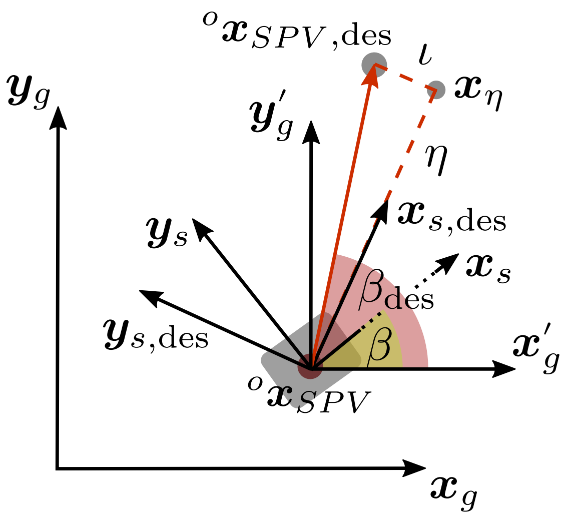


Figure 7.2 Geometric interpretation of the desired steering angle.

where the activation matrix $\boldsymbol{T}$ reads

$$\boldsymbol{T} = \begin{cases} [\boldsymbol{x}_s\ \boldsymbol{0}_{3\times1}\ \boldsymbol{0}_{3\times1}]^T & \text{if } \in \mathbb{F} \\ [\boldsymbol{x}_s \quad \boldsymbol{y}_s \quad \boldsymbol{z}_s\ ]^T & \text{if } \notin \mathbb{F} \end{cases}.$$

This Jacobian definition ensures that the SPV assumption (6.28) is satisfied even if a non-zero value has been set with ${}^{r|g}\boldsymbol{x}_{SPV,\mathrm{des}}$.

7.3 Steering Strategy

In this section, a strategy to generate the steering task reference that complies with non-holonomic constraint in (6.29) is proposed. Input to the IK controller consists of the desired robot world posture and SPV positions. However, as the non-holonomic constraint is considered, the set of feasible motions depends on the current state. Two types of input, position and velocity based, are discussed in reference to the desired steering angles in Section 7.3.1 and Section 7.3.2, respectively, and a discussion with respect to the SWMR is provided in Section 7.3.3. A strategy for tracking the robot base and support polygon references is designed in Section 7.3.4, and a method to cope with a structural singularity is presented Section 7.3.5.

7.3.1 Position Reference

Consider the robot world posture and SPVs positions are given as the controller inputs. The desired SPV placement then reads

$$^{o}\boldsymbol{x}_{SPV,\text{des}} = {}^{o}\boldsymbol{x}_{r} + {}^{r|g}\boldsymbol{x}_{SPV,\text{des}},$$

and depends only on the robot base and relative SPV. Let us assume, the robot motion is designed to satisfy (6.28) (Section 7.1), and thus a desired SPV position lies on the ground plane, then in the ground frame it reads

$$^{o}\boldsymbol{x}^{(n)}_{SPV,\text{des}} = \begin{bmatrix} ^{o}x^{(n)}_{SPV,\text{des}} & ^{o}y^{(n)}_{SPV,\text{des}} & 0 \end{bmatrix}^{T}.$$

Fig. 7.2 depicts the ground plane where current and desired SPV as well as the wheel steering angle are marked. Symbol $\iota \in \Re^{1}$ describes the SPV motion generated by the camber angle change, $\eta \in \Re^{1}$ stands for the distance travelled with the wheel rolling motion, and $\boldsymbol{x}_{\eta} = \begin{bmatrix} x_{\eta} & y_{\eta} & z_{\eta} \end{bmatrix}^{T} \in \Re^{3}$ represents the SPV position for the constant camber angle. The desired SPV in the desired steering frame ($\mathscr{F}_{s,\text{des}}$) is

$$^{o}\boldsymbol{x}^{(s,\text{des})}_{SPV,\text{des}} = {}^{o}\boldsymbol{x}^{(s,\text{des})}_{SPV} + \begin{bmatrix} \eta_{p} & \iota_{p} & 0 \end{bmatrix}^{T}, \tag{7.19}$$

and

$$\boldsymbol{x}^{(s,\text{des})}_{\eta} = {}^{o}\boldsymbol{x}^{(s,\text{des})}_{SPV,\text{des}} - \begin{bmatrix} 0 & \iota_{p} & 0 \end{bmatrix}^{T}.$$

As discussed in Section 3.5.1, the desired steering angle ($\beta_{p} \in \Re^{1}$) describes transformation from the ground frame ($\mathscr{F}_{n}$) to the desired steering frame ($\mathscr{F}_{s,\text{des}}$); therefore the vector $\boldsymbol{x}_{\eta}$ can be expressed in the ground frame as

$$\boldsymbol{x}^{(n)}_{\eta} = {}^{o}\boldsymbol{x}^{(n)}_{c,\text{des}} - \begin{bmatrix} \cos(\beta_{p}) & -\sin(\beta_{p}) & 0 \\ \sin(\beta_{p}) & \cos(\beta_{p}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \iota_{p} \\ 0 \end{bmatrix}. \tag{7.20}$$

From (6.29) follows that the wheel rolling motion renders SPV translation along the $\boldsymbol{x}_{s}$ only; therefore the desired steering angle should be design to ensure that $\boldsymbol{x}_{s}||(\boldsymbol{x}_{\eta} - {}^{o}\boldsymbol{x}_{SPV})$. It is equal to an angle from the axis $\boldsymbol{x}_n$ to the vector $\boldsymbol{x}_\eta - {}^o\boldsymbol{x}_{SPV}$

$$\beta_p = \tan^{-1}\left(\frac{[{}^o\boldsymbol{x}_n \times (\boldsymbol{x}_\eta - {}^o\boldsymbol{x}_{SPV})]^T \boldsymbol{n}}{\boldsymbol{x}_n^T(\boldsymbol{x}_\eta - {}^o\boldsymbol{x}_{SPV})}\right),$$

that can be evaluated in the ground frame as

$$\beta_p = \tan^{-1}\left(\frac{y_\eta^{(n)} - {}^o y_{SPV}^{(n)}}{x_\eta^{(n)} - {}^o x_{SPV}^{(n)}}\right), \tag{7.21}$$

where ${}^o\boldsymbol{x}_{SPV}^{(n)} = \begin{bmatrix} {}^o x_{SPV}^{(n)} & {}^o y_{SPV}^{(n)} & {}^o z_{SPV}^{(n)} \end{bmatrix}^T$. By substituting (7.20) into (7.21), one gets

$$\Delta x_{SPV} \sin(\beta_p) - \Delta y_{SPV} \cos(\beta_p) = -\iota_p,$$

where $\Delta x_{SPV} = {}^o x_{SPV,\text{des}}^{(n)} - {}^o x_{SPV}^{(n)}$, $\Delta y_{SPV} = {}^o y_{SPV,\text{des}}^{(n)} - {}^o y_{SPV}^{(n)}$. The trigonometric equation of such form solved for the angle yields

$$\beta_p = \tan^{-1}\left(\frac{\Delta y_{SPV}}{\Delta x_{SPV}}\right) + \sin^{-1}\left(\frac{-\iota_p}{\sqrt{\Delta x_{SPV}^2 + \Delta y_{SPV}^2}}\right). \tag{7.22}$$

7.3.2 Velocity Reference

For a robot moving with constant support polygon, the robot reference point and SPVs move as a rigid-body and the SPVs velocities respect (6.28). (7.3) describes the velocity of the SPV under such conditions, while (7.7) allows for variable support polygon reference at the velocity level. To compute the desired steering from the velocity inputs, note that according to (6.29) the SPV velocity is confined in the ground plane, and thus it does not move along the ground normal. In the desired steering frame, it reads

$${}^o\dot{\boldsymbol{x}}_{SPV,des}^{(s,\text{des})} = \begin{bmatrix} \eta_v \\ \iota_v \\ 0 \end{bmatrix}, \tag{7.23}$$

where $\eta_v \in \Re^1$ expresses the SPV velocity related to the wheel rolling motion and $\iota_v \in \Re^1$ symbolizes the SPV velocity associated with the camber angle motion. Furthermore, the ground frame z-axis is parallel to the ground normal and thus the condition (6.28) implies

$$ {}^{o}\dot{\boldsymbol{x}}^{(n)}_{SPV,des} = \begin{bmatrix} {}^{o}\dot{x}^{(n)}_{SPV,des} \\ {}^{o}\dot{y}^{(n)}_{SPV,des} \\ 0 \end{bmatrix}. \tag{7.24} $$

Since the rotation around the ground normal by the steering angle describes the transformation from the ground frame to the steering frame, one gets

$$ \begin{bmatrix} \eta_v \\ \iota_v \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\beta_v) & \sin(\beta_v) & 0 \\ -\sin(\beta_v) & \cos(\beta_v) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{o}\dot{x}^{(n)}_{SPV,\text{des}} \\ {}^{o}\dot{y}^{(n)}_{SPV,\text{des}} \\ 0 \end{bmatrix}, \tag{7.25} $$

where $\beta_v \in \Re^1$ represents the desired steering variable. The equation can be solved for the desired steering angle; it yields

$$ \beta_v = \sin^{-1}\left(\frac{-\iota_v}{\sqrt{{}^{o}\dot{x}^{(n)2}_{SPV,\text{des}} + {}^{o}\dot{y}^{(n)2}_{SPV,\text{des}}}} \right) + \tan^{-1}\left(\frac{{}^{o}\dot{y}^{(n)}_{SPV,\text{des}}}{{}^{o}\dot{x}^{(n)}_{SPV,\text{des}}} \right). \tag{7.26} $$

The term ι in (7.22) and (7.26) represents the change due to the camber angle modification. As these variations are often slow, the camber angle can be considered constant within a time step, i.e., $\iota \approx 0$. Moreover, solutions for position (7.22) and velocity (7.26) inputs are equivalent, where the former is a closed-loop and the latter an open-loop solution. In the next section, (7.26) has been applied to a standard all-steerable platforms.

7.3.3 Steering in all-steerable platforms

Consider (7.26) for standard all-steerable mobile robots, i.e., with the constrained camber angle $\iota_v = 0$ and the platform mobility restricted to the motion plane where ${}^{o}\dot{\boldsymbol{x}}^{(n)}_{b} =$

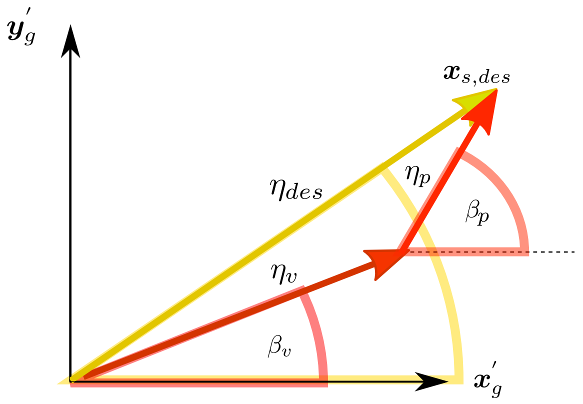


Figure 7.3 Desired steering angle in the combined position/velocity mode.

$[{}^o\dot{x}_b^{(n)}\ {}^o\dot{y}_b^{(n)}\ 0]^T$ and $\boldsymbol{\omega}_b^{(n)} = [0\ 0\ \omega_{z,b}^{(n)}]^T$. The SPV motion from (7.3) then becomes

$${}^o\dot{\boldsymbol{x}}_{SPV,\text{des}}^{(n)} = \begin{bmatrix} {}^o\dot{x}_b^{(n)} \\ {}^o\dot{y}_b^{(n)} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega_{z,b}^{(n)} \end{bmatrix} \times \begin{bmatrix} {}^b x_{SPV}^{(n)} \\ {}^b y_{SPV}^{(n)} \\ 0 \end{bmatrix}, \tag{7.27}$$

and the steering angle (7.26) simplifies to

$$\beta_v = \tan^{-1}\left(\frac{{}^o\dot{x}_b^{(n)} - \omega_{z,b}^{(n)}\,{}^b y_{SPV}^{(n)}}{{}^o\dot{y}_b^{(n)} + \omega_{z,b}^{(n)}\,{}^b x_{SPV}^{(n)}}\right).$$

This is a well-known result in the literature of all-steerable mobile robots that describes a steering motion in compliance with the ICR constraint.

Similarly, (7.26) can be shown to be consistent with solutions obtained from constraints equation in [75, 77, 79, 80] for their respective kinematics. However, it has been obtained without directly solving the system kinematics.

7.3.4 Combined Methods

Even though the non-holonomic constraint for the omnidirectional driving takes a similar form for standard all-steerable mobile platforms and legged-wheeled robots (Section 6.3.1), the nature of this constraint is not alike. For all-steerable platforms, the non-holonomic constraint is a hard constraint imposed by the system design, and the steering tracking errors increase the system's internal forces or cause the wheel to slip. The robot remains still if the non-holonomic constraint is violated. On the other hand, the non-holonomic constraint of the legged-wheeled robot is introduced by control requirements on a support polygon and/or camber angles. The robot base motion is possible without satisfying the non-holonomic constraint by moving the base relative to the support polygon. The support polygon and camber angle have lower priority in the proposed IK scheme than the NSPR constraint and the robot world posture tracking tasks, and therefore the steering tracking errors affect the execution of the support polygon reference.

The open-loop solution (7.26) resolves the wheel orientation, so that the desired SPV trajectory becomes feasible for the robot. However, it does not account for the tracking errors, which as accumulated may destabilize the system. These errors, that arise due to an imperfect realisation of the non-holonomic constraint, occur in the lateral direction to the wheel motion where the SPV motion is constrained. Meanwhile, the tracking error along the wheel rolling direction is compensated by the controller. Given that the outcome of the non-holonomic constraint is an angle, and so it is invariant to the magnitude of the tracking error, introducing SPV tracking feedback directly into the open-loop solution leads to the high variance in the computed steering reference for slow desired motions. To stabilize this approach, one can use (7.22) to introduce a tunable feedback on the SPV position. The steering reference then becomes a vector angle of the two approaches superimposed

$$\beta_{\text{des}} = \tan^{-1}\left(\frac{\mathrm{K}_v \eta_v \sin(\beta_v) + \mathrm{K}_p |\eta_v| \eta_p \sin(\beta_p)}{\mathrm{K}_v \eta_v \cos(\beta_v) + \mathrm{K}_p |\eta_v| \eta_p \cos(\beta_p)}\right), \tag{7.28}$$

where η_p and η_v symbolise the velocity vectors magnitudes computed with (7.19) and (7.23), respectively. $\mathrm{K}_v \in \Re^1$ and $\mathrm{K}_p \in \Re^1$ represent the tuning gains whose ratio ($\frac{\mathrm{K}_p}{\mathrm{K}_v}$) determines the weight in which the feedforward (β_v) and the feedback (β_p) terms contribute to the final steering reference (β_{des}). It ensures that the position feedback does not overshadow the open-loop values during the slow wheeled motion, and thus excessive steering motions due to the imminent miniscule tracking errors are eliminated.

The steering scheme (7.28) has been derived considering the robot reference point and SPV without relying on the specific robot kinematics. Thus, (7.28) can be applied to any legged-wheeled robot with controllable steering angle regardless a specific wheel assembly. However, for some legged-wheeled robots with simpler kinematics, (7.26) is sufficient. This is, for example, a case of standard mobile robots with only steerable wheels, where the support polygon is constant by hardware design. On the other hand, in the standard mobile robots with only caster wheels, their kinematic design does not allow the robot to adopt a

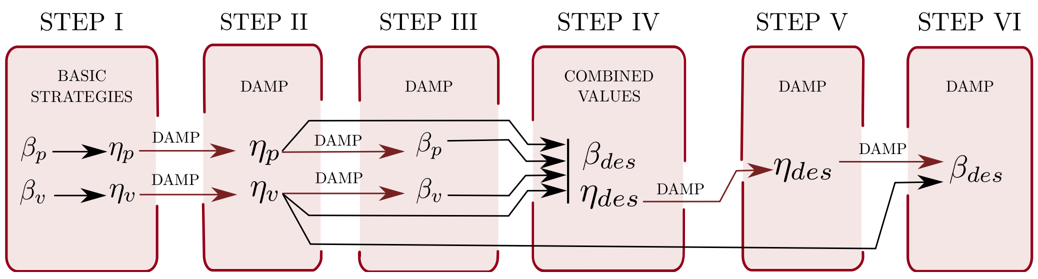


Figure 7.4 Block Diagram of the overall steering scheme.

statically unbalanced pose. In consequence, no direct support polygon control is required, and thus no steering scheme is necessary, and a holonomic omnidirectional driving is achieved.

7.3.5 Structural Singularity of the Non-Holonomic Constraint

If the SPV reference is satisfied perfectly, computation of the steering angle with (7.22), (7.26) and (7.28) leads to a singularity. At such configurations, the motion is not restraint by the non-holonomic constraint and any steering angle can be adopted. However, a small variation in the magnitude of the velocity of the desired SPV may cause an arbitrary desired steering angle. To reduce this effect and to avoid an unstable behaviour in the proximity of singularities, damping have been introduced in the computation of the desired steering angles (β_{des}, β_p, β_v) as well as the magnitudes of the velocity of the SPV (η_{des}, η_p, η_v). The overall algorithm consists of six steps shown in Fig. 7.4.

First, the steering angles for closed-loop (β_p) and open-loop (β_v) strategies are computed directly from (7.22) and (7.26). Next, the closed-loop (η_p) and open-loop (η_v) magnitudes of the velocity of the desired SPV are evaluated directly from (7.19) and (7.25).

In the step two, the magnitudes of the desired SPV motion are damped through the formula that reads

$$\begin{aligned} \eta_{p|t} &= \eta_{p|t-T} + \tanh\left(\frac{\eta_{p|t}}{\varepsilon_p}\right)(\eta_{p|t} - \eta_{p|t-T}), \\ \eta_{v|t} &= \eta_{v|t-T} + \tanh\left(\frac{\eta_{v|t}}{\varepsilon_v}\right)(\eta_{v|t} - \eta_{v|t-T}), \end{aligned} \tag{7.29}$$

where $(.)_{|t-T}$ refers to the value at the previous time step, and $(.)_{|t}$ symbolises current time step values. $\varepsilon_p \in \Re^1$ and $\varepsilon_v \in \Re^1$ denote damping threshold applied for the closed-loop and the open-loop values, respectively. The damped values from (7.29) are used as the measure of the proximity to singularity in the remainder of the damping scheme.

The step three of the damping scheme consists of damping the desired steering angles – computed in the first step of this algorithm; it reads

$$\begin{aligned}\beta_{p|t} &= \beta_{p|t-T} + \tanh\left(\frac{\eta_{p|t}}{\varepsilon_p}\right)(\beta_{p|t} - \beta_{p|t-T}),\\ \beta_{v|t} &= \beta_{v|t-T} + \tanh\left(\frac{\eta_{v|t}}{\varepsilon_v}\right)(\beta_{v|t} - \beta_{v|t-T}).\end{aligned} \tag{7.30}$$

In the step four of the damping scheme, a steering angle (β_{des}) is evaluated based on (7.28) with results from (7.29) and (7.30) sent as inputs. Then, the magnitude of the desired SPV velocity vector ($\eta_{\text{des}} \in \Re^1$) is computed based on Fig. 7.3; it reads

$$\eta_{\text{des}} = \sqrt{\mathrm{K}_v^2\eta_v^2 + \mathrm{K}_p^2\eta_p^2 + 2\mathrm{K}_v\mathrm{K}_p\eta_v\eta_p\cos(\beta_v - \beta_p)} \tag{7.31}$$

with results of (7.29) and (7.30) used as input variables.

In the step five of the damping scheme, the magnitude of the vector of the desired SPV motion (η_{des}) computed in the previous step is damped in the same way as variables in (7.29).

In the final step of the damping scheme, the desired steering is computed

$$\beta_{des|t} = \beta_{des|t-T} + \tanh\left(\frac{|\eta_{v|t}|\eta_{des|t}}{\varepsilon_v\varepsilon_{\text{des}}}\right)(\beta_{des|t} - \beta_{des|t-T}), \tag{7.32}$$

where $\varepsilon_{\text{des}} \in \Re^1$ represents a damping threshold, and the damped values from the previous steps are used as inputs.

7.3.6 Evolution of the steering scheme

Proposed form of the damping scheme as presented in this section has been chosen based on the simulation validation and hardware experiments. In contrary to the original attempt as described in [113], the tuning gains in (7.31) and the scaling factor in (7.28) and (7.32) have been introduced. As a result, the robustness of the proposed scheme improved what allowed for a successful hardware implementation of the omnidirectional driving scheme as shown in Section 7.4. Furthermore, the range of the feasible robot base velocities increased.

7.4 Results

Four experiments have been performed on the CENTAURO robot to test the proposed model and control scheme. The first experiment, described in Section 7.4.1, has been designed to evaluate the control scheme robustness to the non-perfect trajectory and steering reference singularity. The second experiment, presented in Section 7.4.2, verifies the control scheme ability to change to support polygon type, and the third experiment, outlined in Section 7.4.3, tests the control scheme ability to track the support polygon with the non-zero camber angles. Finally, the fourth experiment, described in Section 7.4.4, has been designed to compare the contact point tracking with the proposed steering scheme with respect to the standard approach.

In all of the experiments, the state estimation algorithm as described in Appendix A.1 has been used to estimate the robot world posture. The proposed control scheme executes at the embedded CENTAURO robot computer[2] in 0.4 ms, and the state estimation requires 0.1 ms. The experiments have been performed with the real-time control loop working at 500 Hz frequency. Furthermore, in simulation, the proposed framework have been validated with the non-real-time ROS plugins at the 200 Hz frequency.

Tuning the steering parameters requires a trade-off between the strong enough feedback gain K_v to ensure the convergence of a contact point, and the damping of a steering to prevent the undesirable steering in the slow motion. Impact of the open-loop and the closed-loop solutions is determined through the ratio ($\frac{\mathrm{K}_p}{\mathrm{K}_v}$) (7.28); it has been set up to ($\frac{\mathrm{K}_p}{\mathrm{K}_v} = 0.01$). This value has been determined in the simulation as the minimal value that ensures the contact point convergence. In the steering method described in Section 7.3, the support polygon regulation at the velocity level is considered, in contrary to the steering method proposed in [113] – and outlined in Section 7.3.6 – where the support polygon regulation has been provided with the closed-loop approach. This change allowed to decreased the feedback gains from $\frac{\mathrm{K}_p}{\mathrm{K}_v} = 0.33$;[3]and therefore to increase the range of feasible base velocities.

The damping parameters have been set-up to the smallest values that generated the stable motion for the system near singularity as well as for the robot moving with the very slow motion below 0.005 m/s threshold; these parameters read $\varepsilon_{\mathrm{des}} = 0.001$, $\varepsilon_p = 0.0015$, $\varepsilon_v = 0.0008$.

7.4.1 Steering strategy robustness

In the first experiment, the robot executes a set of support polygon references while concurrently it follows a complex non-smooth world posture reference that includes turning manoeu-

[2]on COM Express conga-TS170 embedded computer with Intel Core i7-6820EQ CPU.

[3]In this work, in (7.28) the scaling factor in the desired steering angle has been introduced, and thus this comparison is not direct.

