



SAPIENZA
UNIVERSITÀ DI ROMA

Learning-based methods for planning and control of humanoid robots

Sapienza University of Rome

PhD Program in Automatic Control, Bioengineering and Operations Research
(XXXV cycle)

Paolo Maria Viceconte

ID number 1601242

Advisors

Prof. Giuseppe Oriolo

Dr. Daniele Pucci

May 2023

Thesis defended on 19 May 2023

in front of a Board of Examiners composed by:

Prof. Paola Cappanera, Università di Firenze, Italy

Prof. Danilo Pani, Università di Cagliari, Italy

Prof. Paolo Valigi, Università di Perugia, Italy


Thesis reviewed by the External Evaluators:

Dr. Francesco Nori, DeepMind, London, UK

Prof. Michiel van de Panne, UBC, Vancouver, Canada

Learning-based methods for planning and control of humanoid robots

PhD thesis. Sapienza University of Rome

© Paolo Maria Viceconte. License: Attribution 4.0 International (CC BY 4.0) 

This thesis has been typeset by L^AT_EX and the Sapthesis class.

Author's email: paolo.viceconte@iit.it, viceconte@diag.uniroma1.it, paolomariaviceconte@gmail.com

What is research but a blind date with knowledge?
Will Harvey

Research is formalized curiosity, it is poking and prying with a purpose.
Zora Neale Hurston

Abstract

Nowadays, humans and robots are more and more likely to coexist as time goes by. The anthropomorphic nature of humanoid robots facilitates physical human-robot interaction, and makes social human-robot interaction more natural. Moreover, it makes humanoids ideal candidates for many applications related to tasks and environments designed for humans.

No matter the application, an ubiquitous requirement for the humanoid is to possess proper locomotion skills. Despite long-lasting research, humanoid locomotion is still far from being a trivial task. A common approach to address humanoid locomotion consists in decomposing its complexity by means of a model-based hierarchical control architecture. To cope with computational constraints, simplified models for the humanoid are employed in some of the architectural layers. At the same time, the redundancy of the humanoid with respect to the locomotion task as well as the closeness of such a task to human locomotion suggest a data-driven approach to learn it directly from experience.

This thesis investigates the application of learning-based techniques to planning and control of humanoid locomotion. In particular, both deep reinforcement learning and deep supervised learning are considered to address humanoid locomotion tasks in a crescendo of complexity.

First, we employ deep reinforcement learning to study the spontaneous emergence of balancing and push recovery strategies for the humanoid, which represent essential prerequisites for more complex locomotion tasks.

Then, by making use of motion capture data collected from human subjects, we employ deep supervised learning to shape the robot walking trajectories towards an improved human-likeness.

The proposed approaches are validated on real and simulated humanoid robots. Specifically, on two versions of the iCub humanoid: iCub v2.7 and iCub v3.

Acknowledgments

It was not an easy walk, and if I got this far I can only thank those who stood by me.

I am deeply grateful to Prof. Oriolo for teaching me the value of research in the first place. All the times we exchanged views, your determination in getting to the core of the problem from a scientific standpoint inspired me profound reflections.

I cannot thank enough Daniele for welcoming me in the AMI lab and boosting my research activity with all the available resources. You have constantly been looking ahead, coming up with new ideas that often pushed me out of my local minimum.

A special thank goes to Raffaello and Stefano for their careful supervision. Your suggestions saved me days and your dedication was pivotal in my time of need.

I also wish to thank Prof. Van de Panne and Dr. Nori for having enthusiastically served as thesis reviewers. The feedback you provided me was truly valuable.

Finally, let me thank all the current and former members of the AMI lab. Silvio, Diego, Lorenzo R., Prash, Gabri, Claudia, Moe, Kourosh, Yeshi, Valentino, Punith, Fabio B., Gianluca, Gianmarco, Hosam, Evie, Cheng, Francesca, Marta, Guglielmo, Roberto, Dario, Ehsan, Lorenzo M., Affaf, Giovanni, Italo, Enrico, Marco, Nuno, Tong, Milad, Prajval, Riccardo Z., Adi, Andrea, Emilio, Venus, Fabio D., Anqing, Filippo, Saverio, Gabriele. I had the pleasure to work with some of you, and it was exciting. The kindness that you each demonstrate keeps making my day in the lab.

Un grazie, dal profondo del cuore, va tutti coloro che continuano a farmi costantemente emozionare.

Ai Baciotti et al. Giuse, Ines, Giulio, Anto, Rick, Carlotta, Marco, Yle, Rore: riuscite a creare un'atmosfera magica, sfornando sorrisi che non hanno prezzo.

Ai Pirati. Sia in campo che fuori, siete una squadra fortissimi!

Ai ragazzi "di Roma". Trap, Rob, Vincenzo, Giuseppe: ci eravamo abituati ad una quotidianità che manca terribilmente, rivedervi è ogni volta una gioia immensa.

To the FTW group. Xavi, Vic, Jenn, Chris, Iwan, Adrien, Yuka, Misato, Hinata, Tomas, Colby: it is just unreal how so many different paths touched each other and bonded like this.

Ai ragazzi del Circolo e del Comitato. Il vostro impegno nell'agire divertendosi è una perenne fonte d'ispirazione.

Ai miei amici di una vita. Vincenzo, Marco, Nicola, Valentino, Emanuele. La naturalezza con cui un minuto insieme spazza via mesi di distanza è indescrivibile. E a Rosangela, il cui animo non smette mai di sorprendermi.

A mio fratello Nicola. Quando hanno fatto le squadre ci hanno messi insieme, ed è incredibilmente bello sapere di poterti sempre passare la palla.

A mia madre Rosa e a mio padre Felice. Il vostro affetto mi sostiene e il vostro esempio mi fortifica.

Ai miei nonni Nicola, Nina, Angelina e Giuseppe. La dolcezza delle vostre carezze è il più intimo dei conforti.

Contents

List of Figures	ix
List of Tables	x
Notation	xi
Prologue	1
I Background & Fundamentals	7
1 Modeling of Floating-Base Robots	8
1.1 Modeling of rigid body systems	8
1.1.1 Rigid body transform	9
1.1.2 Rigid body velocity	10
1.1.3 Force applied to a rigid body	14
1.2 Modeling of multi-body system	15
1.2.1 Multi-body kinematics	19
1.2.2 Multi-body dynamics	21
1.2.3 Centroidal dynamics	22
1.3 Simplified models	23
1.3.1 Linear inverted pendulum	23
1.3.2 Zero moment point	25
1.3.3 Divergent component of motion	26
2 Robot Learning	28
2.1 Supervised deep learning	28
2.1.1 Feedforward neural networks	29
2.1.2 Neural networks training	33
2.1.3 Backpropagation algorithm	37
2.2 Reinforcement learning	40
2.2.1 RL basics	40
2.2.2 Markov decision processes	45
2.2.3 Policy gradient methods	47