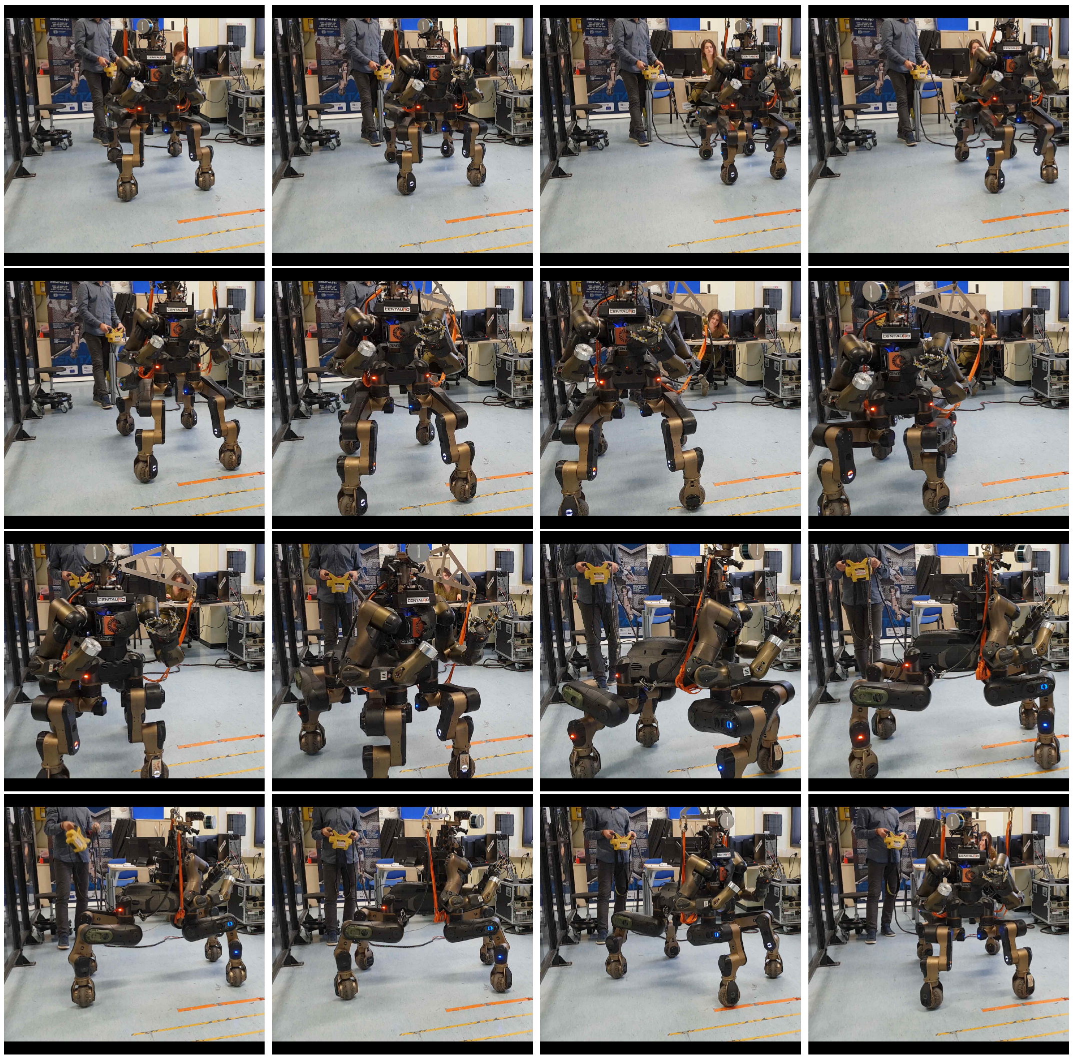


Figure 7.5 Photos from the CENTAURO robot performing the experiment testing the robustness of the motion control scheme.

Robustness of the Motion Control Scheme I set				
cm	front left	front right	rear left	rear right
mean	1.4	1.5	1.8	2.0
std	1.3	1.0	1.0	0.8
50%	1.1	1.4	1.7	2.0

Robustness of the Motion Control Scheme – II set				
cm	front left	front right	rear left	rear right
mean	1.55	1.32	1.66	1.18
std	1.32	1.18	1.61	1.06
50%	1.37	0.69	1.19	0.53

Table 7.1 Statistical data describing the SPV tracking error in the experiment testing the robustness of the motion scheme.

vres; this experiment has been executed twice.[4] Fig. 7.5 shows a few snap-shots from the experiment, and Fig. 7.6 provides corresponding support polygons and commanded/executed CoM trajectories. The control scheme robustness is tested in this experiment by defining the desired base/SPVs trajectories so, at a few time instances, the steering angle computed with (7.22) or (7.26) yields a discontinuous desired trajectory for the steering angle. Additionally, during the motion the robot crosses through the structural singularity of the non-holonomic constrain.

[4]A video recording from the first trial can be seen at https://youtu.be/ZCx8mjBIUOg.

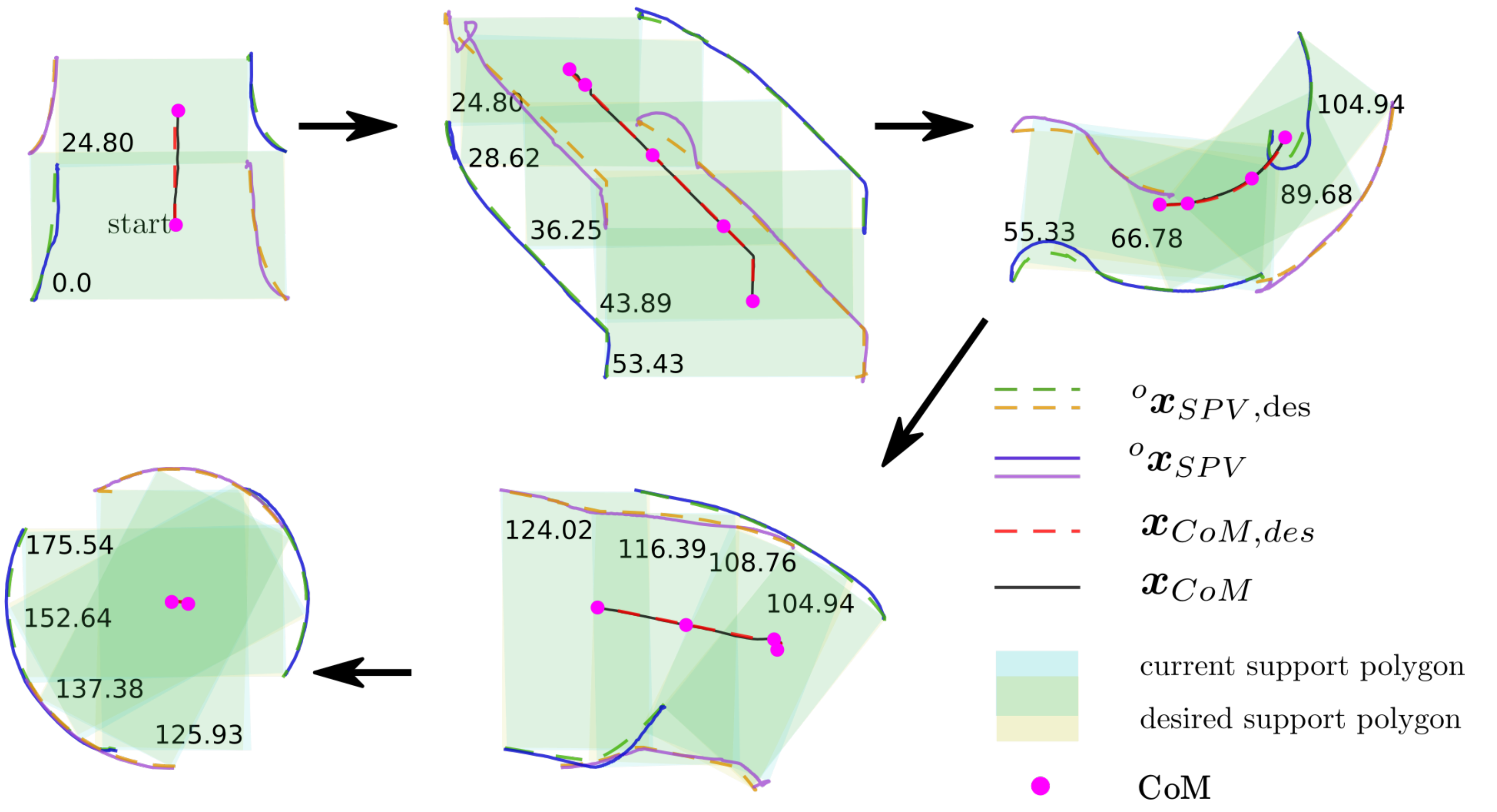


Figure 7.6 Robustness of the motion control scheme – experimental results. Segments of the desired (dashed red) and executed (solid black) CoM trajectories. The SPVs desired (dashed green, dashed orange) and executed (solid blue, solid violet) trajectories are given for a desired (yellow) and executed (blue) support polygons; green areas mark overlaps between the desired and executed support polygons. The purple dot marks the CoM corresponding with the support polygon.

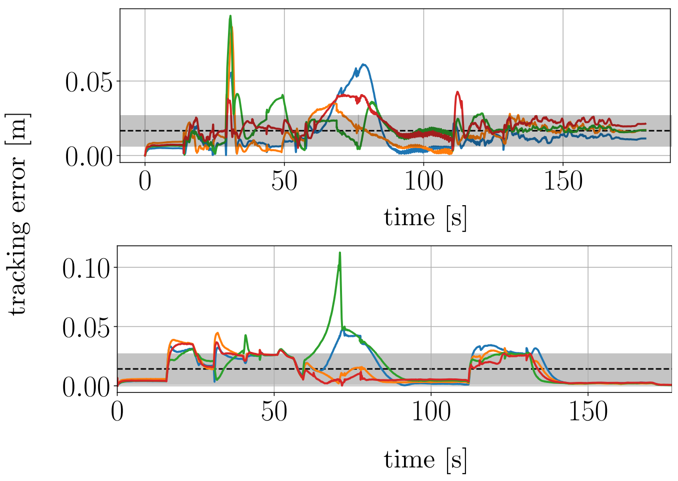


Figure 7.7 Robustness of the motion control scheme – experimental results. Contact point tracking error for front left (blue), front right (yellow), rear left (green) and rear right leg (red). The dashed black line represents error's mean, and the grey box marks points within the standard deviation. Result from the first trial on top, and the second trial at the bottom.

Fig. 7.6 shows desired and executed trajectories of SPVs and CoM, and respective support polygons at a few time steps. The average tracking error of the world posture task from both experiments are given at Fig. 7.8, where the tracking errors for the non-planar base orientation tracking task have been recorded only for the second trial. The average SPV tracking error computed with respect to the estimated robot world posture does not cross 2 cm for any leg, with a standard deviation of 1.3 cm for front left leg, 1.6 cm for rear left leg in the second run, and less than a 1 cm for other end-effectors (Table 7.1). An increase in a tracking error appears in presence of a non-smooth reference at SPV that relates to the violation of a non-holonomic constraint. These errors are compensated thanks to the feedback term (β_p) applied during the motion, and all SPV tracking errors remains under 2 cm at the end of the experiment (Fig. 7.7).

The desired steering reference (β_{des}) follows closely the open-loop solution (β_v) with a slight modification towards a direction indicated by feedback (β_p) (Fig. 7.9, top plot). A discontinuous jumps of a value about π in computed open-loop and closed-loop steering angle references at Fig. 7.9 are related to a sign switch in a desired wheel rotation due to a change in the tracking error sign. The final steering reference β_{des} however, is implemented to always choose the solution closest to the previous one, so that the configuration change is not required.

Middle plot at Fig. 7.9 shows the level of the damping generated by the scheme. It is indicated by the value of $\tanh\left(\frac{\eta}{\varepsilon}\right)$ in (7.30) and (7.32), which range includes a set of real

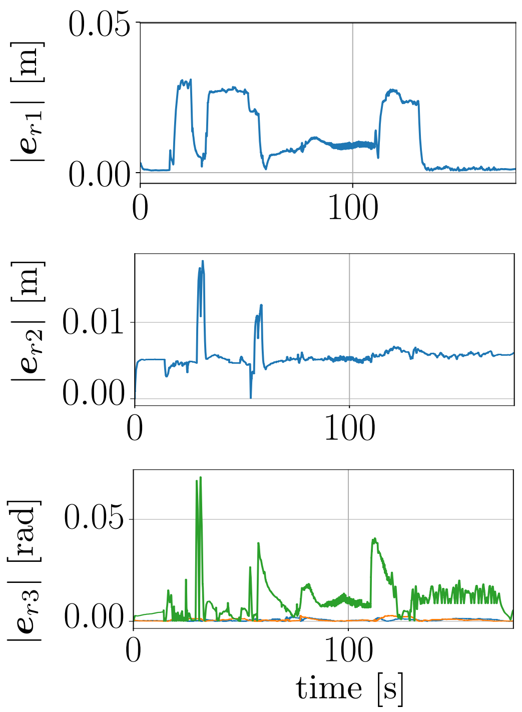


Figure 7.8 Robustness of the motion control scheme – experimental results. World posture tracking error, the CoM planar motion tracking error on top, height regulation in the middle, and the x (blue line), y (red) and z (green) coordinates orientation tracking error at the bottom.

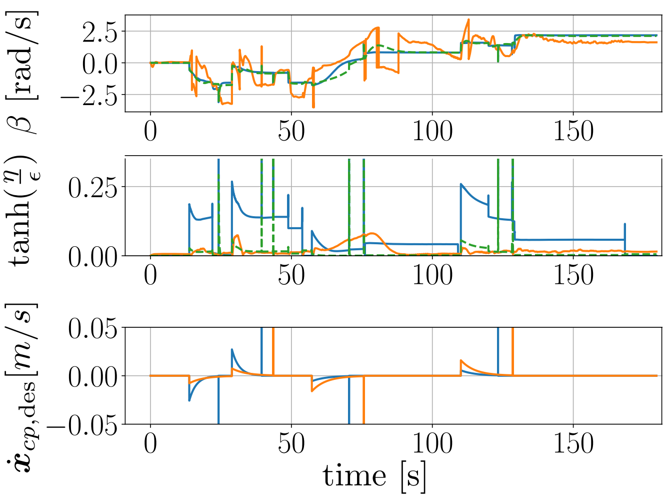


Figure 7.9 Robustness of the motion control scheme – experimental results for left front leg. Top figure: β_{des} (green), β_v (blue) and β_p (yellow); middle figure: $\tanh\left((|\eta_{v|t_i}|\eta_{des|t_i})/(\varepsilon_v\varepsilon_{\mathrm{des}})\right)$ (green), $\tanh\left(\eta_{v|t_i}/\varepsilon_v\right)$ (blue), and $\tanh\left(\eta_{p|t_i}/\varepsilon_p\right)$ (yellow); bottom figure: desired support polygon velocity – $\dot{x}_{SPV,\mathrm{des}}$ (blue) and $\dot{y}_{SPV,\mathrm{des}}$ (yellow) expressed in the frame oriented along with the robot heading.

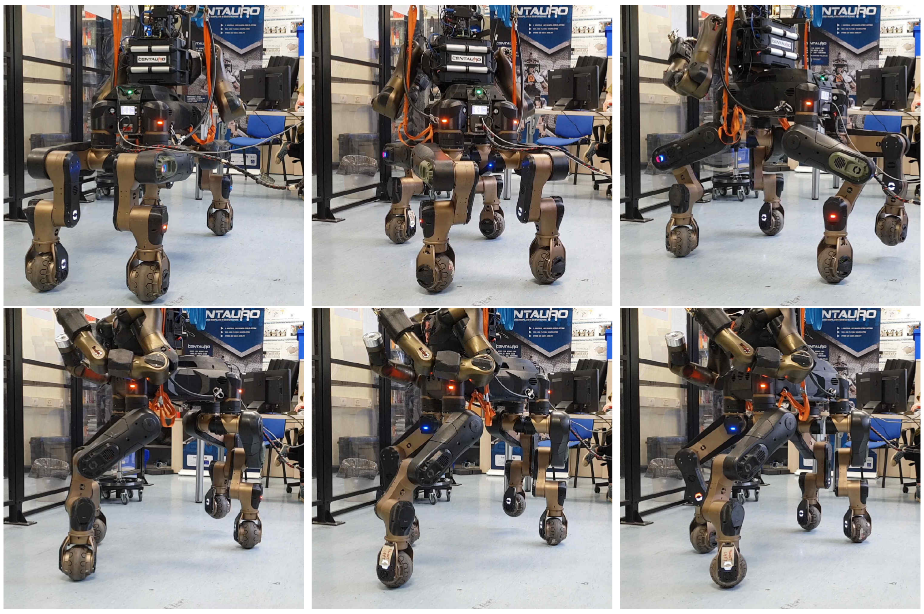


Figure 7.10 Photos of the CENTAURO robot executing the hybrid legged-wheeled motion with whole-body rotation.

numbers from 0 to 1. In the proposed damping scheme, a zero value of the hyperbolic tangent function corresponds to variable being fully damped, i.e., only the previous time step is used. On the contrary, when the hyperbolic tangent function takes the value of one, it corresponds to an inactive damping, when only a current time step is considered.

The whole motion has been confined to a 2 by 3 meter experimental space, and thus a desired robot motion is slow. The damping remains active throughout the whole motion (Fig. 7.9, middle plot) at an average value for the desired steering (β_{des}) of 0.009 and a standard deviation of 0.025 for the front left leg. The desired steering is fully damped when the contact point is commanded to remain still (e.g. 25 s), as well as during the final breaking manoeuvre. It also approaches zero, when the front left leg crosses the structural singularity between 64 s and 70 s. In this time interval, both open-loop and closed-loop solutions remain active, while the damping of the desired steering reaches its minimum value of 0.00023.

Discontinuities of values below a 0.5 mm have appeared between the reference segments. This corresponds with a jump steps in contact point reference (Fig. 7.9, bottom plot) and the level of the damping scheme activation (Fig. 7.9, middle plot) at a few time steps (e.g., 24 s, 39 s, 70 s, 123 s).

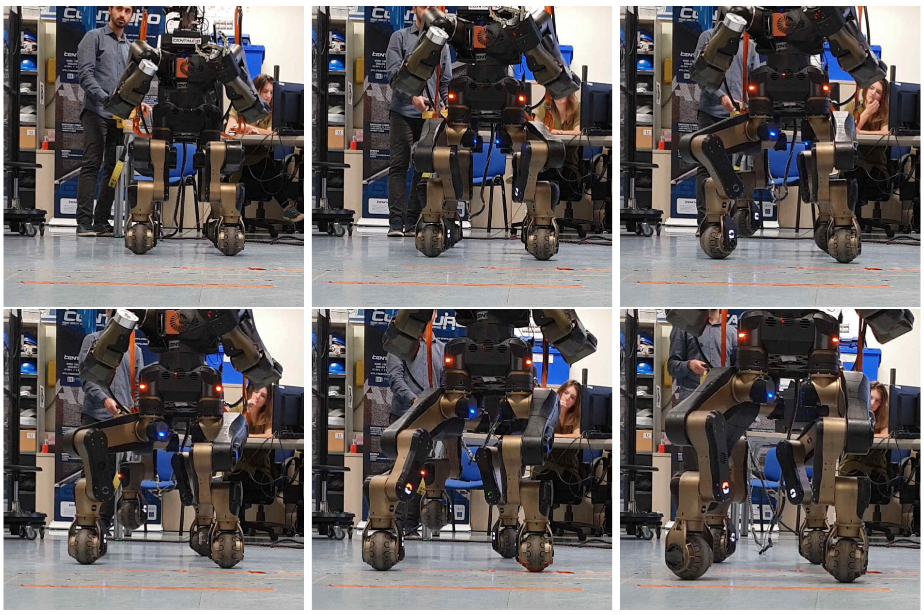


Figure 7.11 Photos of the CENTAURO robot executing the hybrid legged-wheeled motion with whole-body forward motion.

7.4.2 Variable contact polygon type

In the second experiment, the hybrid wheeled-legged motion has been executed four times. First two times, the experiment have been performed with the robot moving forward, while the desired base velocity has been increased between the trials. The third and fourth times, the experiment has been executed on the robot in pure rotation with increasing base angular velocity, while the desired base angular velocity has been increased between the trials.[5] In all experimental conditions, the same set of support polygon commands has been executed. First, during a whole-body wheeled motion a support polygon has been shifted to support the system balance on 3 legs, then a swing leg trajectory has been executed while supporting legs have been commanded to follow a time-varying trajectory. Then a full state support polygon has been restored, and a final support polygon position has been adopted. In the forward motion the robot executed the same set of support polygon references in 75 s with the base translation velocity of 0.02 m/s in the first run, and in 53 s with the base velocity of 0.04 m/s. In the experiments with the angular base motion, the robot executed the same support polygon reference in 75 s with the base velocity of 0.04 rad/s in the first run, and in 53 s with the base angular velocity of 0.1 rad/s.

Fig. 7.10 shows a few snap-shots of the CENTAURO robot executing aforementioned trajectory while rotating in place, and Fig. 7.11 shows a few snap-shots of the CENTAURO robot

[5]A video recording from the first and third experiments can be seen at https://youtu.be/joV4Xg1lo14

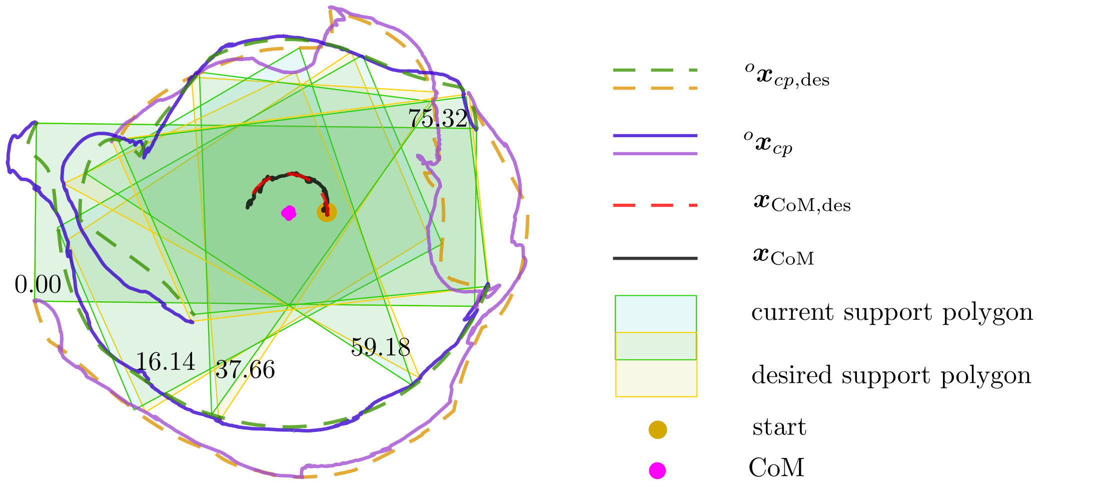


Figure 7.12 Hybrid legged-wheeled motion with base rotation – experimental results. Desired (dashed red) and executed (solid black) CoM trajectories offset 10 cm towards robot heading. The SPVs desired (dashed green, dashed orange) and executed (solid blue, solid violet) trajectories are given for a desired (yellow) and executed (blue) support polygons; green areas mark overlaps between the desired and executed support polygons. The purple dot marks the CoM corresponding with the support polygon.

executing the same support polygon trajectory while moving forward. The world posture task tracking errors are given at Fig. 7.14 and Fig. 7.15, where the non-planar base orientation tracking errors have been recorded only for the second set of experiments.

Desired and executed trajectories of SPVs and CoM, and respective support polygons at a few time steps are given for the first experiment with the base rotation at Fig. 7.12 and the first experiment with the forward motion at Fig. 7.13. To indicate the position of the robot upper-body in the base rotation experiment, at Fig. 7.12 the CoM has been marked with an offset of 10 cm towards the robot heading. The SPVs tracking errors with respect to the estimated world posture for both experiments are shown at Fig. 7.17, and Fig. 7.18; the corresponding statistical data are given at Table 7.2 and Table 7.3.

The average tracking errors for all contact points remain under 2.5 cm for all end-effectors, where the average error as well as standard deviation in the pure rotation is higher than for the forward motion. At the end of all experiments the tracking errors stay under 2 cm for all end-effectors except that of the front left leg in the first experiment, when the system is rotating in place, which has a 2.25 cm steady state position error.

The right rear leg z-coordinate reference and tracking errors for the second set of experiments are given at Fig. 7.16[6]. Fairly low tuning gains have been used for the z-coordinate

[6]These data have not been recorded for the first set of experiments.

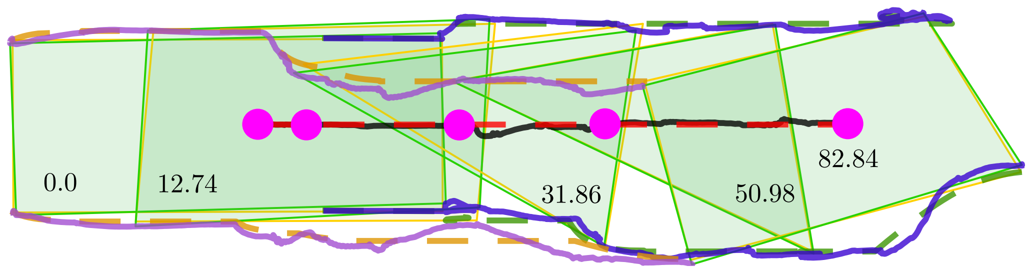


Figure 7.13 Hybrid legged-wheeled motion with base translation – experimental results. Desired (dashed red) and executed (solid black) CoM trajectories. The SPVs desired (dashed green, dashed orange) and executed (solid blue, solid violet) trajectories are given for a desired (yellow) and executed (blue) support polygons; green areas mark overlaps between the desired and executed support polygons. The purple dot marks the CoM corresponding with the support polygon.

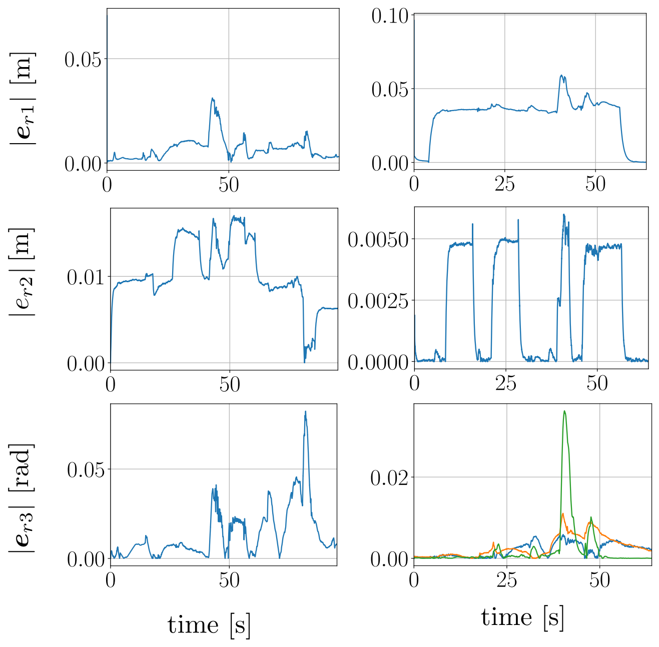


Figure 7.14 Hybrid legged-wheeled motion – results for the forward motion experiments. On the left: first experiment, on the right: second experiment. World posture tracking error, the CoM planar motion tracking error on top, height regulation in the middle, and the x (blue line), y (red) and z (green) coordinates orientation tracking error at the bottom.

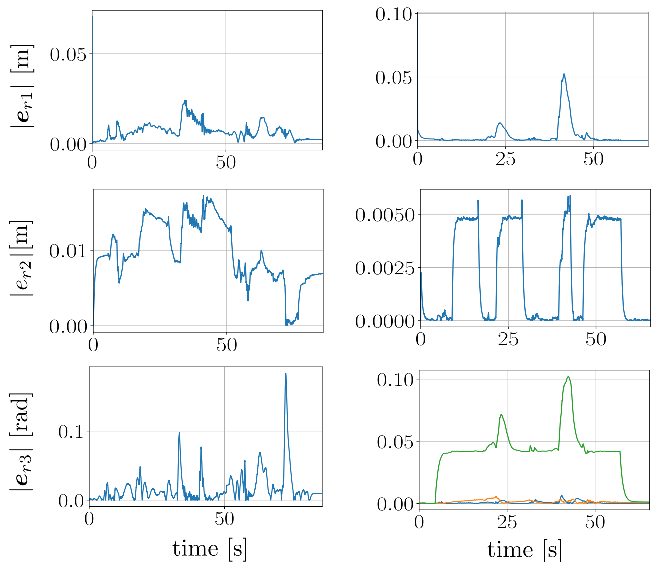


Figure 7.15 Hybrid legged-wheeled motion – results for the rotation motion experiments. On the left: first experiment, on the right: second experiment. World posture tracking error, the CoM planar motion tracking error on top, height regulation in the middle, and the x (blue line), y (red) and z (green) coordinates orientation tracking error at the bottom.

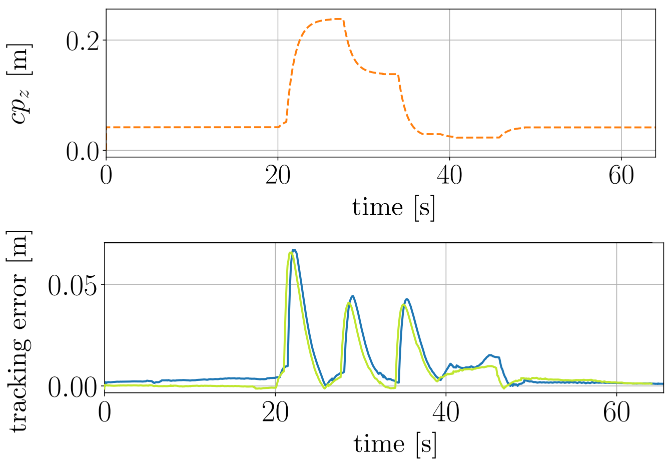


Figure 7.16 Hybrid legged-wheeled motion – results for the second set of experiments for the right rear leg. Top plot: reference of the contact point z-coordinate, bottom plot: z-coordinate tracking error for the robot rotating in place (green line) and the forward motion (blue).

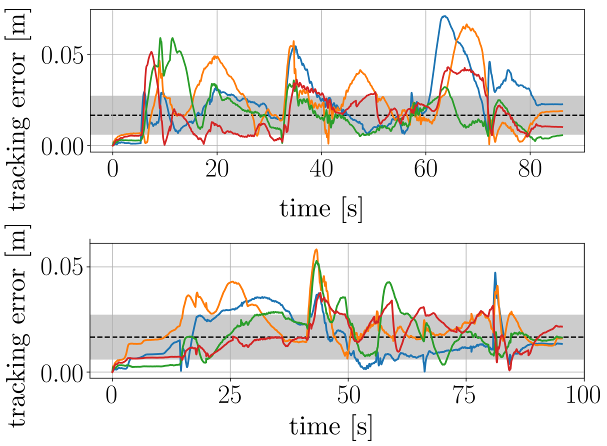


Figure 7.17 Hybrid legged-wheeled motion – results for the first set of experiments. Top plot: base rotation, bottom plot: forward motion. SPV tracking error for front left (blue), front right (yellow), rear left (green) and rear right leg (red). The dashed black line represents error's mean, and the grey box marks points within the standard deviation.

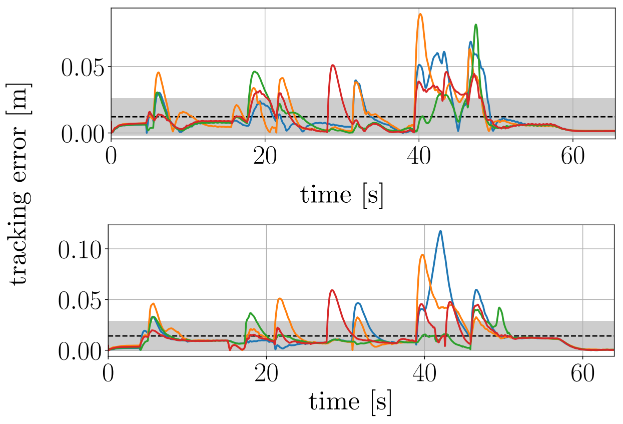


Figure 7.18 Hybrid legged-wheeled motion – results for the second set of experiments. Top plot: base rotation, bottom plot: forward motion. SPV tracking error for front left (blue), front right (yellow), rear left (green) and rear right leg (red). The dashed black line represents error's mean, and the grey box marks points within the standard deviation.

rotation - I set				
cm	front left	front right	rear left	rear right
mean	2.45	2.44	1.79	1.80
std	1.51	1.53	1.22	1.18
50%	2.26	2.02	1.54	1.65

translation - I set				
cm	front left	front right	rear left	rear right
mean	1.35	2.00	1.67	1.54
std	0.82	0.87	1.06	0.81
50%	1.15	1.87	1.63	1.29

Table 7.2 SPV tracking error statistical data for the first set of hybrid motion experiments.

contact point tracking task, and thus tracking errors up to 5 cm have been recorded. However, with time, the tracking errors converge to zero. It should be noted that, the information whether the contact point was touching the ground was hard-coded based on the system performance in simulations (Chapter 4). It resulted in a noticeable, unknown delay between the moment the contact point touched the ground and the moment the controller had received the information about the contact being restored. That caused the system disturbance and may explain the noticeable increase in the tracking errors at the moments of establishing/breaking contacts.

7.4.3 Experiment with a non-zero camber angle

In the third experiment, the robot has been rotating with the constant velocity of 0.1 rad/s while extending its support polygon.[7] When the robot approached its leg workspace limits in the standard configuration with the wheels remaining parallel to the ground normal, the non-zero camber angle configurations have been adopted to extend reach of the robot legs and thus allow the robot to further increase the support polygon.

[7] A video recording from this experiment can be seen at https://youtu.be/jymWhvXupno.

rotation - II set				
cm	front left	front right	rear left	rear right
mean	1.28	1.38	1.00	1.20
std	1.52	1.58	1.19	1.21
50%	0.68	0.70	0.61	0.69

translation – II set				
cm	front left	front right	rear left	rear right
mean	1.60	1.61	1.12	1.31
std	1.97	1.63	0.85	1.14
50%	0.95	1.07	0.93	1.01

Table 7.3 SPV tracking error statistical data for the second set of hybrid motion experiments.

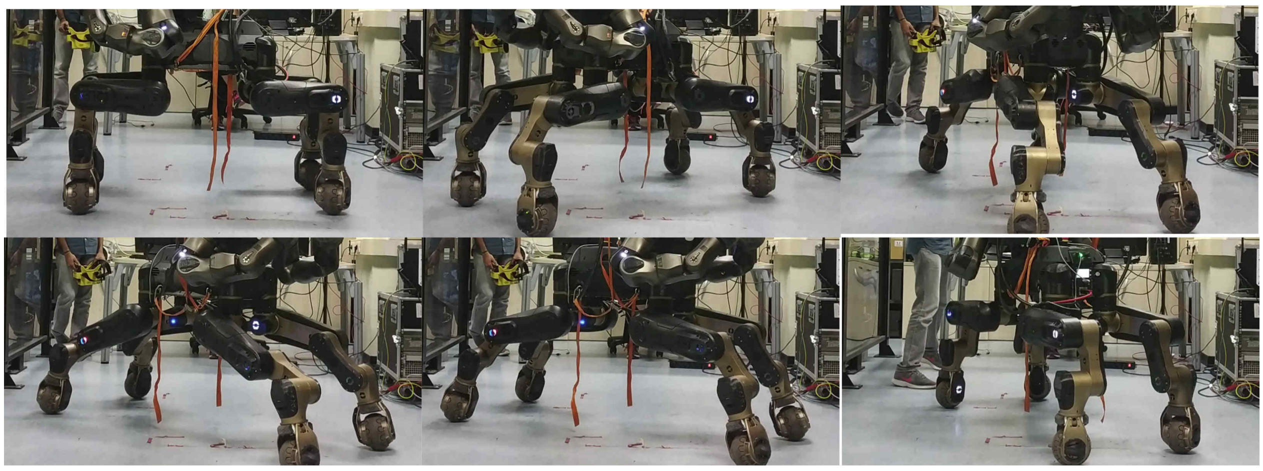


Figure 7.19 Photos of the CENTAURO robot executing the non-zero camber angle reference.

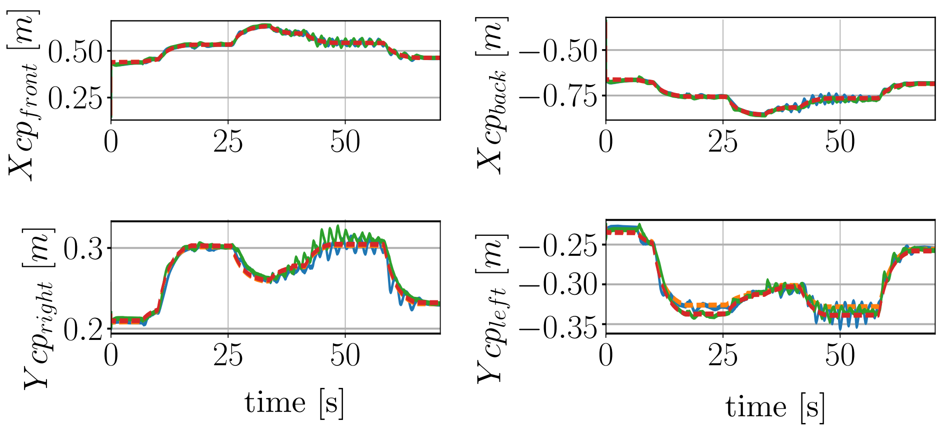


Figure 7.20 Non-zero camber angle experiment. X-coordinates of the SPV desired/executed trajectory on top; the front and rear SPVs on the left/right, respectively. Y-coordinates of the SPV desired/executed trajectory at the bottom; the left and right SPVs on the left/right, respectively.

Fig. 7.19 presents a few snap-shots from the experiment, and the Fig. 7.20 shows the desired/executed SPV trajectories with respect to the estimated robot world posture. The SPVs tracking errors with respect to the estimated robot world posture and the desired/adopted camber angles are shown at Fig. 7.21; the corresponding statistical data that describes the SPVs tracking errors are given at Table 7.4. Furthermore, at Fig. 7.22, the world posture tracking errors have been shown. The presented results show that the camber angle is tracked accurately, and the average SPV tracking errors remain under 1.0 cm for all contact points with the standard deviation remaining around 0.5-0.6 cm for all legs.

Note that in the non-zero camber angle configuration, the CoM motion is induced whenever the steering position is changed because these motion requires adjustment of all 6-DoFs of a leg. Furthermore, this experiment has been performed with the higher steering feedback gains $\frac{K_p}{K_v} = 0.03$ that the minimum required to maintain the support polygon stability $\frac{K_p}{K_v} = 0.01$[8], thus bigger steering adjustments have occurred. The slight steering adjustments induced by the higher steering feedback gains generated the CoM motions, which caused higher tracking errors in two legs in the second part of the experiment (Fig. 7.21).

The experiment shows that the controller allows to track the support polygon with the non-zero camber angle position, what effectively increases the robot workspace by allowing the robot to adopt the positions not reachable with the controllers constraint to the zero camber angle states.

[8]as used in the remaining experiments

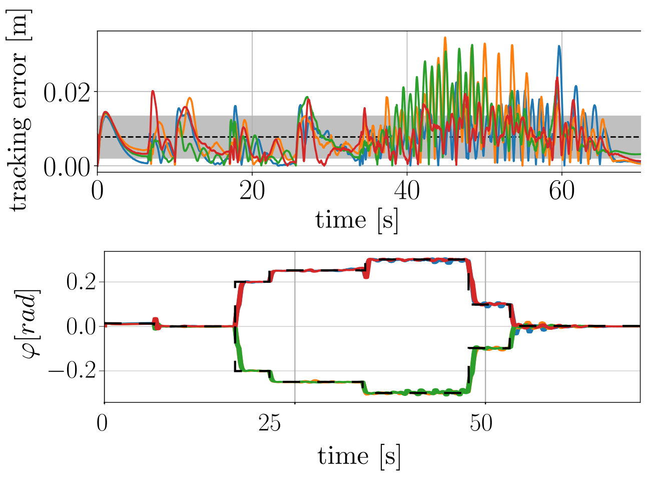


Figure 7.21 Non-zero camber angle experiment. Contact point tracking error on top and the desired/executed camber angle at the bottom for front left (blue), front right (yellow), rear left (green) and rear right leg (red). On top: the dashed black line represents error's mean, and the grey box marks points within the standard deviation, at the bottom: dashed black line representing the camber angle reference.

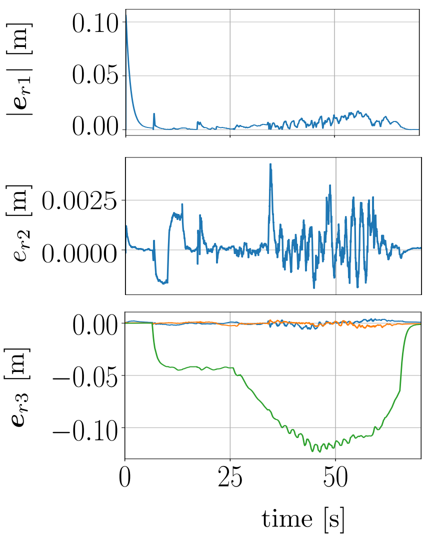


Figure 7.22 Non-zero camber angle experiment. World posture tracking error. The CoM planar motion tracking error on top, height regulation in the middle, and the x (blue line), y (red) and z (green) coordinates base orientation tracking error at the bottom.

Non-zero camber angle experiment				
cm	front left	front right	rear left	rear right
mean	0.7	0.9	0.8	0.7
std	0.5	0.6	0.6	0.5
50%	0.6	0.7	0.6	0.6

Table 7.4 Statistical data describing the SPV tracking error in the experiment with the non-zero camber angle.

7.4.4 Evaluation of the combined steering approach

In the fourth experiment, the robot has been given a constant forward velocity reference of 0.04 m/s and the constant support polygon reference. The experiment have been performed twice in the robot with high impedance gains of 300 to 2000 Nm/rad on the lower-level control. In the first trial, no steering feedback has been applied, what corresponds with the standard method to resolve the non-holonomic constraint at the velocity level (7.26). In the second trial, the feedback gain of a value $\frac{K_p}{K_v} = 0.01$ have been applied, and the full proposed steering scheme (7.28) have been active.

At Fig. 7.23, the y-coordinate of the contact point tracking error with respect to the estimated robot world posture are shown. These values represent the tracking error in the direction perpendicular to the desired base motion. The results show that, when the proposed steering method is applied the initial contact point tracking errors, that come from the sensor noise and state estimation, converge to zero. On the other hand, when the standard steering approach have been applied, for 3 legs, the tracking errors increased during the motion.

In this experiment, the robot moved by 3.66 m, and the overall tracking errors in the experiment without steering feeedback have not crossed 2 cm. On the longer trajectory, the support polygon could diverge, or it could find a stable position. When the high impedance gains or stiff position control are applied, the support polygon may find the stable position and keep the support polygon bounded even when no steering feedback is applied. However, when the robot is controlled with softer impedance gains or the desired base trajectory requires the motion of the wheel steering angles, the bounded support polygon cannot be ensured for the platform with CENTAURO robot kinematics if the non-holonomic constraint is resolved without the feedback.

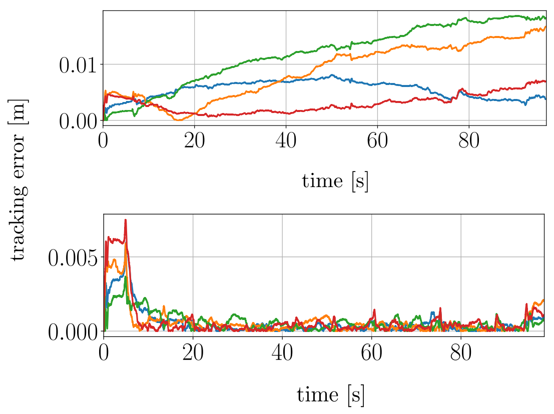


Figure 7.23 Evaluation of the steering feedback. The contact point y-coordinate tracking results for the forward motion without the steering feedback on top, and the proposed steering method at the bottom. Results for front left (blue), front right (yellow), rear left (green) and rear right leg (red).

7.5 Conclusion

In this chapter, an omnidirectional driving scheme for the CENTAURO robot has been proposed. To that end, a first-order IK scheme that resolves a whole-body motion of the CENTAURO robot, and that supports both wheeled and legged mobility, has been developed based on the kinematic model developed in Chapter 6. In this control scheme, a continuous wheel motion is resolved through the IK scheme which generates robot motion compliant with the NSPR condition. It tracks world posture of the robot, its SPVs, camber and steering angles. Finally, a higher-level scheme resolving a steering motion to comply with the non-holonomic constraint has been designed, and the damping scheme has been proposed to tackle the structural singularity.

The proposed control scheme has been tested in the set of experiments designed to evaluate motion control scheme robustness to a non-smooth base trajectory, its performance near the structural singularity, and its ability to change a support polygon type during the whole-body motion. Presented experimental results show that the proposed control scheme produces a stable motion and correct tracking of a given world posture reference and the SPVs positions. The system remains stable in presence of a non-smooth reference, and it generates continuous steering reference near the structural singularity. Furthermore, presented results validate the controller ability to support 3 and 4 leg configurations, as well as its capability to modify a support polygon configuration during the whole-body motion of the robot. The controller ability to track the desired wheel camber angles while adjusting the robot support polygon has been experimentally verified. Finally, the proposed approach to resolve the non-holonomic constrained has been experimentally compared with the standard method.

The proposed controller has been formulated in such a way that non-flat and uneven grounds can be incorporated directly through the ground normal direction near the wheel. However, further experiments are needed to verify the controller performance in these situations. Moreover, experiments in the scenarios closer to the real-world applications would provide a valuable feedback on the controller performance.

Chapter 8

Reactive Support Polygon Adaptation

In this chapter[1], a new reactive control scheme has been proposed for the CENTAURO robot[2]. The developed controller engages the robot articulated leg and steerable wheels to continuously adjust the support polygon in the entire 2-D space of the ground plane to response to unknown disturbances. To this end, a novel quadratic programming task has been designed in the cartesian-space to push the system support polygons away from the robot CoM, while respecting the leg workspace limits. To account for the hardware limits the cartesian-space optimisation task is expressed in the joint-space. To that end, to resolve the non-holonomic constraint in the SPV tracking (Section 6.3.1) in the means of the quadratic programming, a second-order kinematic model developed in Section 6.4 is used. The proposed control is experimentally verified on the CENTAURO robot, demonstrating the support polygon adjustment when external disturbances are applied to the robot.

This chapter is organised as follows; The cartesian-space support polygon adaption scheme is proposed in Section 8.1, the second-order SPV kinematic model is expressed in the terms of the joint-space velocities in Section 8.2, and the joint-space task is introduced in Section 8.3. The experimental results are given in Section 8.4, and the conclusions are drawn in Section 8.5.

[1]The work included in this chapter has been presented in the IEEE Robotics and Automation Letters article titled "Reactive Support Polygon Adaptation for the Hybrid Legged-Wheeled CENTAURO robot" by Malgorzata Kamedula and Nikos G. Tsagarakis.

[2]In this chapter, only the lower-body kinematics is considered. The upper-body motion is expressed through the whole-body CoM.

8.1 Support polygon adaptation task

This section examines two main factors that restrict the legged robots motion in the cartesian-space: the support polygon stability margin (SPSM) that quantifies the system balance, and the workspace boundary that defines a robot capability to move an end-effector in a given direction. Then, the support polygon adaptation task is formulated and discussed.