3	State of the Art and Thesis Context	52
3.1	State of the art in Bipedal Locomotion	52
3.1.1	Trajectory optimization layer	53
3.1.2	Simplified model control layer	55
3.1.3	Whole-body control layer	56
3.1.4	Reinforcement learning based approaches	57
3.2	State of the art in Character Animation	58
3.2.1	Kinematic motion synthesis	58
3.2.2	Physics-based motion synthesis	62
3.3	Thesis context	63
3.4	The iCub humanoid robot	65
3.4.1	The iCub v2.7 humanoid	66
3.4.2	The iCub v3 humanoid	67
3.4.3	Software infrastructure	69
II	Contribution	70
4	Learning Whole-Body Push-Recovery Strategies	71
4.1	Environment	72
4.1.1	State definition	72
4.1.2	Reward shaping	74
4.1.3	Episode specifications	79
4.2	Agent	80
4.2.1	Action definition	80
4.2.2	Policy representation and training	81
4.3	Results	82
4.3.1	Reward shaping	82
4.3.2	Emerging behaviours	84
4.3.3	Push-recovery performances	86
4.4	Conclusions	89
5	Learning Human-Like Whole-Body Trajectory Generators	90
5.1	Background	91
5.1.1	Whole-body Geometric Retargeting	91
5.1.2	Mode-Adaptive Neural Networks	92
5.2	Dataset collection	95
5.2.1	Motion capture dataset	96
5.3	Retargeting	96
5.3.1	Kinematically-feasible base motion retargeting	97
5.4	Trajectory generation	98
5.4.1	Features extraction	98
5.4.2	User input processing	101
5.5	Results	104
5.5.1	Learned walking patterns	104

5.5.2	Human-likeness	107
5.6	Conclusions	109
6	Human-Like Whole-Body Control of Humanoid Robots	110
6.1	Background	111
6.1.1	Trajectory optimization layer	111
6.1.2	Simplified model control layer	114
6.1.3	Whole-body QP control layer	115
6.2	Trajectory control	117
6.3	Crouching ability	119
6.3.1	Dataset collection and retargeting	119
6.3.2	Trajectory generation and control	120
6.4	Results	121
6.4.1	Controlled walking patterns	122
6.4.2	Transferability on a different platform	124
6.4.3	Robustness analysis	126
6.4.4	Human-likeness	133
6.4.5	Crouching	135
6.5	Conclusions	137
	Epilogue	138
	Bibliography	139
	Appendices	161
	A Texture Task for the ANA Avatar XPRIZE Finals	162

List of Figures

1	Humanoid robots developed worldwide	2
1.1	Mixed reference frame	12
1.2	Multi-body system model	16
1.3	Multi-body system as a graph	16
1.4	LIP model representation	23
2.1	Artificial neuron	29
2.2	Common activation functions	30
2.3	Feedforward neural network	31
2.4	Gradient-based optimization	35
2.5	Agent-environment interaction loop in RL	40
3.1	Hierarchical humanoid control architecture	53
3.2	Walking pattern generation example	56
3.3	Taxonomy of character animation literature	59
3.4	Examples of character animation systems	61
3.5	The iCub v2.7 and iCub v3 humanoids	66
3.6	Force-torque and inertial sensors on iCub v2.7	67
3.7	Comparison between iCub v2.7 and iCub v3	68
4.1	Detailed agent-environment interaction loop	72
4.2	Learning curves	82
4.3	Individual reward components evolution	83
4.4	Push-recovery emerging behaviours	84
4.5	Push-recovery performances for deterministic planar forces	86
4.6	Push-recovery performances for random spherical forces	88
5.1	Human-robot frame correspondences for WBGR	92
5.2	MANN architecture	94
5.3	Dataset collection	95
5.4	MANN features visualization	100
5.5	User input processing	102
5.6	MANN-generated forward walking	105
5.7	MANN-generated backward walking	105
5.8	MANN-generated left-side walking	105
5.9	MANN-generated right-side walking	105
5.10	MANN-generated mixed walking	106

5.11	Blending coefficients activation	107
5.12	Human-likeness of generated trajectories	108
6.1	ADHERENT architecture	112
6.2	DCM trajectory planning	113
6.3	Dataset collection for crouching motions	119
6.4	Trajectory control on iCub v2.7	123
6.5	Trajectory control on iCub v3	125
6.6	Robustness analysis of ADHERENT on iCub v2.7	127
6.7	ZMP, DCM and CoM tracking – forward walking on iCub v2.7 . . .	128
6.8	ZMP, DCM and CoM tracking – left-side walking on iCub v2.7 . . .	129
6.9	Robustness analysis of ADHERENT on iCub v3	130
6.10	ZMP, DCM and CoM tracking – backward walking on iCub v3 . . .	131
6.11	ZMP, DCM and CoM tracking – right-side walking on iCub v3 . . .	132
6.12	Human-likeness of controlled trajectories for iCub v2.7	134
6.13	Human-likeness of controlled trajectories for iCub v3	134
6.14	Crouching-to-upright walking transition on simulated iCub v2.7 . . .	135
6.15	Blending coefficients activation while crouching	135
A.1	The iCub v3 sensorized hand	163
A.2	Data collection for the rock classifier	163
A.3	Rock classifier input	165
A.4	Rock classifier performances	165

List of Tables

4.1	Observation components	73
4.2	Reward components	77
4.3	Training parameters	81
5.1	Motion capture dataset specifications	96
6.1	Crouching motion capture dataset specifications	119
6.2	Nominal and experimental velocity and step size for iCub v2.7 . . .	127
6.3	Nominal and experimental velocity and step size for iCub v3 . . .	130
6.4	Crouching features comparison	136

Notation

The following notation will be adopted throughout the thesis.

- The i -th component of a vector x is denoted by x_i .
- The transpose operator is denoted by $(\cdot)^\top$.
- Given a function of time $f(t)$, the dot notation denotes the time derivative, i.e., $\dot{f} := \frac{df}{dt}$. Higher-order derivatives are denoted by a corresponding amount of dots.
- $I_n \in \mathbb{R}^{n \times n}$ denotes the identity matrix of dimension n .
- $0_{n \times m} \in \mathbb{R}^{n \times m}$ denotes a zero matrix of dimension $n \times m$, while $0_n = 0_{n \times 1}$ is a zero column vector of dimension n .
- e_i is the canonical base in \mathbb{R}^n , i.e., $e_i = [0, \dots, 0, 1, 0, \dots, 0]^\top \in \mathbb{R}^n$, where the only unitary element is in position i .
- $\mathcal{I} = (o_{\mathcal{I}}, [\mathcal{I}])$ is a fixed inertial frame with respect to which the robot's absolute pose is measured. $o_{\mathcal{I}}$ denotes the origin of the frame. Its orientation $[\mathcal{I}]$ is such that the z axis points against gravity while the x axis defines the forward direction.
- Given the inertial frame $\mathcal{I} = (o_{\mathcal{I}}, [\mathcal{I}])$ and a frame $A = (o_A, [A])$, the frame $A[\mathcal{I}] = (o_A, [\mathcal{I}])$ is defined with origin in o_A and orientation as the inertial frame.
- The operator \times defines the cross product in \mathbb{R}^3 .
- Given a vector $w \in \mathbb{R}^3$, the *hat* operator $(\cdot)^\wedge$ defines the skew-symmetric operation associated with the cross product in \mathbb{R}^3 , i.e., the skew-symmetric matrix $w^\wedge \in \mathbb{R}^{3 \times 3}$

$$w^\wedge := \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}^\wedge = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}$$

is such that, given another vector $u \in \mathbb{R}^3$, $w^\wedge u = w \times u$. Its inverse is the *vec* operator $(\cdot)^\vee$, meaning that

$$(w^\wedge)^\vee := \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}^\vee = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = w.$$