In the presence of sudden disturbance, steady-state position errors in lower-level controller increase, and a theoretically balanced reference may generate an unbalanced pose, see Fig. 8.1 on the left, where the disturbance applied to the robot pushes the CoM outside the support polygon. To counteract this effect, a reflex motion that adjusts the support polygon to improve the system balance has been developed in this section. The proposed scheme takes advantage of the 6-DoF CENTAURO robot leg kinematics by actively using its steering motion to regulate the support polygon and enhance the robot balance.

The proposed task combines two approaches to maintain the system balance. First, it adjusts the SPVs to ensure sufficient SPSM of the disturbed CoM, as shown in Fig. 8.1 in the middle. At Fig. 8.1, the CoM is pushed to the side, and the SP (light purple area) does not fully encapsulate the CoM safety boundary (cyan circle). In response, the SPVs are moved to the side to extend the SP (purple area), so the CoM safety boundary (green circle) lies within the new SP (purple area). Since the robot legs are not massless, the CoM is moved towards the direction of SPVs motion (green versus cyan circle). However, the SP edge on the left side moves more than the CoM, and the SPSM is increased.

In the second reflex, shown at Fig. 8.1 on the right, when part of the CoM safety margin (cyan circle) lies outside the SP (light purple area), the legs mass – and thus the CoM (green circle) – is shifted away from the SP edge, and the SPSM is increased. The SP extends (purple area) with the SPVs motion. Since, for the affected SP edge, the CoM safety margin was within the SP, the system balance is not significantly affected, unless the leg workspace limit is approached.

8.1.1 Support Polygon Stability Margin

In this section, the SPSM is defined, and the linear model for the SPSM is derived. The SPSM is the shortest distance from the robot CoM to the side of the convex polygon constructed upon robot-ground contact points. A robot is statically balanced when its CoM remains within its support polygon (Section 3.2.6). Thus, the SPSM measures how far the system is

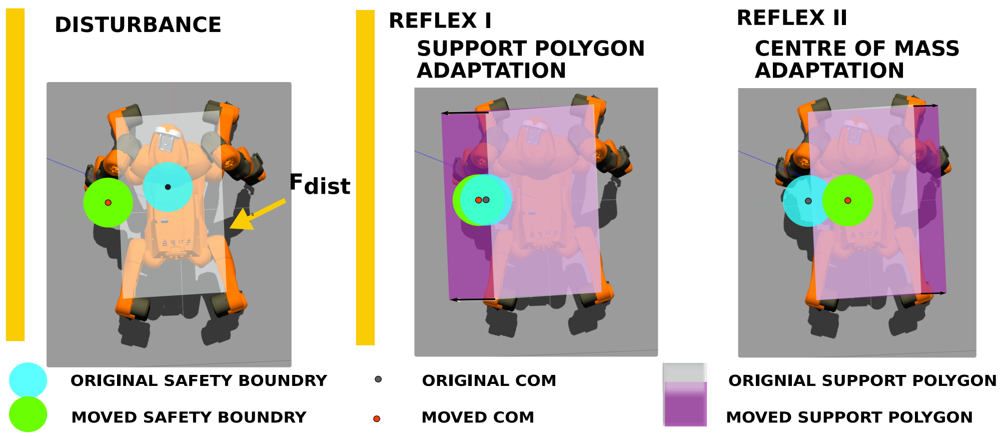


Figure 8.1 Reactive support polygon adaptation in response to the unknown disturbance - concept. On the left, robot response to the disturbance with no reactive control; in the middle, support polygon edge adjustment and the CoM adaptation on the right.

from an unbalanced pose. The SPSM reads

$$\mathrm{m} = \min_{i \in \mathcal{E}} \mathrm{m}_i,$$

where set $\mathcal{E}$ includes all polygon edges described by the pair of adjacent SPVs, and

$$\mathrm{m}_i = (\boldsymbol{x}_{CoM} - \boldsymbol{x}_{SPV,j}) \times \frac{(\boldsymbol{x}_{SPV,k} - \boldsymbol{x}_{SPV,j}) \times \boldsymbol{n}}{|(\boldsymbol{x}_{SPV,k} - \boldsymbol{x}_{SPV,j}) \times \boldsymbol{n}|}, \tag{8.1}$$

where $\boldsymbol{n} \in \Re^3$ symbolises the ground normal vector, $\boldsymbol{x}_{CoM} \in \Re^3$ stands for the CoM position vector, $j,k \in \mathcal{L}$, where $\mathcal{L}$ denotes a set of robot legs in the ground contact. Vectors $\boldsymbol{x}_{SPV,j}$ and $\boldsymbol{x}_{SPV,k} \in \Re^3$ represent the position of the support polygons describing an edge i, and thus only pairs of adjectent SPVs are considered as feasible j,k pairs in (8.1). In practice, to maintain the system balance, an offset between the support polygon edge and the system CoM is needed to account for the modelling inaccuracies, unpredicted disturbances and control tracking errors. These minimum SPSMs, defined for each support polygon edge ($\boldsymbol{m}_{min} \in \Re^{|\mathcal{E}|}$), are referred to as 'safety margins' in this work. The SPSM is described by the non-linear function (8.1) with respect to the support polygons. To express the safety margin limits in a linear form, (8.1) has been moved to the velocity space. Accounting for the rolling motion that constraints the support polygon to stay on the plane, i.e., $\forall i \in \mathcal{L} : \boldsymbol{n}^T \dot{\boldsymbol{x}}_{SPV,i} = 0$ (6.28); the time derivative of (8.1) reads

$$\dot{\boldsymbol{m}} = \begin{bmatrix} \dot{\mathrm{m}}_1 \dots \dot{\mathrm{m}}_{|\mathcal{E}|} \end{bmatrix}^T = \mathbf{J}_m^{SPV} \dot{\boldsymbol{x}}_{SPV} + \mathbf{J}_m^{CoM} \dot{\boldsymbol{x}}_{CoM}, \tag{8.2}$$

where $\mathbf{J}_m^{SPV} \in \Re^{|\mathcal{E}|\times 2p}$ and $\mathbf{J}_m^{CoM} \in \Re^{|\mathcal{E}|\times 3}$ denote respective Jacobians; $\boldsymbol{x}_{SPV}^T = \left[\boldsymbol{x}_{SPV,1|Pn}^T \dots \boldsymbol{x}_{SPV,p|Pn}^T\right]$, $\boldsymbol{x}_{SPV,i|Pn} \in \Re^2$ describes the ith leg support polygon in the ground normal null-space.

8.1.2 Workspace Boundaries

In this section, the robot workspace is discussed, and the linear model is derived. In this work, the reach of a leg is considered to be the main indicator of the robot workspace; the joint-space limits are tackled separately in Section 8.2. The maximum leg reach is a constant characteristic of the hardware that depends on the system kinematic arrangement and length of the robotic limbs. For most of the standard robotic systems, the workspace boundary can be approximated by the distance between the two distal points of the fully extended leg. Consequently, a measure of the distance from the workspace limits is a distance between these two distal points at the current time step. This value is referred to as the 'virtual leg length' in this work.

For the CENTAURO robot leg, the two distal points are the point defined by the cross-section between the hip yaw and hip pitch axes ($\forall i \in \{1, \dots p\} : \boldsymbol{x}_{h,i} \in \Re^3$) and the wheel contact point. Slightly smaller values than the fully extended legs have been considered as a limit to a reachable leg lengths ($\boldsymbol{w}_{max} \in \Re^p$) to account for the wheel camber angle limits. For convenience, the square of the adopted virtual leg lengths ($\boldsymbol{w} = \left[{w_1}^2 \dots {w_p}^2\right]$) is computed, where

$$\forall i \in \{1, \dots p\} \quad w_i^2 = (\boldsymbol{x}_{SPV,i} - \boldsymbol{x}_{h,i})^T (\boldsymbol{x}_{SPV,i} - \boldsymbol{x}_{h,i}). \tag{8.3}$$

The vector of maximum workspace limits $\boldsymbol{w}_{max}$ should be adjusted accordingly.

To define the workspace limits in the linear form, (8.3) has been expressed at the velocity level. In the rolling motion – i.e assuming that $\boldsymbol{n}^T \dot{\boldsymbol{x}}_{SPV,i} = 0$ – it reads

$$\dot{\boldsymbol{w}} = \mathbf{J}_w^{SPV} \dot{\boldsymbol{x}}_{SPV} + \mathbf{J}_w^h \dot{\boldsymbol{x}}_h, \tag{8.4}$$

where $\mathbf{J}_w^{SPV} \in \Re^{p\times 2p}$ and $\mathbf{J}_w^h \in \Re^{p\times 3p}$ denote respective Jacobians, and $\boldsymbol{x}_h^T = \left[\boldsymbol{x}_{h,1}^T \dots \boldsymbol{x}_{h,p}^T\right]$.

8.1.3 Support Polygon Adaptation Task in the Cartesian-Space

The proposed support polygon adaptation task is designed to minimise the support polygons motion while respecting the safety margin and maximum workspace constraint. The task reads

$$\begin{aligned}
\underset{\dot{\boldsymbol{x}}_{SPV},\,\boldsymbol{\delta}}{\text{minimise}} \quad & \dot{\boldsymbol{x}}_{SPV}^T \dot{\boldsymbol{x}}_{SPV} + \boldsymbol{\delta}^T \boldsymbol{W} \boldsymbol{\delta} \\
\text{subject to} \quad & \mathbf{J}_m^{SPV} \dot{\boldsymbol{x}}_{SPV} \geq \Delta \boldsymbol{m}_{min} - \boldsymbol{\delta}_m \\
& \mathbf{J}_w^{SPV} \dot{\boldsymbol{x}}_{SPV} \leq \Delta \boldsymbol{w}_{max} + \boldsymbol{\delta}_w \\
& \boldsymbol{\delta}_w \geq \mathbf{0}, \quad \boldsymbol{\delta}_m \geq \mathbf{0}
\end{aligned} \tag{8.5}$$

where $\boldsymbol{\delta}^T = \begin{bmatrix} \boldsymbol{\delta}_w^T & \boldsymbol{\delta}_m^T \end{bmatrix}$, $\boldsymbol{\delta}_w \in \Re^p$, $\boldsymbol{\delta}_m \in \Re^{|\mathcal{E}|}$ represent the slack variables in the constraints of the workspace/SPSMs, respectively; $\boldsymbol{W} \in \Re^{(p+|\mathcal{E}|)\times(p+|\mathcal{E}|)}$ denotes diagonal weight matrix. The position SPSM/workspace limits are expressed at the velocity level through the finite difference between the designed limits and the current values: $\Delta(.)_{min/max} = \frac{(.)_{min/max}-(.)}{T}$, where $T \in \Re^1$ is the controller time step. The matrices $\mathbf{J}_m^{SPV}$ and $\mathbf{J}_w^{SPV}$, defined in (8.2) and (8.4), relate the support polygons velocities to the estimated SPSM/workspace velocity limits.

In (8.5) the SPSM/workspace limits are defined as soft inequality constraints thorough the slack variables. If these limits are set-up over the theoretical safety margin (Section 8.1.1) and below the hardware workspace limits (Section 8.1.2), the task detects that the system approaches its limits and pushes the support polygon towards the more robust configuration before the physical limit has been reached. Since (8.5) minimises the support polygons motion subject to the inequality constraints, the support polygon remains constant as long as the robot stays within its safety boundaries/workspace limits.

In the cartesian-space, there is no straight-forward way to accurately model the impact of the support polygon motion on the robot CoM. In the design of (8.5), a simple quadruped model with the massless legs has been used. This assumption is removed in Section 8.2, where the joint-space control is proposed. The applied disturbance is considered to be comprised in the base motion, and the hip origins are rigidly attached to the robot base. As a result, the CoM ($\dot{\boldsymbol{x}}_{CoM}$) and hip ($\dot{\boldsymbol{x}}_h$) motions are fully defined by the base motion. With the massless legs, the SPVs motion does not affect the system CoM. Thus, only the first approach to improve the system balance (Fig. 8.1) could be implemented. Note, that only the part of Jacobians associated with support polygons motion is taken into account in the constraints in (8.5), and so, the support polygon adaptation starts when the soft limits have already been violated on the hardware.

8.2 Joint-space model

In this section, limitations of the operational-space formulation in (8.5) are discussed, and non-holonomy in the first-order kinematic model that results in the non-linear function with respect to the wheel steering motion is recalled. Then, the second-order kinematics derived in Section 6.4 is moved to the space of joint velocities.

The cartesian-space task (8.5) generates the minimal support polygon motion that satisfies the safety margin and robot workspace limits. However, (8.5) assumes that the desired wheel rotation can be achieved instantaneously. While the solution feasibility is restricted by the joint-space limits that cannot be easily accounted for at the operational level. These limits become imperative when the desired steering orientation is outside the hardware limits. If the position limit has been reached, the unconstrained task maybe finding an infeasible solution even if the reachable solution exists. As a result, joint limits may delay/stop the controller reaction.

This situation is visualised at Fig. 8.2, which shows the unconstrained cartesian-space task in the middle, and the constrained joint-space solution on the right. In an initial position, the same in both cases, a part of the CoM safety boundary (cyan circle) lies outside of the support polygon (light red area). The unconstrained cartesian-space task (in the middle) finds the support polygon that encapsulates the CoM safety boundary with the smallest SPVs motion (purple area). However, one of the SPVs lies in the area not reachable by the robot due to the joint-space limits (yellow area). As a result, the robot only partially executes the desired support polygon reference, stopping on the edge of the feasible space (red area), and part of the safety boundary remains outside of the support polygon. If the SPVs constrained by the joint-space limits will move forward (red area on the right), the constrained solution can be fully executed by the robot (the purple and red areas overlap on the right side). At the expense of a bigger support polygons motion, a safety boundary lies fully inside the support polygon.

To overcome these pitfalls, (8.5) may be moved to the joint-space. However, while driving with restrictive camber angles as typical in wheeled robots, the support polygon can only be controlled along the wheel rolling direction (6.29). As a result, the control over the 2D ground space can only be achieved rotating the wheel, and so the rolling direction, around the ground normal beforehand. This is a nonlinear function with respect to the wheel steering position, while the wheel steering velocity does not affect the SPV motion. Thus, the standard quadratic optimization methods cannot be applied to control the system support polygon in the whole 2D ground space. To solve an optimization problem (8.5) in the whole 2D

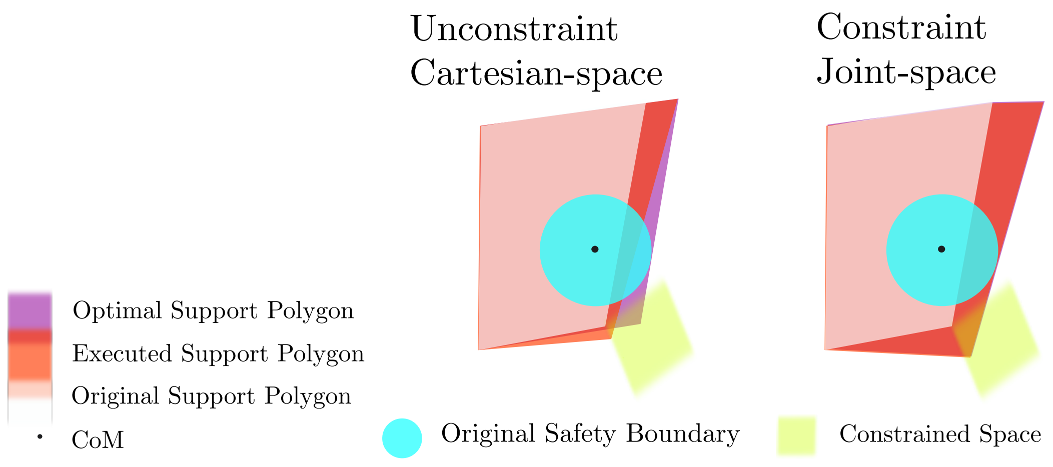


Figure 8.2 On the left: torus model of a wheel SPV. In the middle: cartesian-space solution infeasible due to the joint-space limits and the hardware response. On the right: feasible, constrained joint-space solution.

ground in the joint-space, (6.46) that depends on the wheel angular velocities/accelerations is expressed in terms of the wheel's angular velocities. Then (8.5) is moved to the joint-space, and proposed reactive control is integrated with the lower-level IK.

In a general legged-wheeled platform the camber angle is not constrained, and the mapping from components of the wheel angular velocity/acceleration vector to the joint-space is coupled and dependent on the robot state. In general mapping from the joint space to the wheel-space reads

$$\boldsymbol{\omega}_w = \mathbf{J}_\omega \dot{\boldsymbol{q}}, \; \dot{\boldsymbol{\omega}}_w = \mathbf{J}_\omega \ddot{\boldsymbol{q}} + \dot{\mathbf{J}}_\omega \dot{\boldsymbol{q}}, \tag{8.6}$$

where $\boldsymbol{\omega}_w^T = \begin{bmatrix} \boldsymbol{\omega}_{w,1}^T \cdots \boldsymbol{\omega}_{w,p}^T \end{bmatrix}$, $\mathbf{J}_\omega \in \Re^{3p\times n}$ denotes the wheel orientation Jacobian, a standard variable in the rigid-body kinematics. Expressing (3.42) as the joint-space optimisation problem with (6.46) and (8.6) is not trivial, as it imposes an optimisation for the velocity and acceleration of same variables. Furthermore, some joint-space constraints may affect the wheel orientation around the ground normal at the acceleration level. A lower-level control in Section 7.2 relies on the first order-kinematics that expects the SPV position/velocity as input. Since, the acceleration dynamics may be too fast for the underlying first-order IK, the linear SPV velocity model that can solve the non-holonomic constraint is developed from (6.46). To that end, the Euler integration method has been applied to the SPV acceleration and the wheel angular acceleration. It reads

$$\dot{\boldsymbol{x}}_{SPV,i} = \dot{\boldsymbol{x}}^{t-T}_{SPV,i} + \left(\mathbf{J}_{c,i} \begin{bmatrix} (\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T)\frac{\boldsymbol{\omega}_{w,i} - \boldsymbol{\omega}^{t-T}_{w,i}}{T} \\ \boldsymbol{n}\boldsymbol{n}^T \boldsymbol{\omega}_{w,i} \end{bmatrix} + \boldsymbol{A}_{c,i} \right) T,$$

where superscript $(.)^{t-T}$ symbolises a variable at the previous time step, and $\boldsymbol{n}\boldsymbol{n}^T$, $(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T)$ map to the ground normal space/null-space, respectively. Finally, the above equation in a linear form reads

$$\dot{\boldsymbol{x}}_{SPV,i} = \mathbf{J}_{SPV,i}\boldsymbol{\omega}_{w,i} + \boldsymbol{A}_{SPV,i}. \tag{8.7}$$

Similarly to (8.1), taking into account only the support polygon velocity in the ground normal null-space, and grouping (8.7) for all legs in ground contact, a linear equation can be defined.

$$\dot{\boldsymbol{x}}_{SPV} = \mathbf{J}_{SPV}\boldsymbol{\omega}_w + \boldsymbol{A}_{SPV}, \text{ where} \tag{8.8}$$

$\mathbf{J}^T_{SPV} = \begin{bmatrix} \mathbf{J}^T_{SPV,1|Pn} \cdots \mathbf{J}^T_{SPV,p|Pn} \end{bmatrix}$, $\boldsymbol{A}^T_{SPV} = \begin{bmatrix} \boldsymbol{A}^T_{SPV,1|Pn} \cdots \boldsymbol{A}^T_{SPV,p|Pn} \end{bmatrix}$, $\mathbf{J}_{SPV,i|Pn} \in \Re^{2\times n}$ stands for the projection of $\mathbf{J}_{SPV,i}$ to the ground-normal null-space, and $\boldsymbol{A}_{SPV,i|Pn} \in \Re^2$ denotes the projection of $\boldsymbol{A}_{SPV,i}$ to the ground-normal null-space.

8.3 Support Polygon Adaptation in the Joint-Space

In this section, (8.5) is expressed at the joint-space level with joint-space limits added as inequality constraints. Then, a set of constraints is introduced to ensure that the final solution is feasible to execute on the robot.

To move (8.5) to the joint-space, based on (8.6) and (8.8), a transformation $\dot{\boldsymbol{x}}_{SPV} = \mathbf{J}_{SPV}\mathbf{J}_{\omega}\dot{\boldsymbol{q}}$ is applied. As preliminary simulation results indicated that the dynamics of the offset term ($\boldsymbol{A}_{SPV}$) is too fast for an underlying first-order IK control, it has been neglected. The joint-space task reads

$$\begin{aligned} \underset{\dot{\boldsymbol{q}}}{\text{minimize}} \quad & \dot{\boldsymbol{q}}^T \boldsymbol{Q} \dot{\boldsymbol{q}} + \boldsymbol{\delta}^T \boldsymbol{W} \boldsymbol{\delta} \\ & \boldsymbol{Q} = \boldsymbol{J}^T_{\omega}(\mathbf{J}^T_{SPV}\mathbf{J}_{SPV} + \boldsymbol{\rho}_{SPV})\mathbf{J}_{\omega} + \boldsymbol{\rho}_{\omega} \\ \text{subject to} \quad & \mathbf{J}_M \dot{\boldsymbol{q}} \geq \Delta \boldsymbol{m}_{min} - \boldsymbol{\delta}_m \\ & \mathbf{J}_W \dot{\boldsymbol{q}} \leq \Delta \boldsymbol{w}_{max} + \boldsymbol{\delta}_w \\ & \boldsymbol{\delta}_w \geq \mathbf{0}, \quad \boldsymbol{\delta}_m \geq \mathbf{0}, \end{aligned} \tag{8.9}$$

where $\boldsymbol{\rho}_{cp} \in \Re^{3p\times 3p}$ and $\boldsymbol{\rho}_{\omega} \in \Re^{n\times n}$ are the diagonal damping matrices to prevent the rank degeneration in the cost matrix $\boldsymbol{Q}$, when $\mathbf{J}_{SPV}$ or $\mathbf{J}_{\omega}$ is singular. The SPSM constraint maps from the operational-space to the joint-space by introducing $\dot{\boldsymbol{x}}_{SPV} = \mathbf{J}_{SPV}\mathbf{J}_{\omega}\dot{\boldsymbol{q}}$ to (8.2); the SPSM Jacobian reads

$$\mathbf{J}_M = \mathbf{J}_m^{SPV}\mathbf{J}_{SPV}\mathbf{J}_{\omega} + \mathbf{J}_m^{CoM}\mathbf{J}_{CoM}, \tag{8.10}$$

where $\mathbf{J}_{CoM} \in \Re^{3\times n}$ denotes robot's CoM Jacobian. Likewise, the workspace Jacobian from (8.4) in joint-space reads

$$\mathbf{J}_W = \mathbf{J}_w^{SPV}\mathbf{J}_{SPV}\mathbf{J}_{\omega} + \mathbf{J}_w^{h}\mathbf{J}_h, \tag{8.11}$$

where $\mathbf{J}_h \in \Re^{3p\times n}$ maps $\dot{\boldsymbol{x}}_h$ to the joint-space. The rigid-body kinematics directly provides $\mathbf{J}_{CoM}$ and $\mathbf{J}_h$.