- Given a vector $\mathbf{v} = [v^\top, w^\top]^\top \in \mathbb{R}^6$, with $v, w \in \mathbb{R}^3$, the *hat* operator $(\cdot)^\wedge$ is defined as

$$\mathbf{v}^\wedge := \begin{bmatrix} v \\ w \end{bmatrix}^\wedge = \begin{bmatrix} w^\wedge & v \\ \mathbf{0}_{1 \times 3} & 0 \end{bmatrix}.$$

Its inverse is the *vee* operator $(\cdot)^\vee$, meaning that

$$(\mathbf{v}^\wedge)^\vee := \begin{bmatrix} w^\wedge & v \\ \mathbf{0}_{1 \times 3} & 0 \end{bmatrix}^\vee = \begin{bmatrix} v \\ w \end{bmatrix} = \mathbf{v}.$$

- The set $\text{SO}(3)$, i.e., Special Orthogonal Group in three dimensions, is the set of $\mathbb{R}^{3 \times 3}$ orthogonal matrices with determinant equal to one, namely

$$\text{SO}(3) := \left\{ R \in \mathbb{R}^{3 \times 3} \mid R^\top R = I_3, \det(R) = 1 \right\}.$$

- The set $\mathfrak{so}(3)$ is the set of 3D skew-symmetric matrices

$$\mathfrak{so}(3) := \left\{ S \in \mathbb{R}^{3 \times 3} \mid S^\top = -S \right\}.$$

- The set $\text{SE}(3)$, i.e., Special Euclidean Group in three dimensions, is defined as

$$\text{SE}(3) := \left\{ \begin{bmatrix} R & p \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 4} \mid R \in \text{SO}(3), p \in \mathbb{R}^3 \right\}.$$

- The set $\mathfrak{se}(3)$ is defined as

$$\mathfrak{se}(3) := \left\{ \begin{bmatrix} w^\wedge & v \\ \mathbf{0}_{1 \times 3} & 0 \end{bmatrix} \in \mathbb{R}^{4 \times 4} \mid w^\wedge \in \mathfrak{so}(3), v \in \mathbb{R}^3 \right\}.$$

- ${}^A R_B \in \text{SO}(3)$ and ${}^A H_B \in \text{SE}(3)$ denote the rotation and transformation matrices that transform a vector ${}^B x$ expressed in the frame B into a vector ${}^A x$ expressed in the frame A .
- ${}^C v_{A,B} \in \mathbb{R}^6$ is the 6D velocity (aka twist) of frame B with respect to frame A , whose coordinates are expressed in frame C .
- ${}^A f_F \in \mathbb{R}^6$ denotes the 6D force (aka wrench) applied in frame F , whose coordinates are expressed in frame A .
- $x_{\text{CoM}} \in \mathbb{R}^3$ is the center of mass (CoM) position expressed in \mathcal{I} .

Prologue

As of May 2023, the humanoid robot Atlas¹ by Boston Dynamics has been shown to grasp, carry and promptly deliver, while climbing stairs and jumping with stunning agility, a heavy bag of tools to a worker who asked for it². The humanoid robot Ameca³ by Engineered Arts has been demonstrated to perform lively gestures and facial expressions which impressively mimic the human ones⁴. While these robots, as many others, already exhibit unique physical and social interaction skills, new humanoids with an even higher degree of autonomy such as Tesla Bot⁵ by Tesla and CyberOne⁶ by Xiaomi have been announced. With a global market valued at USD 1.5 billion in 2022 and estimated to reach USD 17.3 billion by 2027⁷, humanoid robotics cannot be considered anymore as a field of purely prototyping research but gets closer and closer to industry and broader commercialization, with humans and humanoids more likely to coexist as time goes by.

However, long-lasting research in humanoid robotics lies behind today's exciting achievements. Although the idea of building anthropomorphic machines has deep philosophical roots dating back centuries, a starting point for humanoid robotics research can be identified with the presentation of the WABOT-1 humanoid by Waseda University in 1973. Since then, a large amount of robots with human-like resemblance have appeared worldwide. Just to mention some: P2 and ASIMO by Honda, TORO by the German Aerospace Center, Valkyrie by the National Aeronautics and Space Administration agency of the United States Government, and more recently Digit by Agility Robotics and TALOS by PAL Robotics. Initially developed, as any robot, to replace people at tedious and tiring tasks, humanoids gained over time a role that goes far beyond the mere execution of repetitive tasks. Structurally suited for human-centered environments, they are now designed also to assist humans and collaborate with them both in workplaces and domestic settings. They are expected to operate in disaster scenarios, autonomously or under remote control, in place of humans currently risking their lives for rescue operations. Even more generally, they could represent physical avatars enabling human beings to exist, through them, in a remote location, and be able to physically interact with the remote environment as if they were actually there.

¹<https://www.bostondynamics.com/atlas>

²https://www.youtube.com/watch?v=-e1_QhJ1EhQ

³<https://www.engineeredarts.co.uk/robot/ameca/>

⁴https://vimeo.com/651929733?embedded=true&source=vimeo_logo&owner=63229520

⁵https://www.youtube.com/live/ODSJsviD_SU?feature=share&t=836

⁶<https://www.youtube.com/watch?v=CJhneBJIf0k>

⁷www.marketsandmarkets.com/Market-Reports/humanoid-robot-market-99567653.html

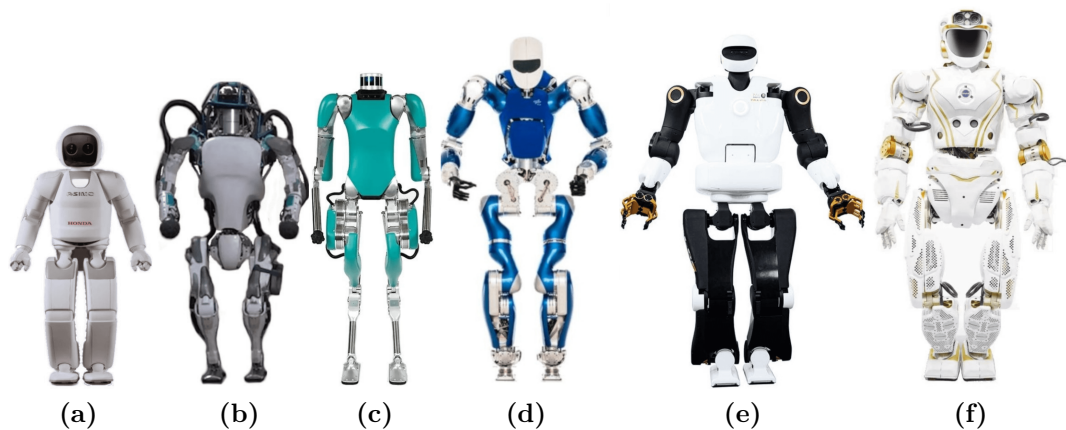


Figure 1. Examples of humanoids developed worldwide, by increasing size. (a) ASIMO: 130 cm, 48 kg, 2011. (b) Atlas: 150 cm, 75 kg, 2016. (c) Digit: 155 cm, 42.2 kg, 2019. (d) TORO: 174 cm, 76.4 kg, 2014. (e) TALOS: 175 cm, 95 kg, 2017. (f) Valkyrie: 187 cm, 129 kg, 2013. Images and specifications taken from [Ficht and Behnke, 2021].