Furthermore, the joint-space constraints have been added to (8.9) to account for the robot hardware limits

$$\begin{aligned} &\max(\dot{\boldsymbol{q}}_{min}, \Delta\boldsymbol{q}_{min}) \leq \dot{\boldsymbol{q}} \leq \min(\dot{\boldsymbol{q}}_{max}, \Delta\boldsymbol{q}_{max}), \\ &\boldsymbol{\tau}_{min}T + \boldsymbol{\tau}_b \leq \boldsymbol{M}\dot{\boldsymbol{q}} \leq \boldsymbol{\tau}_{max}T + \boldsymbol{\tau}_b, \end{aligned} \tag{8.12}$$

where $\boldsymbol{\tau}_b = -\boldsymbol{F}_b T + \boldsymbol{M}\dot{\boldsymbol{q}}^{t-T}$, and $\boldsymbol{F}_b \in \Re^n$ stands for the bias force that includes gravity, coriolis, centrifugal and ground reaction forces (3.22). Subscripts $(.)_{min/max}$ denote minimum/maximum position ($\boldsymbol{q}$), velocity ($\dot{\boldsymbol{q}}$) and torque ($\boldsymbol{\tau}$) joint-space limits.

In the joint-space, additional constraints are needed to ensure the wheel rolling motion on the ground plane. It includes the non-sliding pure rolling condition and the constraint on the camber angle. Furthermore, the robot world posture has been constrained to account for the motion of the robot base and the upper-body. In this work, the robot motion at the current time step has been used as an approximation for the next step motion. These constraints read

$$\begin{aligned} &\mathbf{0} = \mathbf{J}_{cp}\dot{\boldsymbol{q}}, \quad {}^{o}\dot{\boldsymbol{x}}_r^{t-T} = \mathbf{J}_r\dot{\boldsymbol{q}}, \\ &\dot{\boldsymbol{\varphi}}^{t-T} - \varepsilon \leq \mathbf{J}_{\varphi}\dot{\boldsymbol{q}} \leq \dot{\boldsymbol{\varphi}}^{t-T} + \varepsilon, \end{aligned} \tag{8.13}$$

where $\mathbf{J}_{cp} \in \Re^{3p\times n}$ denotes the Jacobian from the world origin to the wheel-ground contact points; it represents the Jacobian of the robot non-sliding, pure-rolling constraints (Section 3.2.2). The robot world posture is marked with ${}^{o}\dot{\boldsymbol{x}}_r \in \Re^6$, and $\mathbf{J}_r \in \Re^{6\times n}$ denotes the corresponding Jacobian as described in Section 7.2.2. Wheels camber angles state and Jacobian are denoted by $\boldsymbol{\varphi} \in \Re^p$ and $\mathbf{J}_{\varphi} \in \Re^{p\times n}$, respectively, see Section 7.2.3 for details. To extend the search space of the feasible support polygons, a small tolerance ($\varepsilon = 1e^{-5} rad/s$) on the camber angle task has been adopted.

Robot legs are assumed massless, in (8.5). However, for the CENTAURO robot each leg constitute over 11% of its mass, and thus, the leg position considerably affects the system CoM. To account for this effect, in (8.9), the hip and CoM motions in (8.2) and (8.4) are considered directly in (8.10), (8.11). As a result, both approaches discussed in Section 8.1 to balance the robot are implemented in (8.9). Furthermore, (8.13) provides an approximation for the next step CoM motion through the constraint on the robot world posture. As this motion is considered in the optimisation process, the task (8.9) tries to see one step ahead, when the constraints will be violated, and to react beforehand.

8.3.1 Integration with the Lower-Level Inverse Kinematics

Here, integration of (8.9) with the lower-level IK scheme introduced in Section 7.2.2 is outlined. After an optimal solution for (8.9)-(8.13) has been found, desired SPVs are computed, and a component along the wheel rolling motion is applied into the first-order IK control Section 7.2.2. It reads

$$\boldsymbol{x}_{SPV,\mathrm{des}} = \boldsymbol{x}_{SPV,\mathrm{des}}^{t-T} + \boldsymbol{x}_s \boldsymbol{x}_s^T \mathbf{J}_{SPV} \mathbf{J}_{\omega} \dot{\boldsymbol{q}}_{qp} T, \tag{8.14}$$

where $\dot{\boldsymbol{q}}_{qp} \in \Re^n$ stands for the solution from (8.9)-(8.13). The vector of desired wheel orientations around the ground normal ($\boldsymbol{\beta}_{\mathrm{des}} \in \Re^p$) is computed from

$$\boldsymbol{\beta}_{\mathrm{des}} = \boldsymbol{\beta}_{\mathrm{des}}^{t-T} + \mathbf{J}_{\beta} \dot{\boldsymbol{q}}_{qp} T, \tag{8.15}$$

where $\mathbf{J}_{\beta} \in \Re^{p \times n}$ represents the Jacobian for derivative of (7.13). Fig. 8.3 outlines a information flow in the system and the integration with the IK scheme proposed in Section 7.2.

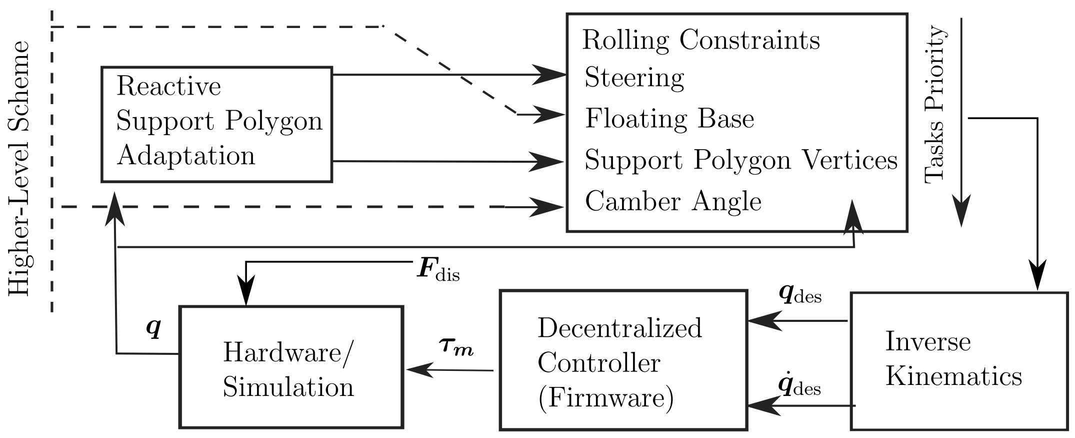


Figure 8.3 Block diagram of the reactive support polygon adaptation scheme.

$\dot{\boldsymbol{q}}_{max/min}$	± 1 rad/s	T	2 ms
$\boldsymbol{\tau}_{max/min}$		$\boldsymbol{\rho}_{SPV}$	10^{-6}
ankle yaw, wheel	± 23 Nm	$\boldsymbol{\rho}_{\omega}$	10^{-7}
ankle pitch	± 98 Nm	w_{max}	0.65 m
other	± 200 Nm	$\boldsymbol{W}$ – SPSM	50
side SPSM: A/B & C/D	0.215/0.27/0.2 m	$\boldsymbol{W}$ – workspace	
front SPSM: A/B & C/D	0.40/0.45/0.45 m	A	80
back SPSM: A/B & C/D	0.40/0.60/0.45 m	B, C, D	100

Table 8.1 Experimental parameters.

8.4 Experimental Results

The proposed scheme has been verified in four experiments with the CENTAURO robot. The experiment ’A’ has been designed to test the controller long-term response and to check the validity of the generated support polygons. Experiments ’B’ and ’C’ tested the controller response to unknown motion/push disturbance, and the experiments ’D’ checked the controller response to a disturbance generated by an unknown weight added to the robot arms.

All joint’s velocities have been limited to 1 rad/s, and torques have been limited to 23 Nm for ankle yaw and wheel, 98 Nm for ankle pitch, and 200 Nm for the remaining joints. The controller was running in the real-time with the 2 ms time step[3]. Diagonal elements in $\boldsymbol{\rho}_{SPV}$ are equal 10^{-6} and 10^{-7} in $\boldsymbol{\rho}_{\omega}$. These are the smallest values that prevented the matrix $\boldsymbol{Q}$ from degenerating. The cost gain in $\boldsymbol{W}$ was 50 for the safety margin. For the workspace limits, the cost was set up to 80 in the experiment ’A’; it was then increased to 100 for the remaining experiments. In all experiments, the maximum virtual leg length has been set-up to 0.65 m, and thus in the squared form (8.3) the workspace soft limits, for all legs, read 0.65^2 m^2 = 0.4225 m^2. An offset from the fully extended CENTAURO robot leg (0.81 m) has been adopted to account for the camber angle/rolling motion constraints, and to provide the controller with sufficient time and space to react before the physical limits are reached. An overview of the experimental parameters is given in Table 8.1.

[3]on COM Express conga-TS170 embedded computer with Intel Core i7-6820EQ CPU; lower-level middleware required free 0.5 ms.

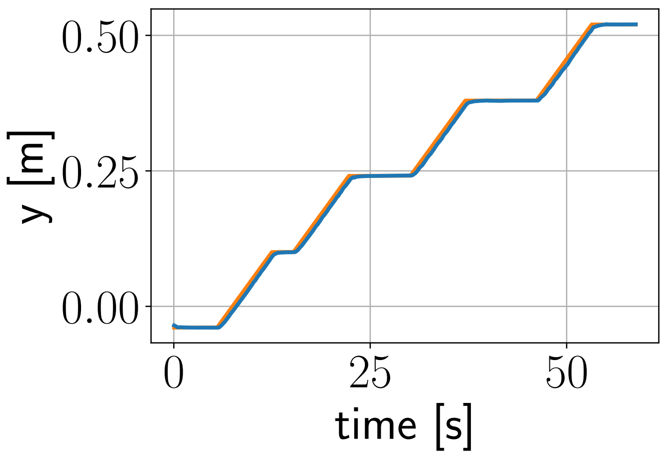


Figure 8.4 Controlled base motion experiment: desired/executed base motion.

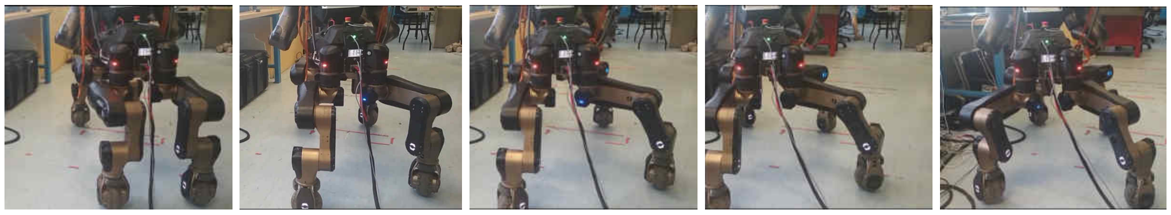


Figure 8.5 Snap-shots from the experiment with the controlled base motion.

8.4.1 Experiment with Controlled Base Motion

In the experiment 'A', the robot base preformed a set of controlled lateral motions with an operator sending constant velocity references of 0.02 m/s for $\sim$7 s directly to the floating base regulation task in the lower-level IK (Fig. 8.3); the position reference and executed motion are shown at Fig. 8.4. The support polygon reference has been designed independently through (8.9)–(8.13) based on the system feedback and without any information about the expected base motion. Fig. 8.5 shows a few snap-shots from the experiment.[4]

The safety margins have been set-up to 0.215 m for the two support polygon edges described by the left/right robot legs. These support polygon edges are referred to as 'side' edges in this work. The safety margins for the support polygon edges described by the front and rear robot legs have been set-up to 0.40 m. These support polygon edges are referred to as the 'front' and 'back' edge, respectively. These safety margins have been set-up based on the robot kinematics, controller dynamics, confidence in the robot model, and to reduce the risk of self-collisions.

[4]A video recording from this experiment can be seen at https://youtu.be/vUyrIt6A8_I

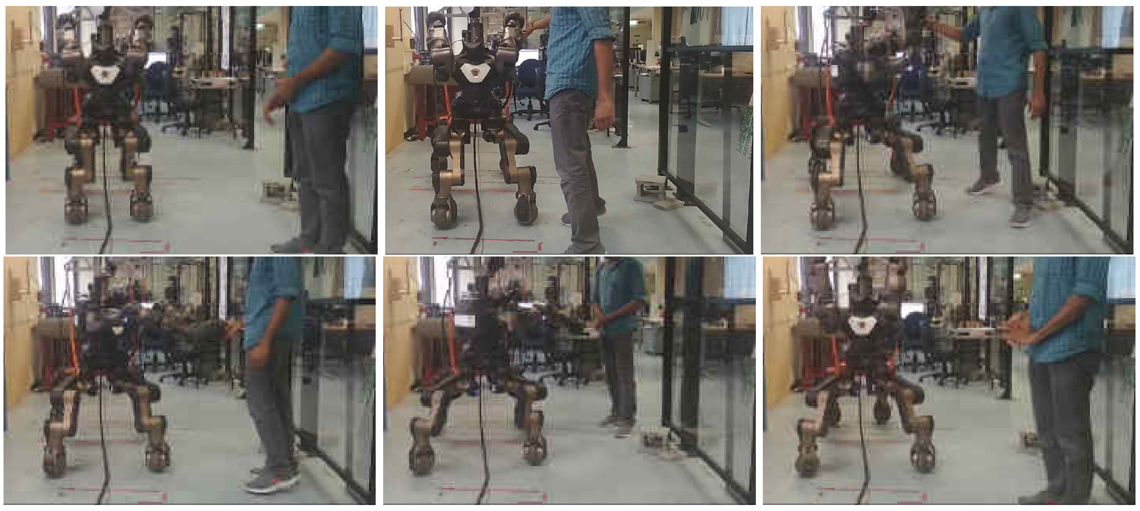


Figure 8.6 Snap-shots from the experiment with the CoM modulation using the upper-body.

8.4.2 Experiment with COM modulation using the Upper-Body

In the experiment ’B’, a human operator back-drove the robot upper-body affecting its CoM. The system adapted its support polygon to the unknown motion so that the designed safety margins have been maintained.[5]

The proposed controller adjusts the support polygon when the violation of the safety margins/workspace limits has been detected. At the end of the disturbance/motion, when the robot base retrieves the original state, the system holds on to the new, adjusted support polygon. It does not restore the original pose. As the upper-body has limited capability to modify the robot CoM, a position that satisfies the safety margin and workspace limits regardless of the upper-body configuration can be found quickly. To test the controller in more challenging conditions, more conservative safety margins of 0.27/0.45/0.60 m for the side/front/back support polygon edges have been applied in this experiment.

8.4.3 Experiment with Unknown Push Disturbance

In the ’C’ experiment, the robot has been controlled in the impedance mode with feed-forward online gravity compensation.[6] The medium joint stiffness gains of 350-800 Nm/rad and damping of 15-20 Nms/rad have been used. A set of unknown push disturbances has been applied to the robot moving its CoM relative to the support polygon. The same restrictive

[5]A video recording from this experiment can be seen at https://youtu.be/ZQzOd0xhtgc
[6]A video recording from this experiment can be seen at https://youtu.be/ib06UKrcEsw

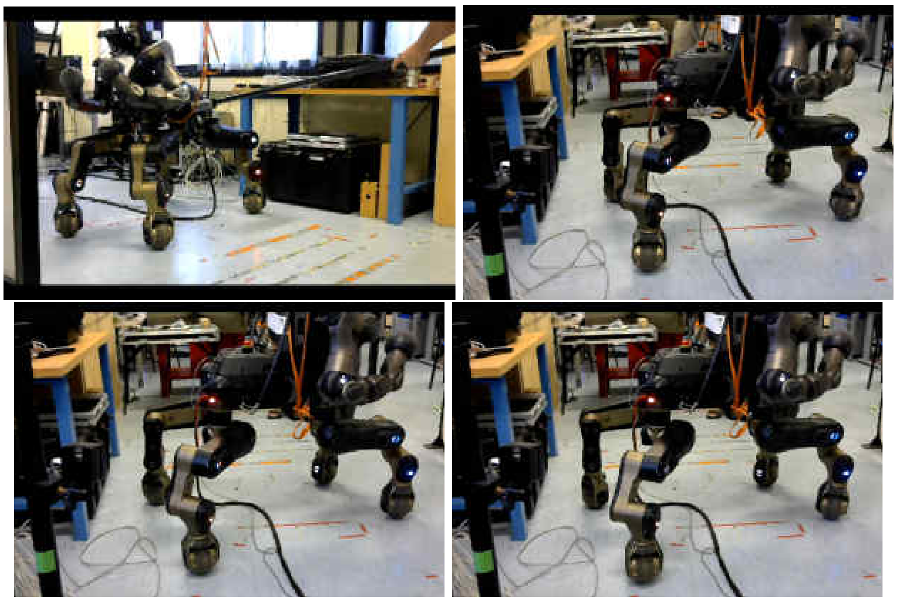


Figure 8.7 Snap-shots from the experiment with the unknown push disturbance applied to the system with medium stiffness impedance control.

safety margins as in the 'B' experiment has been applied to test the controller reaction with disturbance of different magnitude/direction.

8.4.4 Centre of Pressure Experiment

The last two experiments were performed using the centre of pressure (CoP) in place of the CoM as a measure to compute the SPSMs (8.1). The CoP is characterised by faster dynamics and higher sensitivity to the modelling errors than the CoM . To avoid support polygon adaptation on the controller initialisation triggered when the conservative safety margins (experiments B and C) are used, new safety margins of 0.20/0.45/0.45 m for the side/front/back support polygon edges have been chosen.

In the experiment 'D', the weights of an increasing mass of 2, 4, 6 and 8 kg (from $\sim$2 to 8% of the robot mass) were added to the robot arms.[7] Fig. 8.9 on the left the difference between the estimated CoM and CoP is shown to visualise the effect of these masses. In this experiment, each mass imposed a small support polygon modification. To generate a bigger support polygon adjustment in response to the CoP motion, an additional experiment has been performed with the the 10 kg mass added directly to the unloaded end-effector.[8]

[7]A video recording from this experiment can be seen at https://youtu.be/KHpB7Rrzjh0
[8]A video recording from this experiment can be seen at https://youtu.be/z5eEbWc-0YY

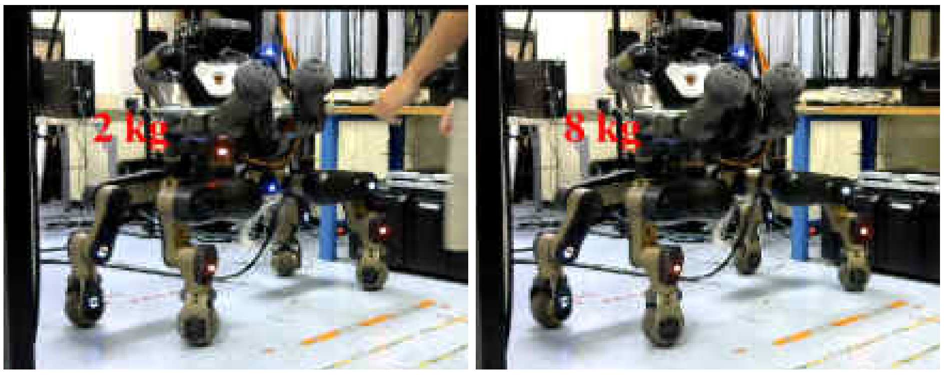


Figure 8.8 Snap-shots from the experiment with the added weights and CoP used to measure the system balance.

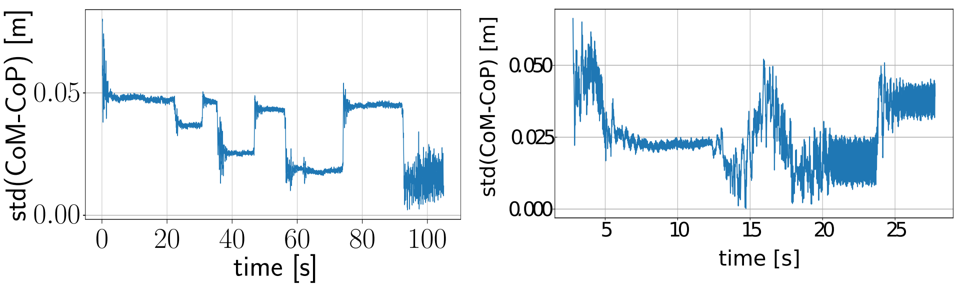


Figure 8.9 CoP tracking experiment: difference between the CoM and CoP. The incremental mass change on the left, and the 10 kg mass on the right.

Fig. 8.9 on the right the difference between the estimated CoM and CoP is shown to visualise the effect of these masses.

8.4.5 Results and Discussion

The proposed controller modulates the support polygons to ensure the desired safety margins while respecting the robot workspace/joint limits even for long-lasting disturbances. In all experiments, the SPSMs and virtual leg lengths overall stay within the designed limits, while some temporary violations occurred during the motion. These can be seen in the results in the SPSMs and adopted virtual leg lengths shown at Fig. 8.10 for the A experiment, at Fig. 8.11 for the B experiment, at Fig. 8.12 for the C experiment, and at Fig. 8.13 and Fig. 8.14 for the incremental and 10 kg CoP experiments, respectively. In these figures the SPSMs are expected to stay above and virtual legs lengths below the limits. These small violations of the soft constraints are expected as due to the control design and to avoid instability sufficiently large offsets to the physical limits have to be applied.

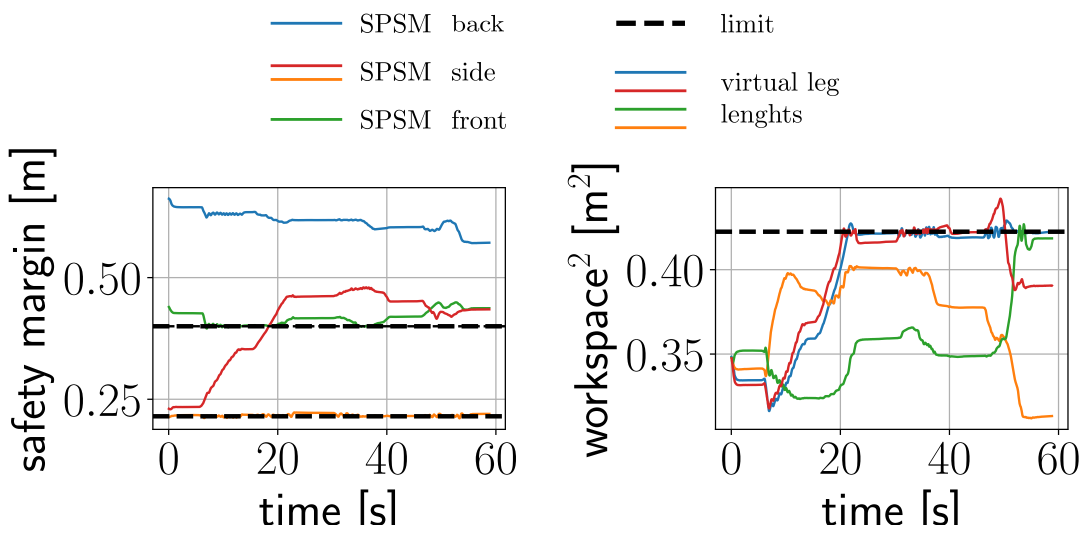


Figure 8.10 Experimental results for the controlled base motion. On the left: Adopted SPSMs for front (green line), back (blue), and side (yellow/red) support polygon edges with the designed safety margins (dashed black) for the side edges 0.215 m, and the front/back edge 0.40/0.40 m. On the right: Adopted virtual leg lengths (solid lines), and their limit of 0.4225 m^2 (dashed black) plotted in the squared form (8.3).