The technological progress required even just to imagine such advanced applications for humanoid robots was driven by several factors. International challenges were definitely one of the key factors that, besides pushing technology, also highlighted its limitations. From 2012 to 2015, the DARPA Robotics Challenge (DRC), funded by the US Defense Advanced Research Projects Agency, promoted the usage of humanoids in disaster scenarios. At the final event, tasks such as driving a utility vehicle, walking through a door, and manipulating a tool to break through a concrete panel were tested in a scenario emulating a nuclear disaster. During the trials, a series of iconic failures highlighted the unripeness of humanoid robotics. For instance, the Running Man humanoid fell while waving to the crowd after having successfully completed all the required tasks. From 2018 to 2022, the ANA Avatar XPRIZE, funded by All Nippon Airways, fostered the usage of robots as avatar systems transporting human presence to a remote location in real time. Despite humanoids being arguably one of the most natural solutions to such a challenge (referred to as telepresence), only two humanoids qualified for the finals, confirming therefore the need for further technological development in the field in order to approach the full exploitation of humanoid robot capabilities.

No matter the application, whether it is domestic assistance, physical human-robot collaboration or telepresence, as well as many others, an ubiquitous task for the humanoid that cannot be avoided is locomotion. Nowadays, *humanoid locomotion* is far from being a trivial task, especially when considered as a general problem, whose solution is not meant to rely on the peculiar strengths of the specific robot used as a development platform. In the case of Atlas, for instance, the locomotion algorithms are designed hand-in-hand with the robot hardware, and the hardware features (e.g., compact and lightweight mechanical components with optimized shapes, powerful hydraulic power unit and battery, etc.) play a fundamental role for the outstanding achieved performances⁸.

⁸<https://slideslive.com/38946802/boston-dynamics>

As regards general approaches to humanoid locomotion, probably the best example so far is given by the model-based hierarchical control architecture left as inheritance by the DRC. Composed by several layers, this architecture copes with the high complexity of the humanoid locomotion problem by splitting it into subproblems, and taking a series of simplifying assumptions when modeling the humanoid (different for each layer) which make the problem tractable online. Using simplified models for the humanoid allows for remarkable results. However, this choice collides with the objective of fully exploiting the humanoid complex mechanical structure, and constrains the set of attainable behaviours. As a result, the robot locomotion patterns obtained by following this approach are often far from those demonstrated by humans. In other words, they cannot be said to exhibit a high degree of *human-likeness*, a property that is definitely not required (and, indeed, often neglected) to walk while maintaining balance, but gains prominence if the robot motion needs to be predictable and interpretable for humans. For instance, human-likeness facilitates physical human-robot interactions, and makes social human-robot interactions more natural. More generally, it increases the social acceptability of robots, paving the way for their integration in our everyday life.

Another general approach to humanoid locomotion, from a completely different perspective, consists in tackling this problem using machine learning techniques. More specifically, deep learning methods which allow to cope with the high-dimensional continuous space of joint references (i.e., positions, velocities or torques, depending on the control mode) used to control the humanoid. Among the different machine learning paradigms, reinforcement learning appears as a natural choice. The agent-environment interaction characterizing reinforcement learning is indeed by definition particularly suitable for robotics, since a robot is probably the most intuitive instance of an agent interacting with its surrounding environment one can think of. Also supervised (imitation) learning can play a role in addressing humanoid locomotion. The way a human walks is arguably the way we would like the humanoids around us to walk. Expert demonstrations collected from human subjects (e.g., using motion tracking systems) represent therefore useful sources of information for the humanoid to learn the complex locomotion task from. Moreover, human-collected demonstrations are whole-body (i.e., they provide a reference also for the redundant degrees of freedom of the humanoid with respect to the locomotion task) and, by definition, human-like.

In this thesis, we investigate the application of learning-based techniques to planning and control of humanoid locomotion. In doing so, we exploit both deep reinforcement learning and deep supervised learning. The former approach allows us to study the spontaneous emergence of locomotion behaviours (specifically, balancing and push-recovery strategies). The latter instead, by making use of motion capture data collected from humans, enables us to shape the robot locomotion patterns (specifically, walking trajectories) towards an improved human-likeness. We strongly believe that such model-free approaches, based on deep neural networks, have the potential to properly represent the complexity of the humanoid while taking full advantage of its redundancy with respect to the locomotion task. Moreover, they are remarkably efficient and particularly suitable to generalize from human demonstration. All these features, combined together, motivate us in the choice of learning-based methods to tackle humanoid locomotion.

This thesis is divided into two parts, briefly summarized in the following along with the content of each chapter.

Part I: Background & Fundamentals

This part provides the reader with the fundamental background about the concepts exploited in the thesis, and reviews the state-of-the-art relevant for the thesis context.

- Chapter 1 describes the kinematics and dynamics modeling of humanoid robots, along with some simplified models commonly adopted for bipedal locomotion.
- Chapter 2 introduces the basics of deep supervised learning and deep reinforcement learning.
- Chapter 3 collects a concise overview of the relevant state-of-the-art related to the fields of bipedal locomotion and character animation, and defines the thesis context besides presenting the humanoid robots used for the experimental validation of the thesis developments.

Part II: Contribution

This part presents the learning-based methods developed as contributions of this thesis, along with their validation on simulated and real robots.

- Chapter 4 illustrates the design of a control policy for balancing and push recovery of humanoid robots, trained using deep reinforcement learning. In our setting, the agent controls joint velocities of both the lower and upper body joints. We define the reward so to encourage the emergence of transient push-recovery strategies as well as steady-state balancing. The policy is validated on a simulated version of the iCub v2.7 humanoid.
- Chapter 5 presents a data-driven whole-body trajectory generator for humanoid locomotion, trained by deep supervised learning. Our trajectory generator adapts a peculiar deep network architecture, proved effective for the animation of quadruped characters in computer graphics, to the generation of walking patterns for humanoid robots. The kinematic validation of the proposed approach is conducted on the iCub v2.7 humanoid model.
- Chapter 6 introduces an end-to-end architecture, named ADHERENT, resulting from the integration of the learning-based trajectory generator presented in Chapter 5 with a state-of-the-art control architecture for humanoid locomotion. In particular, the footstep plan and the human-like postural produced by the learning-based trajectory generator are exploited for trajectory control. The ADHERENT architecture is experimentally validated on both the iCub v2.7 and the iCub v3 humanoid robots.

This research work has been conducted during my tenure as Ph.D. candidate in the *Artificial and Mechanical Intelligence* laboratory at the *Istituto Italiano di Tecnologia* in Genoa, Italy. The doctoral program has been carried out in accordance with the requirements of *Sapienza University of Rome*, Italy, in order to obtain a Ph.D. title.

Summary of publications

The results of the research carried out for this thesis have been (or will be) disseminated in peer-reviewed research publications. For each publication, we mention additional material such as video presentation and released code when available.

The content of Chapter 4 partially appears in:

Ferigo, D., Camoriano, R., Viceconte, P. M., Calandriello, D., Traversaro, S., Rosasco, L., and Pucci, D. (2021). On the Emergence of Whole-body Strategies from Humanoid Robot Push-recovery Learning. *IEEE Robotics and Automation Letters*, 6(4):8561–8568.