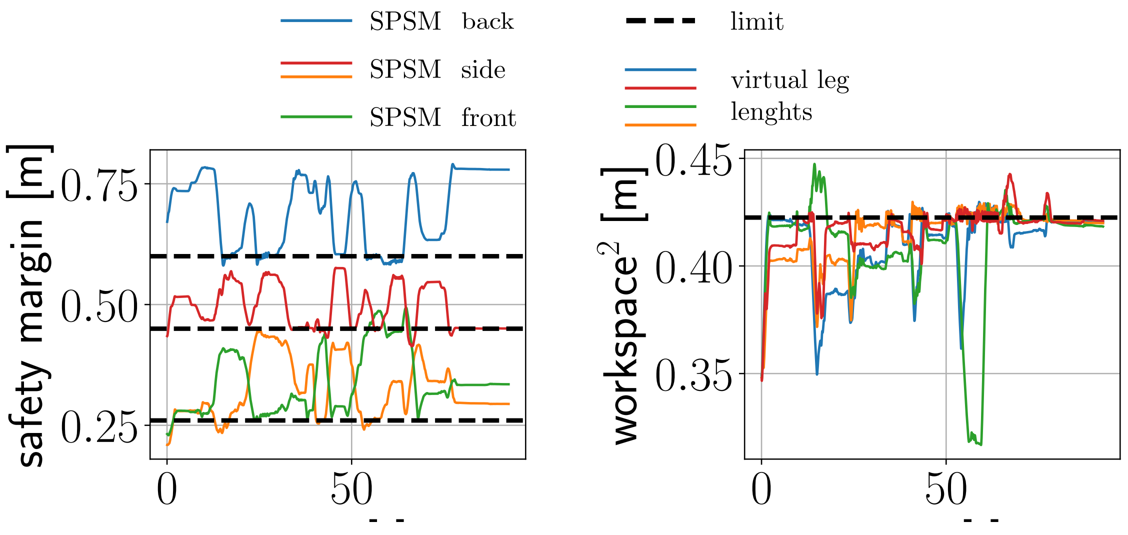


Figure 8.11 Experimental results for the back-driven upper-body. On the left: Adopted SPSMs for front (green line), back (blue), and side (yellow/red) support polygon edges with the designed safety margins (dashed black) for the side edges 0.2 m, and the front/back edge 0.45/0.45 m. On the right: Adopted virtual leg lengths (solid lines), and their limit of 0.4225 m^2 (dashed black) plotted in the squared form (8.3).

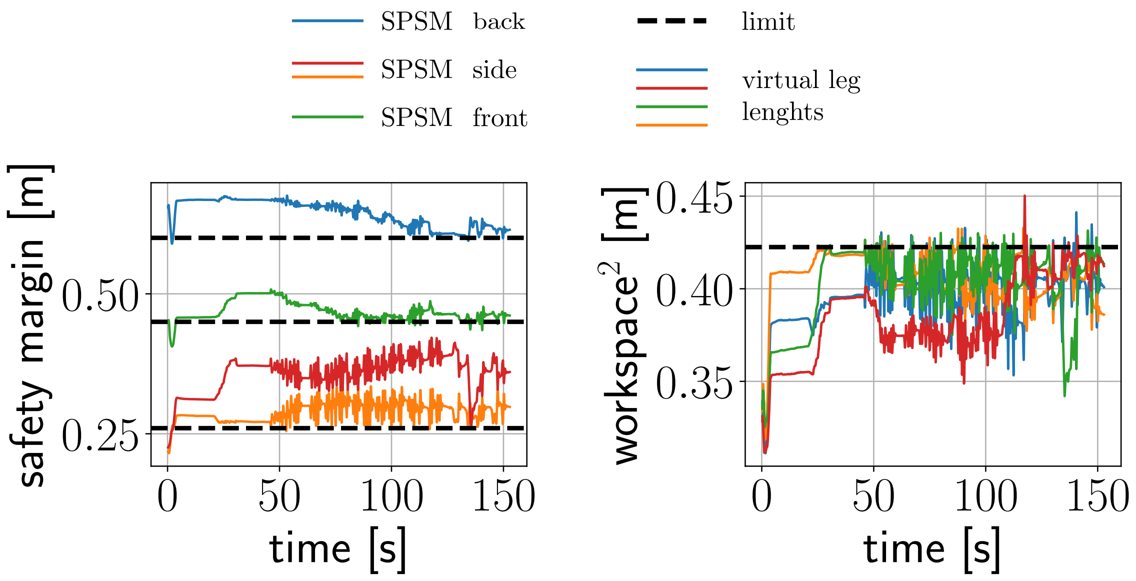


Figure 8.12 Experimental results for the unknown push disturbance. On the left: Adopted SPSMs for front (green line), back (blue), and side (yellow/red) support polygon edges with the designed safety margins (dashed black) for the side edges 0.27 m, and the front/back edge 0.45/0.60 m. On the right: Adopted virtual leg lengths (solid lines), and their limit of 0.4225 m^2 (dashed black) plotted in the squared form (8.3).

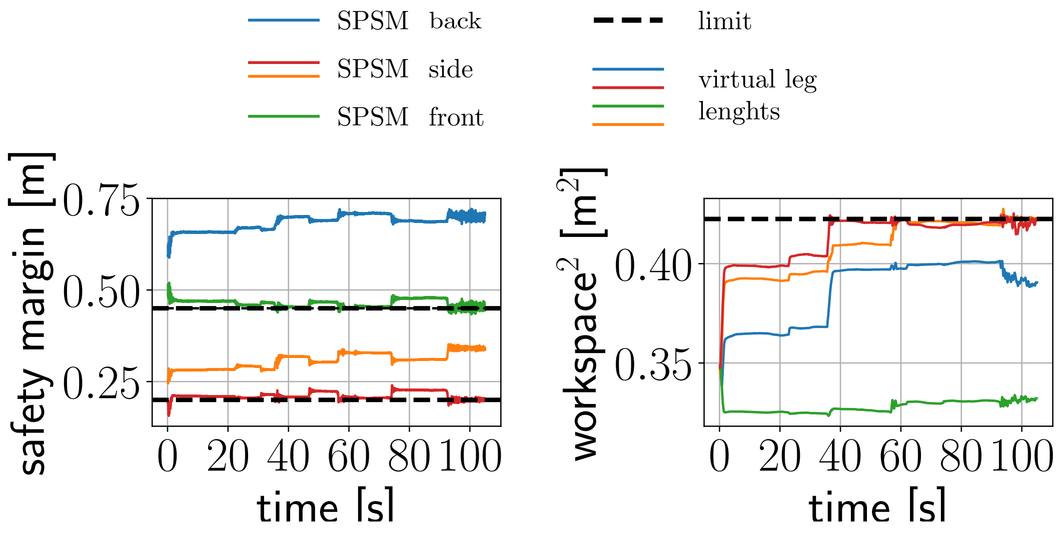


Figure 8.13 Experimental results for the CoP tracking with incremental mass change. On the left: Adopted SPSMs for front (green line), back (blue), and side (yellow/red) support polygon edges with the designed safety margins (dashed black) for the side edges 0.27 m), and the front/back edge 0.45/0.60 m. On the right: Adopted virtual leg lengths (solid lines), and their limit of 0.4225 m^2 (dashed black) plotted in the squared form (8.3).

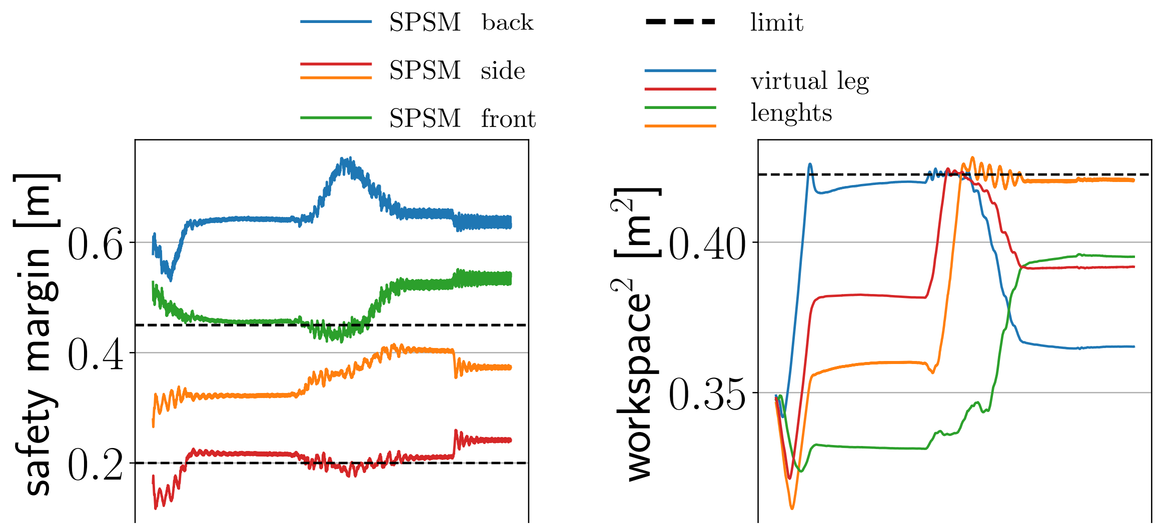


Figure 8.14 Experimental results for the CoP tracking with 10 kg mass. On the left: Adopted SPSMs for front (green line), back (blue), and side (yellow/red) support polygon edges with the designed safety margins (dashed black) for the side edges 0.27 m), and the front/back edge 0.45/0.60 m. On the right: Adopted virtual leg lengths (solid lines), and their limit of 0.4225 m^2 (dashed black) plotted in the squared form (8.3).

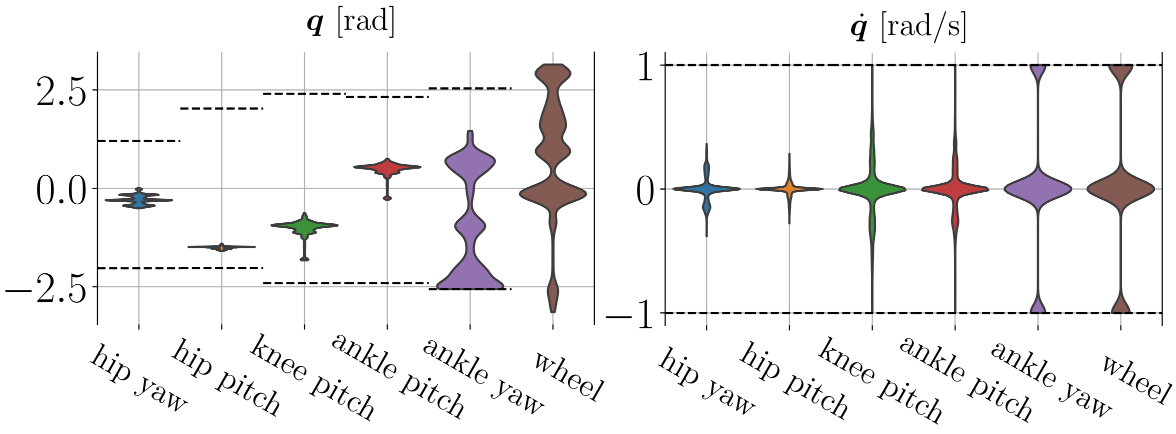


Figure 8.15 The upper-body modulation experiment – front left leg. Kernel density estimate for the joint-space position/velocity on the left/right limited by the minimum/maximum registered value. Lower/upper limits are marked with the dashed black lines; no wheel position limit.

Since the CoM motion is modelled in the joint-space task, the controller tries to see one step ahead when the safety margins will be violated. It then attempts to react just before it would have happened. However, with the unpredictable nature of the disturbance, the constraints may be violated. If the violation arises due to the disturbance being too fast for the assumed velocity/torque limits to satisfy the control requirements within a time step, the support polygon will be pushed towards the more stable position up to the designed joint-space velocity/torque limits. With adopted conservative velocity limits of 1 rad/s, unless a minor disturbance occurred, the optimisation stops on the limits. At Fig. 8.15 the kernel density estimates for the front left leg joint-space DoFs have been shown for the experiment with the back-driven upper-body. The estimates have been cut-off on the minimum/maximum registered value to visualise that the inequality constraints are respected. Fig. 8.15 shows that the ankle yaw and wheel velocity limits are frequently reached during the experiment as well as the ankle yaw position limit.

With the conservative safety margins applied in experiments 'B' and 'C', the controller frequently operates near the designed maximum virtual leg length (Fig. 8.11 and Fig. 8.12). If within the designed joint-space limits there exists no SPVs arrangement that satisfies both the safety margin and workspace limits, the system looks for a solution with the minimal cost of the slack variables (i.e., violation of the soft constraints); e.g., the workspace limit is violated for a few seconds in the experiment A (red line, Fig. 8.10, top right, $\sim$50 s) due to the ankle yaw position limit. In this case, any CoM motion can trigger the change to the new SP that optimises the violation of the soft constraints.

Variation of around 0.3-1 Hz in the results for the C experiment (Fig. 8.12) indicate the disturbance applied to the robot. These disturbances modified the robot CoM and the base position relative to the SPVs, and thus changed the virtual leg length and safety margins. If the same/similar disturbance is applied to the robot consecutively, the controller reaction is triggered only at the first application. In response, a balanced position is adopted and no adjustment is executed after the disturbance is removed. However, a reaction may be triggered if a disturbance of different magnitude/direction is applied. The corresponding x and y-coordinates of the SPVs are given at Fig. 8.16 to show that no SPV motion is triggered until a soft inequality limit is violated.

8.5 Conclusion

In this chapter, a reactive control scheme has been proposed to improve the stability of the CENTAURO robot with the introduction of an active support polygon adaptation task. To

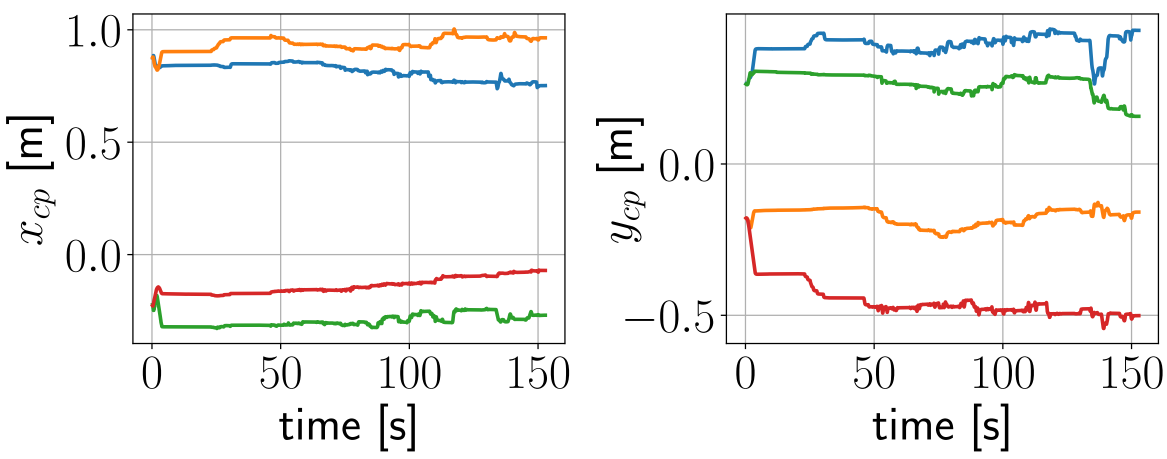


Figure 8.16 SPV x/y-coordinates (left/right side) – unknown push disturbance experiment: front left/right (blue/yellow) and rear left/right (green/red) legs.

take into account the joint-space hardware limits the mixed velocity/acceleration model for the support polygon acceleration developed in Section 6.4 has been used. While the proposed joint-space controller can be applied to any hybrid legged-wheeled robot, its significance arises with the platforms permitting the wheel rotation along the ground normal.

The proposed scheme was implemented and verified on the CENTAURO robot under disturbances of various duration and magnitude. It has been shown that the support polygon that respects the designed safety margins and workspace limits is effectively generated and executed on the robot. Even if no solution exists within the assumed joint limits, the proposed control method designs the SP that improves the robot balance within the robot motion space.

The controller proposed in this chapter is a first step towards the reactive support polygon adaptation in the CENTAURO robot permitting the continuous support polygon motion on the whole ground space. The proposed controller can be directly used when the robot is operating on 3 legs, and when some of the robot legs have lost contact with the ground due to the momentum introduced by the push disturbance. In the latter case, the robot would adopt the SP that ensures a balanced pose after regaining the contact with the ground. However, the controller would not push the robot leg towards the ground to restore the point of contact. If the robot operates on two legs, a more dynamic balancing approaches will be definitely needed.

As a further work, one could implement the proposed scheme at the second-order IK level to improve the controller dynamics. The short-term predictive planning approach could be used to improve the controller search space, avoid local minima and to incorporate stepping. On the other hand, trajectory planning methods require more computation time, which would decrease the controller responsiveness. Moreover, optimisation of the SPVs over a time horizon introduces a non-linearity, while consideration of the steering limits is not trivial in the operational-space. Finally, the practicality of the proposed scheme would be improved with integration of the locomotion scheme. While the desired base motion can be considered directly with the addition of the desired SPV motion into the cost function, a more challenging task would be to restore the original higher-level position, and to decide when such action should initiate.

Chapter 9

Conclusion

In this thesis, the kinematics and control of the hybrid legged-wheeled robots has been studied, and the locomotion framework for the CENTAURO robot has been developed and experimentally verified. The main contributions consit of the novel model for the support polygon vertex of the hybrid legged-wheeled robots, the robust omnidirectional driving scheme, and a reactive support polygon adaptation control scheme. To test the designed controllers, the CENTAURO robot simulator and the software framework for the locomotion research have been developed.

In Chapter 6, novel support polygon vertex model for a general hybrid legged-wheeled robot has been developed and computed at the velocity and acceleration levels without any assumptions on the robot kinematic arrangement or a wheel camber angle. The proposed model describes the support polygon of a general HLW platform as a function of the wheel angular velocities, and thus the robot kinematic constraints are expressed by a set of wheel angular velocities the robot can exert. The proposed model provides a new inside to the mobility of the hybrid legged-wheeled robots, and it has been used to study mobility of the HLW platforms.

Based on the analysis of the developed support polygon model, an omnidirectional driving scheme has been designed in Chapter 7. A continuous wheel motion has been resolved through the IK scheme, which generates robot motion compliant with the NSPR condition. It tracks world posture of the robot, its SPVs, camber and steering angles. A higher-level scheme resolving a steering motion to comply with the non-holonomic constraint has been proposed, and the damping scheme has been developed to tackle the structural singularity.

The proposed control scheme has been implemented and verified on the CENTAURO robot in the set of experiments designed to evaluate motion control scheme robustness to a non-smooth base trajectory, its performance near the structural singularity, and its ability to change a support polygon type during the whole-body motion. Presented experimental results show that the proposed control scheme produces a stable motion and correct tracking of a given world posture reference and the SPVs positions. The system remains stable in presence of a non-smooth reference, and it generates continuous steering reference near the structural singularity. Furthermore, presented results verify the controller ability to support 3 and 4 leg configurations, as well as its capability to modify a support polygon configuration during the whole-body motion of the robot.

To improve the CENTAURO robot response to the unpredicted disturbances, an inverse kinematics scheme developed in Chapter 7 has been extend in Chapter 8 with a new reactive control scheme. The developed support polygon adaptation control takes advantage of the CENTAURO robot six DoFs legged-wheeled structure that allows for the continuous support polygon regulation in the entire 2-D space of the ground plane. To this end, a novel quadratic programming task has been designed to push the system SPVs away from the robot CoM, while respecting the leg workspace limits. To take into account the hardware limits the mixed velocity/acceleration model for the support polygon acceleration developed in Chapter 6 has been used to express the proposed task in the joint-space. While the proposed joint-space controller can be applied to any hybrid legged-wheeled robot, its significance arises with the platforms permitting the wheel rotation along the ground normal.

The proposed scheme has been tested on the CENTAURO robot under disturbances of various duration and magnitude. It has been shown that the support polygon that respects the designed safety margins and workspace limits is effectively generated and executed on the robot. Even if no solution within the assumed joint limits exists, the proposed control generates the support polygon that improves the robot stability within the designed robot motion space.

The experiments presented in this thesis has been performed with the Simplifying Operations in Locomotion - Framework for Efficient Research (SOL-FER), a software for the locomotion research introduced in Chapter 5. SOL-FER has been designed to provide a way to quickly try out ideas by maximizing the flexibility and reconfigurability of the developed modules. Finally, the simulator framework for the SEA actuated CENTAURO robot that models the actuation dynamics has been developed with the ROS middleware and Gazebo simulator as presented in Chapter 4.

The further work could consist of the more challenging experiments for the proposed omnidirectional driving scheme to verify its performance in the scenarios that more closely resemble the real-world applications; e.g., uneven and more compliant grounds. To that end, the estimator for the ground normal would be required. Furthermore, an integration of the proposed reactive control scheme with the omnidirectional driving scheme would greatly improve practicality of the proposed framework, and integration of the algorithm to restore the original higher-level reference after the disturbance has finished would provide a valuable contribution to the control framework presented in this thesis.

The proposed models and controllers as well as the CENTAURO robot simulation framework have been presented in the published/submitted publications as given in the remainder of this chapter. Furthermore, list of the attended PhD courses have been attached at the end of this chapter. All of the presented work has been a part of the European Union’s Horizon 2020 research and innovation programme under grant agreement No 644839 (CENTAURO).

Publications

[1] **Malgorzata Kamedula** and Nikos G Tsagarakis. “Reactive Support Polygon Adaptation for the Hybrid Legged-Wheeled CENTAURO robot”. In: *IEEE Robotics and Automation Letters* (2020 (accepted)).

[2] **Malgorzata Kamedula**, Navvab Kashiri, and Nikos G Tsagarakis. “Wheeled Motion Kinematics and Control of a Hybrid Mobility CENTAURO robot”. In: *Robotics and Autonomous Systems* (2019 (in revision)).

[3] Navvab Kashiri, Lorenzo Baccelliere, Luca Muratore, Arturo Laurenzi, Zeyu Ren, Enrico Mingo Hoffman, **Malgorzata Kamedula**, et al. “CENTAURO: A Hybrid Locomotion and High Power Resilient Manipulation Platform”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1595–1602.

[4] **Malgorzata Kamedula**, Navvab Kashiri, and Nikos G Tsagarakis. “On the kinematics of wheeled motion control of a hybrid wheeled-legged centauro robot”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2018, pp. 2426–2433.

[5] Lorenzo Baccelliere, Navvab Kashiri, Luca Muratore, Arturo Laurenzi, **Malgorzata Kamedula**, Alessio Margan, Stefano Cordasco, et al. “Development of a human size and strength compliant bi-manual platform for realistic heavy manipulation tasks”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2017, pp. 5594–5601.

[6] **Malgorzata Kamedula**, Navvab Kashiri, Darwin G Caldwell, and Nikos G Tsagarakis. “A Compliant Actuation Dynamics Gazebo-ROS Plugin for Effective Simulation of Soft Robotics Systems: Application to CENTAURO Robot.” In: *International Conference on Informatics in Control, Automation and Robotics*. 2016, pp. 485–491.

Training Related to the PhD Programme

- **An Introduction to Spatial (6D) Vectors and Their Use in Robot Dynamics**

Instructor(s):	Roy Featherstone
When/where:	March 2017, IIT, Via Morego 30, Bolzaneto, Genova, Italy
Credits awarded:	5
Final grade/approval:	Passed

- **An introduction to dynamic optimization and optimal control: models, solutions, and approximations**

Instructor(s):	Giorgio Gnecco
When/where:	March, 2017, DIBRIS, Via Dodecaneso 35, Genova, Italy
Credits awarded:	6
Final grade/approval:	Passed

- **Italian course for PhD students**

Instructor(s):	Elena Firpo
When/where:	February–May 2017, DAFIST, Via Balbi 2, Genova, Italy
Credits awarded:	2
Final grade/approval:	Passed

- **Human-Robot Interaction**

Instructor(s):	Francesco Rea, Alessandra Sciutti
When/where:	Oct 2017, IIT, via Morego 30, Genova, Italy
Credits awarded:	5
Final grade/approval:	Passed

- **Computational Robot Dynamics**

Instructor(s):	Roy Featherstone
When/where:	Mar 2018, IIT, via Morego 30, Genova, Italy
Credits awarded:	3
Final grade/approval:	Passed

- **Science Writing and Presentation Skills**

Instructor(s):	Alberto Diaspro, Antonio Sgorbissa
When/where:	Mar - May 2018, DIBRIS, Villa Bonino, Genova, Italy
Credits awarded:	4
Final grade/approval:	Passed

- **Data Acquisition and Data Analysis Methods**

Instructor(s):	Carlo Canali, Alessandro Pistone
When/where:	Apr-May 2018, IIT, via Morego 30, Genova, Italy
Credits awarded:	5
Final grade/approval:	Passed

- **Ethical, Bioethical and Legal Issues in Bioengineering and Robotics**

Instructor(s):	Linda Battistuzzi, Valentina Di Gregorio
When/where:	February 20 –March 8 2019, DIBRIS, Villa Bonino, Genova, Italy
Credits awarded:	5
Final grade/approval:	Passed

- **Robot programming with ROS**

Instructor(s):	Carmine Tommaso Recchiuto,
When/where:	June 5th – June 26th 2019, DIBRIS, Via All’Opera Pia 13, Genova, Italy
Credits awarded:	5
Final grade/approval:	Passed

References

[1] Navvab Kashiri et al. “CENTAURO: A Hybrid Locomotion and High Power Resilient Manipulation Platform”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1595–1602.

[2] Paul Hebert et al. “Mobile manipulation and mobility as manipulation—design and algorithms of RoboSimian”. In: *Journal of Field Robotics* 32.2 (2015). 00025, pp. 255–274. (Visited on 05/25/2016).

[3] Marko Bjelonic et al. “Keep Rollin’-Whole-Body Motion Control and Planning for Wheeled Quadrupedal Robots”. In: *IEEE Robotics and Automation Letters* (2019).

[4] Max Schwarz et al. “NimbRo Rescue: Solving disaster-response tasks with the mobile manipulation robot Momaro”. In: *Journal of Field Robotics* 34.2 (2017), pp. 400–425.

[5] M. Lauria et al. “Elastic locomotion of a four steered mobile robot”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 2721–2722.

[6] T. Tanaka and S. Hirose. “Development of leg-wheel hybrid quadruped AirHopper; design of powerful light-weight leg with wheel”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 3890–3895.

[7] D. Lu et al. “Design and development of a leg-wheel hybrid robot #x201C;HyTRo-I #x201D;” in: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 6031–6036.

[8] S. C. Chen et al. “Quattroped: A Leg–Wheel Transformable Robot”. In: *IEEE/ASME Transactions on Mechatronics* 19.2 (2014), pp. 730–742.

[9] G. Endo and S. Hirose. “Study on Roller-Walker (system integration and basic experiments)”. In: *1999 IEEE International Conference on Robotics and Automation, 1999. Proceedings*. Vol. 3. 1999, 2032–2037 vol.3.

[10] Tom Erez, Yuval Tassa, and Emanuel Todorov. “Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx”. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4397–4404. (Visited on 03/24/2016).

[11] Russel Smith. “ODE: Open dynamics engine”. In: *Online at: http://www. ode. org* (2003).

[12] Rosen Diankov and James Kuffner. “Openrave: A planning architecture for autonomous robotics”. In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34* 79 (2008). (Visited on 03/23/2016).

[13] Michael Sherman and P. Eastman. “Simbody”. In: *Online at: simtk.org/home/simbody* (2015).

[14] Emo Todorov. “MuJoCo physics engine.” In: *Online at: www.mujoco.org* (2015).

[15] Erwin Coumans. “Bullet physics engine. Open Source Software: http://bulletphysics. org”. In: *Open Source Software: http://bulletphysics. org* 1 (2010).

[16] S. Carpin et al. “USARSim: a robot simulator for research and education”. In: *2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 1400–1405.

[17] T. Hirano, T. Sueyoshi, and A. Kawamura. “Development of ROCOS (Robot Control Simulator)-Jump of human-type biped robot by the adaptive impedance control”. In: 2000, pp. 606–611. ISBN: 978-0-7803-5976-5. (Visited on 03/22/2016).

[18] Open Source Robotics Foundation. “Gazebo”. In: *Online at: http://gazebosim.org/* (2016).

[19] Olivier Michel. “Webots: Symbiosis between virtual and real mobile robots”. In: *Virtual Worlds*. 1998, pp. 254–263. (Visited on 03/23/2016).

[20] Gregor Jochmann et al. “The Virtual Space Robotics Testbed: Comprehensive Means for the Development and Evaluation of Components for Robotic Exploration Missions”. en. In: *KI - Künstliche Intelligenz* 28.2 (2014), pp. 85–92. (Visited on 03/24/2016).

[21] E. Rohmer, S. P. N. Singh, and M. Freese. “CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework”. In: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. www.coppeliarobotics.com. 2013.

[22] Serena Ivaldi, Vincent Padois, and Francesco Nori. “Tools for dynamics simulation of robots: a survey based on user feedback”. In: *arXiv:1402.7050 [cs]* (2014). arXiv: 1402.7050. (Visited on 03/22/2016).

[23] Toni Petrinić, Edouard Ivanjko, and Ivan Petrović. “AMORsim - A Mobile Robot Simulator for Matlab”. English. In: 2006. (Visited on 03/23/2016).

[24] Nicolas Bredeche et al. “Roborobo! a Fast Robot Simulator for Swarm and Collective Robotics”. In: *arXiv:1304.2888 [cs]* (2013). arXiv: 1304.2888. (Visited on 03/23/2016).

[25] C. Pinciroli et al. “ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011, pp. 5027–5034.

[26] Paavo Heiskanen. *Development of a dynamic simulator of a mobile robot for astronaut assistance*. ENG. Master Thesis, Continuation Courses. 2008. (Visited on 03/22/2016).

[27] Vadim Tikhanoff et al. “An open-source simulator for cognitive robotics research: the prototype of the iCub humanoid robot simulator”. In: *Proceedings of the 8th workshop on performance metrics for intelligent systems*. ACM, 2008, pp. 57–61. (Visited on 03/24/2016).

[28] Timothée Habra, Paul Fisette, Renaud Ronsse, et al. “Robotran-Yarp interface: a framework for real-time controller development based on multibody dynamics simulation”. In: *ECCOMAS Thematic Conference Multibody Dynamics 2015*. 2015. (Visited on 03/23/2016).

[29] I. Ha et al. “Development of open humanoid platform DARwIn-OP”. In: *2011 Proceedings of SICE Annual Conference (SICE)*. 2011, pp. 2178–2181.

[30] Philipp Allgeuer et al. “A ROS-based software framework for the NimbRo-OP humanoid open platform”. In: *Proceedings of 8th Workshop on Humanoid Soccer Robots, IEEE-RAS Int. Conference on Humanoid Robots, Atlanta, USA*. 2013. (Visited on 03/23/2016).

[31] T. Asfour et al. “ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control”. In: *2006 6th IEEE-RAS International Conference on Humanoid Robots*. 2006, pp. 169–175.

[32] Xu Guan, Haojun Zheng, and Xiuli Zhang. “Biologically inspired quadruped robot biosbot: modeling, simulation and experiment”. In: *2nd International Conference on Autonomous Robots and Agents*. Citeseer, 2004, pp. 261–266. (Visited on 03/24/2016).

[33] H. Song et al. “The Design and Implementation of Quadruped Robot Gait Simulation System”. In: *2010 International Conference on Manufacturing Automation (ICMA)*. 2010, pp. 232–238.

[34] Xuewen Rong et al. “Design and simulation for a hydraulic actuated quadruped robot”. en. In: *Journal of Mechanical Science and Technology* 26.4 (2012), pp. 1171–1177. (Visited on 03/24/2016).

[35] P. S. Freeman and D. E. Orin. “Efficient Dynamic Simulation of a Quadruped Using a Decoupled Tree-Structure Approach”. en. In: *The International Journal of Robotics Research* 10.6 (1991), pp. 619–627. (Visited on 03/24/2016).

[36] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 2009, p. 5. (Visited on 03/23/2016).

[37] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. “YARP: yet another robot platform”. In: *International Journal on Advanced Robotics Systems* 3.1 (2006), pp. 43–48. (Visited on 03/23/2016).

[38] Matthias Kranz et al. “A player/stage system for context-aware intelligent environments”. In: *Proceedings of UbiSys* 6 (2006), pp. 17–21. (Visited on 03/23/2016).

[39] Jean-Christophe Baillie. “Design principles for a universal robotic software platform and application to urbi”. In: *2nd National Workshop on Control Architectures of Robots (CAR’07), Paris, France*. 2007, pp. 150–155. (Visited on 03/23/2016).

[40] E. Einhorn et al. “MIRA - middleware for robotic applications”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012, pp. 2591–2598.

[41] Herman Bruyninckx. “Open robot control software: the OROCOS project”. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 3. IEEE, 2001, pp. 2523–2528. (Visited on 03/23/2016).

[42] Ayssam Elkady and Tarek Sobh. “Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography”. en. In: *Journal of Robotics* 2012 (2012), e959013. (Visited on 03/22/2016).

[43] Paul Fitzpatrick et al. “A middle way for robotics middleware”. en. In: *Journal of Software Engineering for Robotics* 5.2 (2014), pp. 42–49. (Visited on 03/23/2016).

[44] Leonardo Leottau Forero, José Miguel Yáñez, and Javier Ruiz-del-Solar. "Integration of the ROS Framework in Soccer Robotics: The NAO Case". en. In: *RoboCup 2013: Robot World Cup XVII*. Ed. by Sven Behnke et al. Lecture Notes in Computer Science 8371. DOI: 10.1007/978-3-662-44468-9_63. Springer Berlin Heidelberg, 2013, pp. 664–671. ISBN: 978-3-662-44467-2 978-3-662-44468-9. (Visited on 03/23/2016).

[45] Calder Phillips-Grafflin et al. "Toward a user-guided manipulation framework for high-DOF robots with limited communication". en. In: *Intelligent Service Robotics* 7.3 (2014), pp. 121–131. (Visited on 03/22/2016).

[46] Giorgio Metta et al. "The iCub humanoid robot: an open platform for research in embodied cognition". In: *Proceedings of the 8th workshop on performance metrics for intelligent systems*. ACM, 2008, pp. 50–56. (Visited on 03/23/2016).

[47] Olivier Michel, Yvan Bourquin, and Jean-Christophe Baillie. "Robotstadium: Online humanoid robot soccer simulation competition". In: *RoboCup 2008: Robot soccer world cup XII*. Springer, 2008, pp. 580–590. (Visited on 03/23/2016).

[48] J. A. Smith, I. Sharf, and M. Trentini. "PAW: a hybrid wheeled-leg robot". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. 2006, pp. 4043–4048.

[49] James Andrew Smith et al. "Bounding with active wheels and liftoff angle velocity adjustment". In: *The International Journal of Robotics Research* (2009). (Visited on 05/24/2016).

[50] G. Endo and S. Hirose. "Study on Roller-Walker (multi-mode steering control and self-contained locomotion)". In: *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00*. Vol. 3. 2000, 2808–2814 vol.3.

[51] Nam-Su Yuk and Dong-Soo Kwon. "Realization of expressive body motion using leg-wheel hybrid mobile robot: KaMERo1". In: *International Conference on Control, Automation and Systems, 2008. ICCAS 2008*. 2008, pp. 2350–2355.

[52] Roland Siegwart et al. "Innovative design for wheeled locomotion in rough terrain". In: *Robotics and Autonomous Systems*. Intelligent Autonomous Systems - IAS -6 40.2 (2002), pp. 151–162. (Visited on 06/29/2017).

[53] B. Beckman, M. Trentini, and J. Pieper. "Control algorithms for stable range-of-motion behaviours of a multi degree-of-freedom robot". In: *2010 International Conference on Autonomous and Intelligent Systems, AIS 2010*. 2010, pp. 1–6.

[54] P. Tomei. "A simple PD controller for robots with elastic joints". In: *IEEE Transactions on Automatic Control* 36.10 (1991). 00360, pp. 1208–1213.

[55] O. Khatib. "A unified approach for motion and force control of robot manipulators: The operational space formulation". In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53.

[56] Marco Hutter et al. "Quadrupedal locomotion using hierarchical operational space control". en. In: *The International Journal of Robotics Research* (2014). (Visited on 07/22/2017).

[57] A. Shkolnik and R. Tedrake. "Inverse Kinematics for a Point-Foot Quadruped Robot with Dynamic Redundancy Resolution". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 4331–4336.

[58] Yvain de Viragh et al. “Trajectory optimization for wheeled-legged quadrupedal robots using linearized zmp constraints”. In: *IEEE-RAL* (2019).

[59] Sung-Hee Lee and Ambarish Goswami. “A momentum-based balance controller for humanoid robots on non-level and non-stationary ground”. en. In: *Auton Robot* 33.4 (2012). 00075, pp. 399–414. (Visited on 04/06/2016).

[60] A. Herzog et al. “Balancing experiments on a torque-controlled humanoid with hierarchical inverse dynamics”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*. 00042. 2014, pp. 981–988.

[61] S. H. Hyon, R. Osu, and Y. Otaka. “Integration of multi-level postural balancing on humanoid robots”. In: *IEEE International Conference on Robotics and Automation, 2009. ICRA ’09*. 00037. 2009, pp. 1549–1556.

[62] S. h Hyon and G. Cheng. “Passivity-Based Full-Body Force Control for Humanoids and Application to Dynamic Balancing and Locomotion”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 4915–4922.

[63] M. de Lasa and M. Buehler. “Dynamic compliant quadruped walking”. In: *IEEE International Conference on Robotics and Automation, 2001. Proceedings 2001 ICRA*. Vol. 3. 2001, 3153–3158 vol.3.

[64] J. Z. Kolter, M. P. Rodgers, and A. Y. Ng. “A control architecture for quadruped locomotion over rough terrain”. In: *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008*. 00116. 2008, pp. 811–818.

[65] S. i An, Y. Oh, and D. S. Kwon. “Zero-moment point based balance control of leg-wheel hybrid structures with inequality constraints of dynamic behavior”. In: *2012 IEEE International Conference on Robotics and Automation (ICRA)*. 2012, pp. 2365–2370.

[66] S. i An, Y. Oh, and D. S. Kwon. “Zero-moment point based balance control of leg-wheel hybrid structures with inequality constraints of kinodynamic behavior”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 00001. 2012, pp. 2471–2477.

[67] A. Suzumura and Y. Fujimoto. “Real-Time Motion Generation and Control Systems for High Wheel-Legged Robot Mobility”. In: *IEEE Transactions on Industrial Electronics* 61.7 (2014), pp. 3648–3659.

[68] Patrick F. Muir and Charles P. Neuman. “Kinematic modeling of wheeled mobile robots”. en. In: *J. Robotic Syst.* 4.2 (1987), pp. 281–340. (Visited on 02/03/2018).

[69] C. P. Connette et al. “Control of an pseudo-omnidirectional, non-holonomic, mobile robot based on an ICM representation in spherical coordinates”. In: *2008 47th IEEE Conference on Decision and Control*. 2008, pp. 4976–4983.

[70] M. F. Selekwa and J. R. Nistler. “Path tracking control of four wheel independently steered ground robotic vehicles”. In: *2011 50th IEEE Conference on Decision and Control and European Control Conference*. 2011, pp. 6355–6360.

[71] M. Sorour et al. “Motion Discontinuity-Robust Controller for Steerable Mobile Robots”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 452–459.

[72] C. P. Connette et al. “Singularity avoidance for over-actuated, pseudo-omnidirectional, wheeled mobile robots”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 4124–4130.

[73] U. Schwesinger, C. Pradalier, and R. Siegwart. “A novel approach for steering wheel synchronization with velocity/acceleration limits and mechanical constraints”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5360–5366.

[74] C. Stöger, A. Müller, and H. Gattringer. “Kinematic analysis and singularity robust path control of a non-holonomic mobile platform with several steerable driving wheels”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 4140–4145.

[75] M. Sorour et al. “Kinematic modeling and singularity treatment of steerable wheeled mobile robots with joint acceleration limits”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 2110–2115.

[76] Christophe Grand et al. “Stability and traction optimization of a reconfigurable wheel-legged robot”. In: *The International Journal of Robotics Research* 23.10-11 (2004). 00132, pp. 1041–1058. (Visited on 05/17/2016).

[77] Christophe Grand, Faiz Benamar, and Frédéric Plumet. “Motion kinematics analysis of wheeled–legged rover over 3D surface with posture adaptation”. In: *Mechanism and Machine Theory* 45.3 (2010), pp. 477–495. (Visited on 05/17/2016).

[78] P. R. Giordano et al. “On the kinematic modeling and control of a mobile platform equipped with steering wheels and movable legs”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 4080–4087.

[79] A. Dietrich et al. “Singularity avoidance for nonholonomic, omnidirectional wheeled mobile platforms with variable footprint”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 6136–6142.

[80] A. Suzumura and Y. Fujimoto. “Workspace control of a wheel-legged mobile robot for gyrating locomotion with movable leg”. In: *2013 IEEE International Conference on Mechatronics (ICM)*. 2013, pp. 641–647.

[81] K. Nagano and Y. Fujimoto. “A control method of low speed wheeled locomotion for a wheel-legged mobile robot”. In: *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*. 2014, pp. 332–337.

[82] K. Nagano and Y. Fujimoto. “The stable wheeled locomotion in low speed region for a wheel-legged mobile robot”. In: *2015 IEEE International Conference on Mechatronics (ICM)*. 2015, pp. 404–409.

[83] Sang-ik An, Yonghwan Oh, and Dong-Soo Kwon. “Zero-moment point based balance control of leg-wheel hybrid structures with inequality constraints of kinodynamic behavior”. In: *IROS*. 2012.

[84] Pierre Jarrault, Christophe Grand, and Philippe Bidaud. “Robust obstacle crossing of a wheel-legged mobile robot using minimax force distribution and self-reconfiguration”. In: *IROS*. IEEE. 2011.

[85] Guiyang Xin et al. “A model-based hierarchical controller for legged systems subject to external disturbances”. In: *ICRA*. 2018.

[86] C Dario Bellicoso et al. “Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots”. In: *IEEE-RAL* (2018).

[87] Gerardo Bledt, Patrick M Wensing, and Sangbae Kim. “Policy-regularized model predictive control to stabilize diverse quadrupedal gaits for the MIT cheetah”. In: *IROS*. IEEE. 2017.

[88] Jared Di Carlo et al. “Dynamic locomotion in the MIT Cheetah 3 through convex model-predictive control”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* 2018.

[89] Zhijun Li et al. “Dynamic balance optimization and control of quadruped robot systems with flexible joints”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2015).

[90] Alexander W Winkler, Carlos Mastalli, Ioannis Havoutis, et al. “Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain”. In: *ICRA*. 2015.

[91] Victor Barasuol, Jonas Buchli, et al. “A reactive controller framework for quadrupedal locomotion on challenging terrain”. In: *ICRA*. 2013.

[92] Ting-Yung Wen John. “Control of Nonholonomic Systems”. In: *The Control Handbook (three volume set)*. Ed. by William S Levine. CRC press, 2018. Chap. 76.3, pp. 1359–1368.

[93] Reza N Jazar. “Theory of applied robotics: kinematics, dynamics, and control”. In: Springer Science & Business Media, 2010. Chap. Velocity Kinematics.

[94] Reza N Jazar. “Theory of applied robotics: kinematics, dynamics, and control”. In: Springer Science & Business Media, 2010. Chap. Acceleration Kinematics.

[95] Chung Woojin and Iagnemma Karl. “Wheeled Robots”. In: *Springer handbook of robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer, 2016. Chap. 24, pp. 575–594.

[96] Wieber Pierre-Brice, Tedrake Russ, and Kuindersma Scott. “Modeling and Control of Legged Robots”. In: *Springer handbook of robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer, 2016. Chap. 48, pp. 1203–1234.

[97] Marko B. Popovic, Ambarish Goswami, and Hugh Herr. “Ground reference points in legged locomotion: Definitions, biological trajectories and control implications”. In: *The International Journal of Robotics Research* 24.12 (2005). 00241, pp. 1013–1032. (Visited on 04/06/2016).

[98] Alin Albu-Schäffer and Antonio Bicchi. “Actuators for Soft Robots”. In: *Springer handbook of robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer, 2016. Chap. 21, pp. 500–530.

[99] Guanfeng Liu and Zexiang Li. “A unified geometric approach to modeling and control of constrained mechanical systems”. In: *IEEE Transactions on Robotics and Automation* 18.4 (2002), pp. 574–587.

[100] F. Aghili. “Inverse and direct dynamics of constrained multibody systems based on orthogonal decomposition of generalized force”. In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 3. 2003, 4035–4041 vol.3.

[101] F. Aghili. “A unified approach for inverse and direct dynamics of constrained multibody systems based on linear projection operator: applications to control and simulation”. In: *IEEE Transactions on Robotics* 21.5 (2005), pp. 834–849.

[102] M. Mistry, J. Buchli, and S. Schaal. “Inverse dynamics control of floating base systems using orthogonal decomposition”. In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 3406–3412.

[103] Chiaverini Stefano, Oriolo Giuseppe, and Maciejewski Anthony A. “Redundant Robots”. In: *Springer handbook of robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer, 2016. Chap. 10, pp. 221–242.

[104] Bruno Siciliano and Jean-Jacques Slotine. “A general framework for managing multiple tasks in highly redundant robotic systems”. In: *Advanced Robotics* (1991), pp. 1211–1216.

[105] JS Yuan. “Closed-loop manipulator control using quaternion feedback”. In: *IEEE Journal on Robotics and Automation* 4.4 (1988), pp. 434–440.

[106] Przemysław Dobrowolski. “Swing-twist decomposition in Clifford algebra”. In: *arXiv preprint arXiv:1506.05481* (2015).

[107] Navvab Kashiri, Jörn Malzahn, and Nikos G Tsagarakis. “On the sensor design of torque controlled actuators: A comparison study of strain gauge and encoder-based principles”. In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1186–1194.

[108] Manuel G Catalano et al. “Adaptive synergies for the design and control of the Pisa/IIT SoftHand”. In: *The International Journal of Robotics Research* 33.5 (2014), pp. 768–782.

[109] Zeyu Ren et al. “HERI II: A robust and flexible robotic hand based on modular finger design and under actuation principles”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1449–1455.

[110] N. Kashiri et al. “Enhanced physical interaction performance for compliant joint manipulators using proxy-based Sliding Mode Control”. In: *2014 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*. Vol. 02. 2014, pp. 175–183.

[111] Martin L. Felis. “RBDL: an efficient rigid-body dynamics library using recursive algorithms”. In: *Autonomous Robots* (2016), pp. 1–17.

[112] I. Kao, K. Lynch, and J. Burdick. “Contact Modeling and Manipulation”. In: *Springer handbook of robotics*. Ed. by B. Siciliano and O. Khatib. Springer, 20016. Chap. 37, pp. 931–954.

[113] Malgorzata Kamedula, Navvab Kashiri, and Nikos G Tsagarakis. “On the kinematics of wheeled motion control of a hybrid wheeled-legged centauro robot”. In: *IEEE/RSJ Int. Conf. Intell. Robot. and Syst.* IEEE. 2018, pp. 2426–2433.

Appendix A

State Estimation

In order to test the proposed controllers, simple state estimators for the robot world posture and the ground reaction forces have been implemented in this work. This chapter is dedicated to these state estimation algorithms: in Section A.1 estimation of the robot world posture from the encoders and IMU readings have been described at the geometric (Section A.1.1) and kinematic (Section A.1.2) levels, and an algorithm for the estimation of the ground reaction forces based on the inverse dynamics approach have been given in Section A.2. Finally, in Section A.3 the summary is given.