Video <https://www.youtube.com/watch?v=FaOMtfYZiGA>

My personal contribution to this work primarily regards the definition of the reinforcement learning setting. I have been actively involved in the search, inspired by domain knowledge on humanoid control and led through an iterative process, for an action (Section 4.2.1), state (Section 4.1.1) and reward (Section 4.1.2) suitable for learning the task at issue. As regards the evaluation of the policy performances after training, I conceived, run and analyzed a relevant portion of the simulations involving out-of-sample scenarios to demonstrate the policy robustness and generalization capabilities (Section 4.3.3). I was not the main developer of the training infrastructure, and in particular I did not directly deal with the episodes characterization (Section 4.1.3) nor the implementation of the distributed training setup (Section 4.2.2).

The content of Chapter 5 and Chapter 6 partially appears in:

Viceconte, P. M., Camoriano, R., Romualdi, G., Ferigo, D., Dafarra, S., Traversaro, S., Oriolo, G., Rosasco, L., and Pucci, D. (2022). ADHERENT: Learning Human-like Trajectory Generators for Whole-body Control of Humanoid Robots. *IEEE Robotics and Automation Letters*, 7(2):2779–2786.

Video <https://www.youtube.com/watch?v=s7-pML0ojK8>

Github [ami-iit/paper_viceconte_2021_ral_adherent](https://github.com/ami-iit/paper_viceconte_2021_ral_adherent)

Zenodo https://zenodo.org/record/6201915#.YhOT_Bso-8g

The content of Sections 6.3 and 6.4.5 will be submitted for a publication tentatively titled as:

D’Elia, E., Viceconte, P. M., Rapetti, L., and Pucci, D. Learning Human-like Trajectory Generators for Humanoid Robot Locomotion with Crouching Abilities. (To be submitted).

Video <https://www.youtube.com/watch?v=Dor1hMqAAmo>

My personal contribution to this work regards the conceptualization and definition of the methodology to be followed for the extension of the ADHERENT framework to deal with crouching motions (Section 6.3). I actively contributed to the validation of the proposed approach (Section 6.4.5). Besides partially contributing to the software implementation, I was not the main developer of the software for this work.

Apart from the publications directly relevant to the contributions of this thesis, the content of Appendix A is under review in:

Dafarra, S., Pattacini, U., Romualdi, G., Rapetti, L., Grieco, R., Darvish, K., Milani, G., Valli, E., Sorrentino, I., Viceconte, P. M., Scalzo, A., Traversaro, S., Sartore, C., Elobaid, M., Guedelha, N., Herron, C., Leonessa, A., Metta, G., Maggiali, M., and Pucci, D. (2023). iCub3 Avatar System. *Science Robotics* (Submitted, Under Review).

Preprint <https://arxiv.org/pdf/2203.06972.pdf>

My main contribution for this work was the development of the algorithm for the texture classification task for the finals of the ANA Avatar XPRIZE competition, included among the validation scenarios of the proposed iCub3 avatar system. Details on the proposed solution to address the texture classification task are included in the Appendix A.

Part I
Background & Fundamentals

Chapter 1

Modeling of Floating-Base Robots

In this chapter, we focus on the mathematical description of floating-base robots, considered as systems of multiple interconnected rigid bodies. The kinematic and dynamical models of floating-base robots introduced in this chapter are essential for understanding the planners and controllers presented in Part II. Most of the material presented in this chapter is based on [Siciliano et al., 2008; Featherstone, 2014; Traversaro, 2017; Marsden and Ratiu, 2010; Orin et al., 2013; Kajita et al., 2001, 2003; Vukobratović and Borovac, 2004; Engelsberger et al., 2013, 2015]. More in detail, the chapter is organized as follows.

We start from the modeling of rigid bodies, in terms of their associated reference frames, in Section 1.1. Homogeneous transformations to describe rigid body poses are introduced in Section 1.1.1. Section 1.1.2 deals with different representations of rigid body velocities defined as 6D vectors. In Section 1.1.3, 6D force vectors describing the interaction of rigid bodies with the environment are presented.

Using concepts from graph theory, multi-body systems composed of links interconnected by joints are then modeled in Section 1.2. The forward and differential kinematics of the multi-body system are presented in Section 1.2.1. Section 1.2.2 illustrates the equations of motion describing the multi-body dynamics. Such dynamics projected at the robot center of mass is discussed in Section 1.2.3.

Finally, some approximations of the robot dynamics and related concepts widely used for legged locomotion are introduced in Section 1.3. In particular, the linear inverted pendulum model is illustrated in Section 1.3.1 and the zero moment point in Section 1.3.2, while Section 1.3.3 presents the divergent component of motion.

1.1 Modeling of rigid body systems

A body is said to be *rigid* if it is not subject to internal deformation under the action of external disturbances, i.e., if the distance between each pair of points of the body remains constant no matter the external forces or moments acting on it. Despite being an idealization of how objects behave in the real world, the rigid body assumption underlies the study of most mechanical systems, robots included.

To describe the motion of a rigid body in space, it is convenient to define a *reference frame* $A = (o_A, [A])$ attached to the rigid body, characterized by a point o_A called origin and a 3D orientation frame $[A]$ with an orthonormal basis. This facilitates the description of rigid body motions both relative to each other and expressed with respect to the so-called *inertial frame* $\mathcal{I} = (o_{\mathcal{I}}, [\mathcal{I}])$: a special reference frame assumed fixed on the Earth's surface and therefore, disregarding non-inertial effects due to the Earth's motion, not undergoing any acceleration. In the following, we will often refer to rigid bodies directly in terms of their associated frames.

1.1.1 Rigid body transform

Let us consider two frames: $A = (o_A, [A])$ and $B = (o_B, [B])$. If r_{o_A, o_B} is the 3D vector connecting o_A with o_B , pointing towards o_B , the *position* of frame B with respect to frame A is defined by the *coordinates* of its origin o_B expressed in A , i.e.,

$${}^A o_B = \begin{bmatrix} r_{o_A, o_B} \cdot x_A \\ r_{o_A, o_B} \cdot y_A \\ r_{o_A, o_B} \cdot z_A \end{bmatrix}, \quad (1.1.1)$$

where (\cdot) denotes the scalar product between vectors and x_A, y_A, z_A are the unit vectors constituting the orthonormal basis of $[A]$.

Rotation matrix

Let us assume coincident origins $o_A = o_B$ for the frames A and B . If the unit vectors ${}^A x_B, {}^A y_B$ and ${}^A z_B$ denote the orthonormal basis of $[B]$ expressed in frame A , the *orientation* of frame B with respect to frame A is expressed by the *rotation matrix* ${}^A R_B \in \text{SO}(3)$ defined as

$${}^A R_B = [{}^A x_B \quad {}^A y_B \quad {}^A z_B]. \quad (1.1.2)$$

Given a 3D vector ${}^B p$ whose coordinates are expressed in frame B , the rotation matrix ${}^A R_B$ also allows to express the vector coordinates in frame A as

$${}^A p = {}^A R_B {}^B p, \quad (1.1.3)$$

and represents therefore the coordinate transformation from frame B to A .