A.1 Odometry

To compute the robot world posture, the link-side encoders and Inertial Measurement Unit (IMU) readings have been used. Due to the drift in the IMU readings on the rotation around the gravity vector, this rotation has been removed from the IMU measurements and included into the estimation variable. To that end, the twist-swing decomposition (Section 3.4) has been used to compute the robot orientation without its heading, and the null-space projection matrix with respect to the gravity vector has been applied to the IMU velocity readings,

$$\hat{\boldsymbol{\rho}}_{b|Pg} = \boldsymbol{\rho}_{IMU|Pg}, \quad \hat{\boldsymbol{\omega}}_{b|Pg} = \hat{\boldsymbol{\omega}}_{IMU|Pg}. \tag{A.1}$$

The current twist is estimated using the estimation of the angular velocity computed in the previous time step,

$$\hat{\rho}_{b|g|t} \mathrel{+}= \underline{\hat{\omega}}_{b|g|t-T} T. \tag{A.2}$$

A.1.1 Twist and Position Estimation

The motion of the ith wheel centre is estimated assuming the rolling assumption (3.5), that reads

$$\hat{\boldsymbol{x}}_{w,i} \mathrel{+}= \boldsymbol{x}_{s,i} q_{w,i} r, \tag{A.3}$$

where $\hat{\boldsymbol{x}}_{w,i} \in \Re^3$ symbolises the estimated value, $q_{w,i} \in \Re^1$ stands for the encoder reading for the wheel rolling DoF, $r \in \Re^1$ refers to the wheel radius, and $\boldsymbol{x}_{s,i} \in \Re^3$ denotes the wheel rolling direction that is orthogonal to the ground normal and the wheel axis; $||\boldsymbol{x}_{s,i}|| = 1$. To estimate the base position, the distance from the robot base to the ith wheel centre is computed from the robot forward kinematics

$$\boldsymbol{x}^*_{w,i} = \boldsymbol{x}_{w,i}(\boldsymbol{q}_m), \tag{A.4}$$

where $\boldsymbol{x}^*_{w,i} \in \Re^3$ stands for the estimated value, $\boldsymbol{x}_{w,i}() \in \Re^3$ symbolises the wheel centre forward kinematics function, $\boldsymbol{q}_m \in \Re^n$ is a robot state based on the encoders and IMU readings with the estimated twist. Finally, the estimated base position ($\hat{\boldsymbol{x}}_{b,i} \in \Re^3$) is computed, for each of the robot legs in the ground contact, with

$$\hat{\boldsymbol{x}}_{b,i} = \hat{\boldsymbol{x}}_{w,i} - \boldsymbol{x}^*_{w,i}. \tag{A.5}$$

Standard method to fuse different measurements is to use the Kalman filter that is an optimal state estimator for the linear systems. However, it is not robust to the outliers. When the estimator input consists of the many points and the percentage of outliers remains below a certain threshold, and various robust Kalman filter and outlier detection methods can be used.

In the considered case of the world posture of the CENTAURO robot, only four new estimations are available each step, and thus any outlier point constitutes at least 25% of measurements. The main sources of uncertainties are the IMU and encoders noises, where the former is used directly in all the estimation points, and the latter – unless the sensor malfunctioned – should have only a minor influence on the estimation points. Therefore, a main source of the outliers in the robot world posture estimations is the wheel slippage. To improve robustness of the estimation with minimal computational load, distance between the points has been chosen as a simple measure of the reliability of the estimation. It reads

$$
\begin{aligned}
&\text{if } ||\mathbb{K}|| = 1, \\
&\qquad \hat{\boldsymbol{x}}_b = \hat{\boldsymbol{x}}_{i,b} && \text{where } i \in \mathbb{K}; \\
&\text{if } ||\mathbb{K}|| = 2, \\
&\qquad \hat{\boldsymbol{x}}_b = 0.5\left(\hat{\boldsymbol{x}}_{i,b} + \hat{\boldsymbol{x}}_{j,b}\right) && \text{where } i, j \in \mathbb{K}, i \neq j; \\
&\text{if } ||\mathbb{K}|| = 3, \\
&\qquad \hat{\boldsymbol{x}}_b = \min_{i \in \mathbb{K}} \sum_{j \in \mathbb{K}-\{i\}} ||\hat{\boldsymbol{x}}_{i,b} - \hat{\boldsymbol{x}}_{j,b}||; \\
&\text{if } ||\mathbb{K}|| > 3, \\
&\qquad \mathbb{K} = \mathbb{K} - \{i : \max_{i \in \mathbb{K}} \sum_{j \in \mathbb{K}-\{i\}} ||\hat{\boldsymbol{x}}_{i,b} - \hat{\boldsymbol{x}}_{j,b}||\}.
\end{aligned}
\tag{A.6}
$$

where $\mathbb{K}$ is initialized as a set of all legs in the ground contact. If only one estimation point is available, it is chosen as the final estimation. On the other hand, if two estimation points are available, the average value is used. In case three legs are in the ground contact, the point 'in the middle', i.e., the point closest to the other estimation points, is chosen. Finally, if more than three points are available, the point furthest away from other estimation points is removed from the set, and the algorithm is rerun with the less number of considered points. After choosing the final estimation point, the new estimations for the wheel centres are computed with the forward kinematics considering the estimated robot world posture and encoder readings for the actuated DoFs

$$\hat{\boldsymbol{x}}_{w,i} = \boldsymbol{x}_{w,i}(\hat{\boldsymbol{q}}_b, \boldsymbol{q}_a). \tag{A.7}$$

Despite providing a simple approach to select the final estimation point, this algorithm provides a reliable short-term proprioceptive state estimation with the robustness up to 50% of measurements points. To achieve higher robustness to the wheel slippage, a dedicated slippage detection algorithm would be required. However, for the long-term state estimation, a more complex algorithm that would account for the drift on the robot twist and wheel centre positions would be required.

A.1.2 Kinematics

To estimate the robot base linear velocity and the angular velocity along the gravity vector, note that the position of the i^{th} wheel centre $\boldsymbol{x}_{w,i} \in \Re^3$ reads

$$\boldsymbol{x}_{w,i} = \boldsymbol{x}_b + {}^{b}\boldsymbol{x}_{w,i}, \tag{A.8}$$

where $\boldsymbol{x}_b$ is the position of the robot reference point, and ${}^b\boldsymbol{x}_{w,i} \in \Re^3$ is the vector from the robot reference point to the wheel centre. The wheel centre velocity, analogous to (3.3), reads

$$\dot{\boldsymbol{x}}_{w,i} = \dot{\boldsymbol{x}}_b + \boldsymbol{\omega}^b \times {}^b\boldsymbol{x}_{w,i} + {}^b\dot{\boldsymbol{x}}_{w,i}, \tag{A.9}$$

where $\boldsymbol{\omega}^b \times {}^b\boldsymbol{x}_{w,i}$ represents the point velocity induced by the motion of the robot reference frame, and ${}^b\dot{\boldsymbol{x}}_{w,i}$ denotes the point velocity in the robot reference frame. Taking into account the rolling assumption, (A.9) reads

$$\boldsymbol{\omega}_{w,i} \times \boldsymbol{n}r = \dot{\boldsymbol{x}}_b + \boldsymbol{\omega}_b \times {}^b\boldsymbol{x}_{w,i} + {}^b\dot{\boldsymbol{x}}_{w,i}. \tag{A.10}$$

That can be further decomposed into

$$(\boldsymbol{\omega}_{b|g} + \boldsymbol{\omega}_{b|Pg} + {}^b\boldsymbol{\omega}_{w,i}) \times \boldsymbol{n}r = \dot{\boldsymbol{x}}_b + (\boldsymbol{\omega}_{b|g} + \boldsymbol{\omega}_{b|Pg}) \times {}^b\boldsymbol{x}_{w,i} + {}^b\dot{\boldsymbol{x}}_{w,i} \tag{A.11}$$

where $\boldsymbol{\omega}_{b|g} \in \Re^3$ stands for the base angular velocity along the gravity direction, $\boldsymbol{\omega}_{b|Pg} \in \Re^3$ refers to the base angular velocity in the gravity direction null-space, and ${}^b\boldsymbol{\omega}_{w,i} \in \Re^3$ symbolises a wheel angular velocity generated be the robot actuated DoFs. Then (A.11) can be expressed as

$$-\dot{\boldsymbol{x}}_b + ({}^b\boldsymbol{x}_{w,i} - \boldsymbol{n}r) \times \boldsymbol{\omega}_{b|g} = -(\boldsymbol{\omega}_{b|Pg} + {}^b\boldsymbol{\omega}_{w,i}) \times \boldsymbol{n}r + \boldsymbol{\omega}_{b|Pg} \times {}^b\boldsymbol{x}_{w,i} + {}^b\dot{\boldsymbol{x}}_{w,i}. \tag{A.12}$$

(A.12) is a linear equation with respect to the estimation variable, i.e., robot base linear/angular velocities

$$\mathbf{J}_{b,i} \begin{bmatrix} \dot{\hat{\boldsymbol{x}}}_{b,i} \\ \underline{\hat{\omega}}_{b|g,i} \end{bmatrix} = \boldsymbol{a}_{b,i}. \tag{A.13}$$

An estimation of the robot base velocity can be obtained solving

$$\begin{bmatrix} \mathbf{J}_{b,i} \\ \mathbf{J}_{b,j} \end{bmatrix} \begin{bmatrix} \dot{\hat{\boldsymbol{x}}}_{b,i,j} \\ \underline{\hat{\omega}}_{b|g,i,j} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_{b,i} \\ \boldsymbol{a}_{b,j} \end{bmatrix} \quad \text{where } i, j \in \mathbb{K},\ i \neq j \tag{A.14}$$

for any two wheel points in the contact with the ground that do not belong to the same wheel. At least, two legs on the ground are required to find four estimation variables. However, if there are more contact points, more than one estimation can be computed with (A.14). To consider the wheel slippage, algorithm (A.6) has been adopted to the chose the estimated linear velocity and, independently, the angular velocity. The set $\mathbb{K}$ at the algorithm initialization consists of estimates for all $i,\ j$ pairs of legs in the ground contact.

A.2 Ground Reaction Forces

The robot ground reaction forces are estimated using the rigid-body inverse dynamics with the contact point assumptions; that reads (3.22)

$$\begin{aligned} &\mathbf{M}(\bar{\boldsymbol{q}})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\bar{\boldsymbol{q}}, \dot{\boldsymbol{q}}) + \boldsymbol{F}_g(\bar{\boldsymbol{q}}) = \mathbf{S}^T \boldsymbol{\tau}_t(\boldsymbol{q}_a, \dot{\boldsymbol{q}}_a) + \mathbf{J}_c^T(\bar{\boldsymbol{q}})\boldsymbol{\lambda}, \\ &\ddot{\boldsymbol{x}}_\mathrm{c} = \mathbf{J}_c(\bar{\boldsymbol{q}})\ddot{\boldsymbol{q}} + \dot{\mathbf{J}}_\mathrm{c}(\bar{\boldsymbol{q}}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}, \end{aligned} \tag{A.15}$$

where $\ddot{\boldsymbol{x}}_\mathrm{c} \in \Re^k$ corresponds to the accelerations of the constrained directions that are uniquely defined by the contact point assumptions, and $\mathbf{J}_\mathrm{c} \in \Re^{k \times n}$ represents the corresponding contact points Jacobian. From (3.22), the estimated reaction forces read

$$\hat{\boldsymbol{\lambda}} = \boldsymbol{\Lambda}_\mathrm{c}^{-1}\left(\ddot{\boldsymbol{x}}_\mathrm{c} - \dot{\mathbf{J}}_\mathrm{c}(\bar{\boldsymbol{q}}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \mathbf{J}_\mathrm{c}(\bar{\boldsymbol{q}})\mathbf{M}^{-1}\left(\bar{\boldsymbol{q}})(\boldsymbol{c}(\bar{\boldsymbol{q}}, \dot{\boldsymbol{q}}) + \boldsymbol{F}_g(\bar{\boldsymbol{q}}) - \mathbf{S}^T \boldsymbol{\tau}_t(\boldsymbol{q}_a, \dot{\boldsymbol{q}}_a)\right)\right),$$

where $\boldsymbol{\Lambda}_\mathrm{c} = \mathbf{J}_\mathrm{c}(\bar{\boldsymbol{q}})\mathbf{M}^{-1}(\bar{\boldsymbol{q}})\mathbf{J}_\mathrm{c}^T(\bar{\boldsymbol{q}})$ and $\hat{\boldsymbol{\lambda}} \in \Re^k$ symbolises the vector of the estimated reaction forces. Note that in this method the number, placement and type of the contacts have to be known a priori for the obtained result to be reliable. If the contact point assumption is violated, e.g., a contact point is sliding, and thus

$$\ddot{\boldsymbol{x}}_\mathrm{c}(t) \neq \ddot{\boldsymbol{x}}_\mathrm{c}^*,$$

where $\ddot{\boldsymbol{x}}_\mathrm{c}(t) \in \Re^k$ are the real contact points accelerations at time t, and $\ddot{\boldsymbol{x}}_\mathrm{c}^* \in \Re^k$ denotes the assumed contact point acceleration at time t, the estimated external forces will include a difference between expected and executed accelerations

$$\hat{\boldsymbol{\lambda}} = \boldsymbol{\lambda} - \boldsymbol{\Lambda}_\mathrm{c}^{-1}\left(\ddot{\boldsymbol{x}}_\mathrm{c} - \ddot{\boldsymbol{x}}_\mathrm{c}^*\right),$$

where $\boldsymbol{\lambda} \in \Re^k$ represents the vector of real reaction forces.

A.3 Summary

In this chapter, state estimation algorithms implemented in this work have been described. It includes a robot world posture and velocity estimation robust to the wheel slippage. To achieve higher robustness to the wheel slippage, a dedicated slippage detection algorithm would be required. Proposed algorithms provide a reliable short-term estimation; however, they may be susceptible to the drift in the wheel position estimation and the IMU readings. Thus, for the long-term state estimation, a more complex algorithm that would account for this drift would be required. Furthermore, an algorithm for the estimation of the ground reaction forces based on the contact point assumptions and the robot dynamic model has been described. The algorithms introduced in this chapter have been implemented on the CENTAURO robot and used in the experiments described in Chapter 7 and Chapter 8.

Appendix B

SOL-FER - code listings

```
mapping: # load mappings
    RBDL_LOWER: # mapping name
        loading: model # use the RBDL model
        urdf: /robot_lower_body # from ROS parameter server
    RBDL_UPPER: # mapping name
        loading: model # use the RBDL model
        urdf: # from file
            path: *config
            file: mwoibn/urdf/centauro_upper_body.urdf
```

Listing B.1 Example of the predefined **Bidirectional Map** from external URDF file.

```
    protected: // variables declaration
        // Handler of Robot Points to keep all the Robot Points
        Handler<RobotPoint> _points;

    public:
    // memory allocation, create the Norm Points
    function init(){
        // create the basic Linear Points for the rigid body points
        _points.add(LinearPoint("LeftHand"));
        _points.add(LinearPoint("RightHand"));
        _points.add(LinearPoint("Base"));
        // create the point that computes the difference between the
         ↪  "LeftHand" and "Base"
        _points.add(Minus(_points[0],_points[2]);
        // create the point that computes the difference between the
         ↪  "RightHand" and "Base"
        _points.add(Minus(_points[1],_points[2]);
        // create the point that computes norm of the penultimate
         ↪  point in the Handler
        _points.add(Norm(_points.end(1));
        // create the point that computes norm of the penultimate
         ↪  point in the Handler
        _points.add(Norm(_points.end(1));
        // set an offset between the "LeftHand" frame origin and the
         ↪  tracked point given in the "LeftHand" frame
        _points[0].point.position.setFixed(Vector(0,0,-0.15));
        // set an offset between the "RightHand" frame origin and the
         ↪  tracked point given in the "RightHand" frame
        _points[1].point.position.setFixed(Vector(0,0,-0.15));
   }

   function update(){ // main loop
        // compute the norms
        _points.update(false); // do not update the jacobians
        // print the results
        std::cout << "Right Hand distance from the Base is " <<
         ↪  _points.end(1).getState().transpose() << "." << std::endl;
        std::cout << "Left Hand distance from the Base is " <<
         ↪  _points.end(0).getState().transpose() << "." << std::endl;
        }
```

Listing B.2 Pseudo-code to compute the distances between the end-effectors and the reference point when the tracked points do not coincide with the frames origins.

```cpp
  virtual void compute() = 0;
  virtual void computeJacobian() = 0;

  virtual void update(bool jacobian = true);

  const mwoibn::Matrix& jacobian() const;
  const mwoibn::VectorN& point() const;

  int size();
  int rows();
  int cols();

  Point& operator=(const Point& other);
  Point& operator+(const Point& other);
  Point& operator-(const Point& other);
  Point& operator+=(const Point& other);
  Point& operator-=(const Point& other);
```

Listing B.3 The Robot Points Interface.

```yaml
link_side_online: # name of the communicating pipe
    space: JOINT # pipe space
    dofs: # which DoFs are considered
        chain: all # which part of the robot should be used
        type: actuated # which type of DoFs is considered
        mapping: PYTHON # which mapping should be used
    interface: # which interfaces are active
        position: true
        velocity: true
        effort: true
    function: state # which State it is attached to
```

Listing B.4 Example of the **pipe** configuration in the joint-space.

```
floating_base:
        space: OPERATIONAL
        convention:
                orientation:
                        type: QUATERNION
                        convention: HAMILTONIAN
                position:
                        type: FULL
        dofs:
                chain: all
                type: body
                name: pelvis
                mapping: RBDL
        interface:
                position: true
                velocity: false
                effort: false
```

Listing B.5 Example of the **pipe** configuration for the robot rigid-body.

```
ros:
    feedback:
        whole_body:
            source: desired_state
            message: custom_messages::CustomCmnd
            initialize: false
    controller:
        whole_body:
            sink: desired_state
xbot:
    feedback:
        whole_body:
            layer: NRT
            source: desired_state
    controller:
        whole_body:
            layer: NRT
            sink: desired_state
```

Listing B.6 Example of the middleware specific **pipe** configuration.

```
modules:
        joint_online: # plugin name/configuration id
            layer: NRT # layer the plugins runs at
            robot: joint_space # which robot configuration should be
             ↪  loaded
            controller: direct # which controllers should be loaded
            mode: full # if 'full' all controllers are loaded, if
             ↪  'idle' the desired states are not send to the
             ↪  lower-level controller
            kinematics: false # does it require kinematic update
            dynamics: false # does it require dynamic update
            model_change: false # does it modify the robot state
             ↪  internally
```

Listing B.7 Example of the plugin configuration for the joint-space controller.

```
robot: # robot configurations
        online: # configuration id
                feedback: # which feedbacks should be loaded
                        layer: online # read a predefined set-up
                actuators: # if the actuation model should be loaded
                        read: true
                contacts: # does it need contacts
                        read: true
        offline:
                feedback:
                        list: [whole_body, reference] # read given
                         ↪  list
                actuators:
                        read: false
                contacts:
                        read: true
```

Listing B.8 Example of the predefined **Robot** configurations for the online and offline plugins. The 'reference' and 'whole-body' feedbacks in Code B.8 are both position/velocity feedbacks of the full robot state, where the former subscribes to the robot current state (function: state in the **pipe** configuration) and the latter subscribes to the desired state (function: reference in the **pipe** configuration).

```
feedbacks: # predefined feedback set-ups
        online: [link_side_online, odometry, reference]
        offline: [whole_body]
controllers: # predefined controller set-ups
        direct:
                list: [position_controller, velocity_controller]
```

Listing B.9 Example of the predefined pipeline set-ups.

```
config_name:
        secondary_file: # if present, use secondary file
            path: *support # specify path to the file
            file: upper_body.yaml # file name with file extension
```

Listing B.10 Example of the predefined pipeline set-ups.

```
mwoibn:
        ros: # cheange the urdf to the upper-body
                source:
                        urdf: "/robot_upper_body"
        xbot:
                source:
                        urdf:
                                file: mwoibn/urdf/centauro_upper_body.
        feedbacks:
                online: [link_side_online] # remove the state
                 ↪  estimation from the deafult feedbacks, upper-body
                 ↪  is a static model
```

Listing B.11 An example secondary configuration file for the manipulation plugin. This file changes the robot URDF to the fixed base upper-body model, and it removes the state estimation from the default feedback.

```yaml
mwoibn:
        controller:
                position_controller: # control only the robot
                 ↪  lower-body
                        dofs:
                                chain: base
                                type: unactuated
                                mapping: PYTHON
        ros: # change the robot urdf source to the lower-body urdf
                source:
                       urdf: "/robot_lower_body"
        xbot: # change the robot urdf source to the lower-body urdf
                source:
                        urdf:
                                file:
                                 ↪  "mwoibn/urdf/centauro_lower_body.urdf"
```

Listing B.12 An example secondary configuration file for the odometry plugin. In this file, the default whole-body robot URDF is changed to the lower-body model, and the default position controller configuration is modified to control only the robot floating base, i.e. only unactuated DoFs in the ’base’ chain.

```
public:
Base(mwoibn::robot_class::Robot& robot);

virtual ~Base() {}

virtual void init() = 0; // runs at the robot initalization
virtual void update() = 0; // runs at each update loop
virtual void send() = 0; // runs the outputs communication layer
virtual void stop() = 0; // runs when the module is stoped
virtual void close() = 0; // runs on the module shut down
virtual void setRate(double rate); // update the module update
  ↪ frequency

virtual void startLog(mwoibn::common::Logger& logger); // initialize
  ↪ the logger
virtual void log(mwoibn::common::Logger& logger, double time) = 0; //
  ↪ write the log loop

mwoibn::robot_class::Robot& model(){return _robot;}; // provides acces
  ↪ to the Robot object

mwoibn::common::Flag kinematics; // does the kinematic update is
  ↪ required?
mwoibn::common::Flag dynamics; // does the dynamic update is required?
mwoibn::common::Flag modify; // does the module modifes the robot
  ↪ current state?

const std::string& name(){return _name;} // name of the module
```

Listing B.13 Public interface of the module interface.

```
#include "mgnss/plugins/generator.h" // the plugin template
#include "mgnss/ros_callbacks/joint_states.h" // the higher-level communication
#include "mgnss/controllers/joint_states.h" // component - joint-space controller
#include <custom_services/jointStateCmnd.h> // ros topic

template<typename Subscriber, typename Service, typename Node, typename Publisher>
class JointStates : public mgnss::plugins::Generator<Subscriber, Service, Node, Publisher>
{
  typedef mgnss::plugins::Generator<Subscriber, Service, Node, Publisher> Generator_;

public:
    JointStates() : Generator_("joint_states"){ }

    virtual ~JointStates(){}

protected:

// specify component to be loaded
virtual void _resetPrt(YAML::Node config){
    Generator_::controller_ptr.reset(new mgnss::controllers::JointStates(
    *Generator_::_robot_ptr.begin()->second, config));
}

// initialize the higher-level communication
virtual void _initCallbacks(YAML::Node config){
    Generator_::_srv.push_back(Generator_::n->template advertiseService<custom_services::jointStateCmnd::Request, custom_services::jointStateCmnd::Response>(Generator_::controller_ptr->name() + "/trajectory",
    boost::bind(
    &mgnss::ros_callbacks::joint_states::referenceHandler,_1, _2, static_cast<mgnss::controllers::JointStates*>(
    Generator_::controller_ptr.get()))));

}
};
```

Listing B.14 Plugin template specialization for the joint-space controller **Component**.

```
shared:
    layer: NRT
    plugins: [UpperBodyIK::upper_body_ik, odometry3::odometry,
     ↪ ground_forces, gravity_compensation,
     ↪ NwheelsZMPII::wheeled_motion]
```

Listing B.15 A configuration of the **shared plugin** to load 5 **Components/Plugins**.