Homogeneous transformation

Let us relax the assumption of frames A and B having coincident origins. Then, the coordinate transformation of ${}^B p$ into frame A is given by

$${}^A p = {}^A R_B {}^B p + {}^A o_B, \quad (1.1.4)$$

or, in matrix representation, by

$$\begin{bmatrix} {}^A p \\ 1 \end{bmatrix} = \begin{bmatrix} {}^A R_B & {}^A o_B \\ 0_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} {}^B p \\ 1 \end{bmatrix}. \quad (1.1.5)$$

If ${}^A \tilde{p} := [{}^A p^\top \ 1]^\top \in \mathbb{R}^4$ and ${}^B \tilde{p} := [{}^B p^\top \ 1]^\top \in \mathbb{R}^4$ are the homogeneous representations of ${}^A p$ and ${}^B p$, respectively, then Eq. (1.1.5) assumes the more compact form

$${}^A \tilde{p} = {}^A H_B {}^B \tilde{p}, \quad (1.1.6)$$

where ${}^A H_B \in \text{SE}(3)$ is referred to as *homogeneous transformation* from B to A . Besides mapping homogeneous representations of vectors, homogeneous transformations provide a compact representation of the position-and-orientation, i.e. *pose* [Siciliano et al., 2008], of a frame with respect to another.

Given a homogeneous transformation ${}^A H_B$, it can be easily shown, by premultiplying both sides of Eq. (1.1.4) with ${}^A R_B^\top$ and exploiting the orthogonality properties of rotation matrices, that its inverse ${}^B H_A$ is equal to

$${}^B H_A = {}^A H_B^{-1} = \begin{bmatrix} {}^B R_A & {}^B o_A \\ 0_{1 \times 3} & 1 \end{bmatrix} = \begin{bmatrix} {}^A R_B^\top & -{}^A R_B^\top {}^A o_B \\ 0_{1 \times 3} & 1 \end{bmatrix}. \quad (1.1.7)$$

1.1.2 Rigid body velocity

The 6D velocity (aka twist) of a frame $v_{A,B} \in \mathbb{R}^6$ can be represented in several ways, depending on the frame with respect to which the velocity coordinates are expressed. Such different representations, also referred to as *trivializations*, arise from manipulating the time derivative of the homogeneous transformation ${}^A H_B$, i.e.,

$${}^A \dot{H}_B = \frac{d}{dt}({}^A H_B) = \begin{bmatrix} {}^A \dot{R}_B & {}^A \dot{o}_B \\ 0_{1 \times 3} & 0 \end{bmatrix}. \quad (1.1.8)$$

Before introducing different 6D velocity trivializations, let us express the term ${}^A \dot{R}_B$ in Eq. (1.1.8) in terms of the angular velocity of frame B relative to frame A expressed either in A , i.e., ${}^A \omega_{A,B}$, or in B , i.e., ${}^B \omega_{A,B}$. Given the rotation matrix ${}^A R_B$ and its derivative ${}^A \dot{R}_B$, by differentiation of the $\text{SO}(3)$ orthogonality conditions

$${}^A R_B {}^A R_B^\top = I_3, \quad (1.1.9)$$

$${}^A R_B^\top {}^A R_B = I_3, \quad (1.1.10)$$

we obtain

$${}^A \dot{R}_B {}^A R_B^\top + {}^A R_B {}^A \dot{R}_B^\top = 0_{3 \times 3}, \quad (1.1.11)$$

$${}^A \dot{R}_B^\top {}^A R_B + {}^A R_B^\top {}^A \dot{R}_B = 0_{3 \times 3}, \quad (1.1.12)$$

which can be rearranged as

$${}^A \dot{R}_B {}^A R_B^\top = - \left({}^A \dot{R}_B {}^A R_B^\top \right)^\top, \quad (1.1.13)$$

$${}^A R_B^\top {}^A \dot{R}_B = - \left({}^A R_B^\top {}^A \dot{R}_B \right)^\top, \quad (1.1.14)$$

where both ${}^A \dot{R}_B {}^A R_B^\top \in \mathfrak{so}(3)$ and ${}^A R_B^\top {}^A \dot{R}_B \in \mathfrak{so}(3)$ are 3D skew-symmetric matrices, different from each other because of the matrix product being not commutative, that can be rewritten as applications of the operator $(\cdot)^\wedge$ on two different 3D vectors, i.e.,

$${}^A \dot{R}_B {}^A R_B^\top = {}^A \omega_{A,B}^\wedge, \quad (1.1.15)$$

$${}^A R_B^\top {}^A \dot{R}_B = {}^B \omega_{A,B}^\wedge. \quad (1.1.16)$$

Therefore, it follows that

$${}^A \dot{R}_B = {}^A \omega_{A,B}^\wedge {}^A R_B = {}^A R_B {}^B \omega_{A,B}^\wedge, \quad (1.1.17)$$

where the time derivative ${}^A \dot{R}_B$ of the rotation matrix ${}^A R_B$ is indeed expressed in terms of the rotation matrix itself and of the angular velocity of frame B with respect to frame A expressed either in A , i.e., ${}^A \omega_{A,B}$, or in B , i.e., ${}^B \omega_{A,B}$.

Left trivialized velocity

Let us focus on Eq. (1.1.8) again. A more compact representation of ${}^A\dot{H}_B$ can be obtained by premultiplying it with the inverse of ${}^A H_B$ from Eq. (1.1.7), i.e.,

$$\begin{aligned} {}^A H_B^{-1} {}^A \dot{H}_B &= \begin{bmatrix} {}^A R_B^\top & -{}^A R_B^\top {}^A o_B \\ 0_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} {}^A \dot{R}_B & {}^A \dot{o}_B \\ 0_{1 \times 3} & 0 \end{bmatrix} \\ &= \begin{bmatrix} {}^A R_B^\top {}^A \dot{R}_B & {}^A R_B^\top {}^A \dot{o}_B \\ 0_{1 \times 3} & 0 \end{bmatrix}. \end{aligned} \quad (1.1.18)$$

By defining ${}^B v_{A,B} \in \mathbb{R}^3$ so that

$${}^B v_{A,B} := {}^A R_B^\top {}^A \dot{o}_B, \quad (1.1.19)$$

and recalling the definition of ${}^B \omega_{A,B}$ so that Eq. (1.1.16) holds, the *left trivialized* velocity of frame B with respect to frame A is defined by

$${}^B \mathbf{v}_{A,B} := \begin{bmatrix} {}^B v_{A,B} \\ {}^B \omega_{A,B} \end{bmatrix} \in \mathbb{R}^6, \quad (1.1.20)$$

where ${}^B v_{A,B}$ and ${}^B \omega_{A,B}$ are the *linear* and *angular* components of ${}^B \mathbf{v}_{A,B}$, respectively. From Eq. (1.1.20) it follows, by construction, that

$${}^B \mathbf{v}_{A,B}^\wedge = {}^A H_B^{-1} {}^A \dot{H}_B \in \mathfrak{se}(3). \quad (1.1.21)$$

If the frame B is attached to the moving rigid body and the frame A is the inertial frame, the left trivialized velocity is also called *body velocity*, since the coordinates of both its linear and angular components are expressed in the moving frame B .

Right trivialized velocity

Similarly to Eq. (1.1.18), by multiplying ${}^A \dot{H}_B$ times the inverse of ${}^A H_B$, one has

$$\begin{aligned} {}^A \dot{H}_B {}^A H_B^{-1} &= \begin{bmatrix} {}^A \dot{R}_B & {}^A \dot{o}_B \\ 0_{1 \times 3} & 0 \end{bmatrix} \begin{bmatrix} {}^A R_B^\top & -{}^A R_B^\top {}^A o_B \\ 0_{1 \times 3} & 1 \end{bmatrix} \\ &= \begin{bmatrix} {}^A \dot{R}_B {}^A R_B^\top & {}^A \dot{o}_B - {}^A \dot{R}_B {}^A R_B^\top {}^A o_B \\ 0_{1 \times 3} & 0 \end{bmatrix}. \end{aligned} \quad (1.1.22)$$

By defining ${}^A v_{A,B} \in \mathbb{R}^3$ so that

$${}^A v_{A,B} := {}^A \dot{o}_B - {}^A \dot{R}_B {}^A R_B^\top {}^A o_B, \quad (1.1.23)$$

and by recalling the definition of ${}^A \omega_{A,B}$ so that Eq. (1.1.15) holds, the *right trivialized* velocity of frame B with respect to frame A is defined by

$${}^A \mathbf{v}_{A,B} := \begin{bmatrix} {}^A v_{A,B} \\ {}^A \omega_{A,B} \end{bmatrix} \in \mathbb{R}^6, \quad (1.1.24)$$

where ${}^A v_{A,B}$ and ${}^A \omega_{A,B}$ are the *linear* and *angular* components of ${}^A \mathbf{v}_{A,B}$, respectively. From Eq.(1.1.24) it follows, by construction, that

$${}^A \mathbf{v}_{A,B}^\wedge = {}^A \dot{H}_B {}^A H_B^{-1} \in \mathfrak{se}(3). \quad (1.1.25)$$

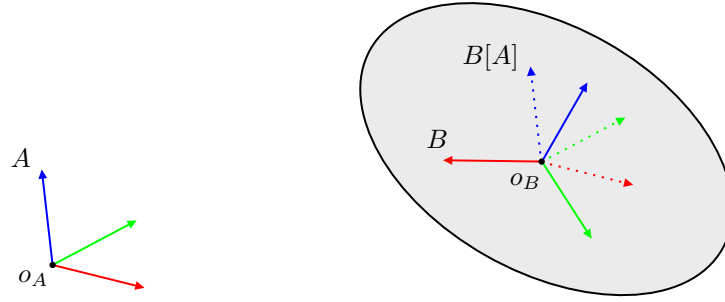


Figure 1.1. Representation of the mixed reference frame $B[A]$, shown in dotted lines, having the same origin o_B of frame B and the same orientation $[A]$ of frame A . The orthonormal x , y and z axes of each frame are depicted in red, green and blue, respectively.

If the frame B is attached to the moving rigid body and the frame A is the inertial frame, the right trivialized velocity is also called *inertial velocity*, since the coordinates of both its linear and angular components are expressed in the inertial frame A .

Notice that, by exploiting Eq. (1.1.15) and the skew-symmetric nature of ${}^A\omega_{A,B}^\wedge$, the linear part of the right trivialized velocity can alternatively be written as

$${}^A v_{A,B} = {}^A \dot{o}_B + {}^A o_B \wedge {}^A \omega_{A,B}. \quad (1.1.26)$$

Adjoint matrix for velocity transformations

Eq. (1.1.21) and Eq. (1.1.25) directly provide the matrix form of the mapping between the left and right trivialized velocities, i.e.,

$${}^A v_{A,B}^\wedge = {}^A H_B {}^B v_{A,B}^\wedge {}^A H_B^{-1}. \quad (1.1.27)$$

The same mapping can also be expressed, using 6D vectors, in the linear form

$${}^A v_{A,B} = {}^A X_B {}^B v_{A,B}, \quad (1.1.28)$$

where ${}^A X_B$ is usually called *adjoint* matrix and defined as follows.

First, recall the rotational equivalence of cross products, stating that, for any rotation matrix $R \in \text{SO}(3)$ and vectors $a, b \in \mathbb{R}^3$, one has that

$$(Ra) \times (Rb) = R(a \times b). \quad (1.1.29)$$

Then, let us prove that the relation

$$R a^\wedge R^\top = (Ra)^\wedge \quad (1.1.30)$$

holds, by multiplying the right-hand side term by an arbitrary vector $c \in \mathbb{R}^3$, i.e.,

$$(Ra)^\wedge c = (Ra) \times c \quad (1.1.31a)$$

$$= (Ra) \times (RR^\top c) \quad (1.1.31b)$$

$$= R(a \times (R^\top c)) \quad (1.1.31c)$$

$$= Ra^\wedge R^\top c, \quad (1.1.31d)$$

where the passage from Eq. (1.1.31b) to Eq. (1.1.31c) makes use of Eq. (1.1.29).

Finally, let us define the adjoint matrix ${}^A X_B$ by developing Eq. (1.1.27), with no superscripts and subscripts for the sake of readability, as

$${}^A v_{A,B} = \left({}^A v_{A,B}^\wedge \right)^\vee = \left({}^A H_B {}^B v_{A,B}^\wedge {}^A H_B^{-1} \right)^\vee \quad (1.1.32a)$$

$$= \left(\begin{bmatrix} R & o \\ 0_{1 \times 3} & 1 \end{bmatrix} \begin{bmatrix} \omega^\wedge & v \\ 0_{1 \times 3} & 0 \end{bmatrix} \begin{bmatrix} R^\top & -R^\top o \\ 0_{1 \times 3} & 1 \end{bmatrix} \right)^\vee \quad (1.1.32b)$$

$$= \begin{bmatrix} R\omega^\wedge R^\top & -R\omega^\wedge R^\top o + Rv \\ 0_{1 \times 3} & 0 \end{bmatrix}^\vee \quad (1.1.32c)$$

$$= \begin{bmatrix} (R\omega)^\wedge & -(R\omega)^\wedge o + Rv \\ 0_{1 \times 3} & 0 \end{bmatrix}^\vee \quad (1.1.32d)$$

$$= \begin{bmatrix} (R\omega)^\wedge & o^\wedge R\omega + Rv \\ 0_{1 \times 3} & 0 \end{bmatrix}^\vee \quad (1.1.32e)$$

$$= \begin{bmatrix} o^\wedge R\omega + Rv \\ R\omega \end{bmatrix} \quad (1.1.32f)$$

$$= \begin{bmatrix} {}^A R_B & {}^A o_B^\wedge {}^A R_B \\ 0_{3 \times 3} & {}^A R_B \end{bmatrix} \begin{bmatrix} {}^B v_{A,B} \\ {}^B \omega_{A,B} \end{bmatrix} = {}^A X_B {}^B v_{A,B}, \quad (1.1.32g)$$

where the passage from Eq. (1.1.32c) to Eq. (1.1.32d) exploits Eq. (1.1.30) and then the skew-symmetric property $a^\wedge b = -b^\wedge a$ for $a, b \in \mathbb{R}^3$ is used to get to Eq. (1.1.32e). The adjoint matrix ${}^A X_B$ thus obtained, i.e.,

$${}^A X_B = \begin{bmatrix} {}^A R_B & {}^A o_B^\wedge {}^A R_B \\ 0_{3 \times 3} & {}^A R_B \end{bmatrix} \in \mathbb{R}^{6 \times 6}, \quad (1.1.33)$$

besides mapping left to right trivialized velocities, assumes the more general meaning of velocity transformation that changes the frame in which the 6D velocity coordinates are expressed. Given a generic 6D velocity ${}^B v_{C,D}$ of frame D relative to frame C , expressed in frame B , the adjoint matrix allows indeed to express it in frame A as

$${}^A v_{C,D} = {}^A X_B {}^B v_{C,D}. \quad (1.1.34)$$

Mixed trivialized velocity

The left and right trivialized velocities introduced in Eq. (1.1.20) and Eq. (1.1.24), respectively, are constructed by serializing the linear component followed by the angular one. While the angular component corresponds in both representations to the angular velocity $\omega_{A,B}$, expressed in frame A for the left trivialized velocity and in frame B for the right trivialized one, in neither case the linear component corresponds to the time derivative of the position ${}^A o_B$ of frame B relative to frame A – see Eq. (1.1.19) for the left trivialized velocity and Eq. (1.1.23) for the right one.

On the other side, it is also common in robotics [Siciliano et al., 2008] to represent the 6D velocity of a rigid body just as

$$\begin{bmatrix} {}^A \dot{o}_B \\ {}^A \omega_{A,B} \end{bmatrix}, \quad (1.1.35)$$

a representation referred to as *hybrid* or *mixed* in [Traversaro et al., 2017].

This velocity can be easily related to the left and right trivialized velocities by introducing the so-called mixed reference frame $B[A] = (o_B, [A])$, that is, a frame having the same origin o_B of frame B and the same orientation $[A]$ of frame A – see Figure 1.1. Since the relative pose between frames B and $B[A]$ is

$${}^{B[A]}H_B = \begin{bmatrix} {}^A R_B & 0_{3 \times 1} \\ 0_{1 \times 3} & 1 \end{bmatrix}, \quad (1.1.36)$$

the velocity of frame B relative to A expressed in $B[A]$ is

$${}^{B[A]}v_{A,B} = {}^{B[A]}X_B {}^B v_{A,B} = \begin{bmatrix} {}^A R_B & 0_{3 \times 3} \\ 0_{3 \times 3} & {}^A R_B \end{bmatrix} \begin{bmatrix} {}^A R_B^\top {}^A \dot{o}_B \\ {}^B \omega_{A,B} \end{bmatrix} = \begin{bmatrix} {}^A \dot{o}_B \\ {}^A \omega_{A,B} \end{bmatrix}. \quad (1.1.37)$$

Throughout this thesis, we will call ${}^{B[A]}v_{A,B}$ in Eq. (1.1.37) the *mixed* velocity of frame B with respect to A , where "mixed" is because its linear and angular components relate to the left and right trivialized velocities, respectively.

1.1.3 Force applied to a rigid body

The interaction between a rigid body and the environment can be conveniently described by 6D force vectors (aka wrenches) $f \in \mathbb{R}^6$ containing a linear force $f \in \mathbb{R}^3$, the resultant of all the forces acting on the body, and a torque $\mu \in \mathbb{R}^3$, resulting from the moments of these forces with respect to a given point. Given a frame $B = (o_B, [B])$, the coordinates of a 6D force with respect to B are indicated by

$${}_B f := \begin{bmatrix} {}_B f \\ {}_B \mu \end{bmatrix} \in \mathbb{R}^6. \quad (1.1.38)$$

In this notation, B is the frame in which the 6D force vector ${}_B f$ is represented, meaning that the torque ${}_B \mu$ is taken with respect to o_B and both ${}_B f$ and ${}_B \mu$ are expressed in $[B]$. No indication is given on the point of application of the force, that is not assumed to be o_B .

As in the case of 6D velocities, also for 6D forces a linear transformation can be defined to change the frame in which the force coordinates are expressed. In particular, given a 6D force ${}_B f$ expressed in B , it can be expressed in A via

$${}_A f = {}_A X^B {}_B f, \quad (1.1.39)$$

where the force coordinate transformation ${}_A X^B$, induced by the velocity transformation in Eq. (1.1.33), is defined as

$${}_A X^B = {}_A X_B^\top = \begin{bmatrix} {}^A R_B & 0_{3 \times 3} \\ {}_A o_B \wedge {}^A R_B & {}^A R_B \end{bmatrix} \in \mathbb{R}^{6 \times 6}. \quad (1.1.40)$$

For instance, given the mixed reference frame $B[A] = (o_B, [A])$, related to the frame B as in Eq. (1.1.36), the force coordinate transformation ${}_{B[A]}X^B$ can be used to obtain the so-called *mixed* force representation [Traversaro and Saccon, 2019] as

$${}_{B[A]} f = {}_{B[A]}X^B {}_B f = \begin{bmatrix} {}^A R_B & 0_{3 \times 3} \\ 0_{3 \times 3} & {}^A R_B \end{bmatrix} \begin{bmatrix} {}_B f \\ {}_B \mu \end{bmatrix} = \begin{bmatrix} {}_A f \\ {}_{B[A]} \mu \end{bmatrix}. \quad (1.1.41)$$

1.2 Modeling of multi-body system

Complex mechanical structures such as humanoid robots are commonly described as systems of multiple interconnected rigid bodies. Let us consider a robot as a multi-body mechanical system composed of n_l rigid bodies, referred to as *links*, connected by n_j mechanisms, called *joints*. Figure 1.2 presents a multi-body system consisting of $n_l = 6$ links connected by $n_j = 5$ joints.

Each joint connects pairs of links together and limits their relative motion, acting as a kinematic constraint for the system. The constraint imposed by the joint depends upon the nature of the joint and its degrees of freedom, as well as on whether the corresponding connection generates a closed kinematic chain. In this thesis we only consider:

- *revolute* joints with a single degree of freedom, i.e., joints providing for pure rotation along a single axis;
- *open* kinematic chains, i.e., sequences of interconnected links with no loops.

These assumptions simplify the design and modeling of humanoid robots while enabling a wide range of anthropomorphic motions for the robot. For more general formulations that do not take into account the aforementioned assumptions, the reader is referred to [Featherstone, 2014].

Topology

Let us denote by $\mathcal{L} = \{L_1, \dots, L_{n_l}\}$ and $\mathcal{J} = \{J_1, \dots, J_{n_j}\}$ the sets of all the links and joints of the multi-body system, respectively. Each link L_i can be connected to multiple joints. Each joint J_i connects its parent link L_{p_i} to its child link L_{c_i} , and therefore is modeled as a pair of connected links $J_i = \{L_{p_i}, L_{c_i}\} \in \mathcal{J}$.

The couple $(\mathcal{L}, \mathcal{J})$ represents an *undirected graph*. The links in \mathcal{L} are the nodes of the graph and the joints in \mathcal{J} its edges. The graph is assumed to be *connected*, meaning that there exists at least one path between each pair of links. For the open kinematic chains considered in this thesis, the graph is also *acyclic*, meaning that the path between any two links is unique.

The connected acyclic undirected graph representing the multi-body system is also referred to as *kinematic tree* and is such that the number of links relates to the number of joints by $n_l = n_j + 1$. Figure 1.3 shows the kinematic tree of the multi-body system in Figure 1.2.

We assume that no link in \mathcal{L} has an a priori fixed pose with respect to the inertial frame \mathcal{I} , i.e., the system is *free floating*. Any link in \mathcal{L} can be then selected to be the *base* B of the free floating system, that is, the root of the kinematic tree. For instance, a common choice for humanoid robots is to place the base at the robot waist [Wensing et al., 2015] or at one of the robot feet [Iwasaki et al., 2012]. The pose of the base B with respect to the inertial frame \mathcal{I} is defined as

$${}^{\mathcal{I}}H_B = \begin{bmatrix} {}^{\mathcal{I}}R_B & {}^{\mathcal{I}}p_B \\ 0_{1 \times 3} & 1 \end{bmatrix}. \quad (1.2.1)$$

This transformation varies over time, being the system free floating, and is therefore pictured as a dashed arrow in Figure 1.2.

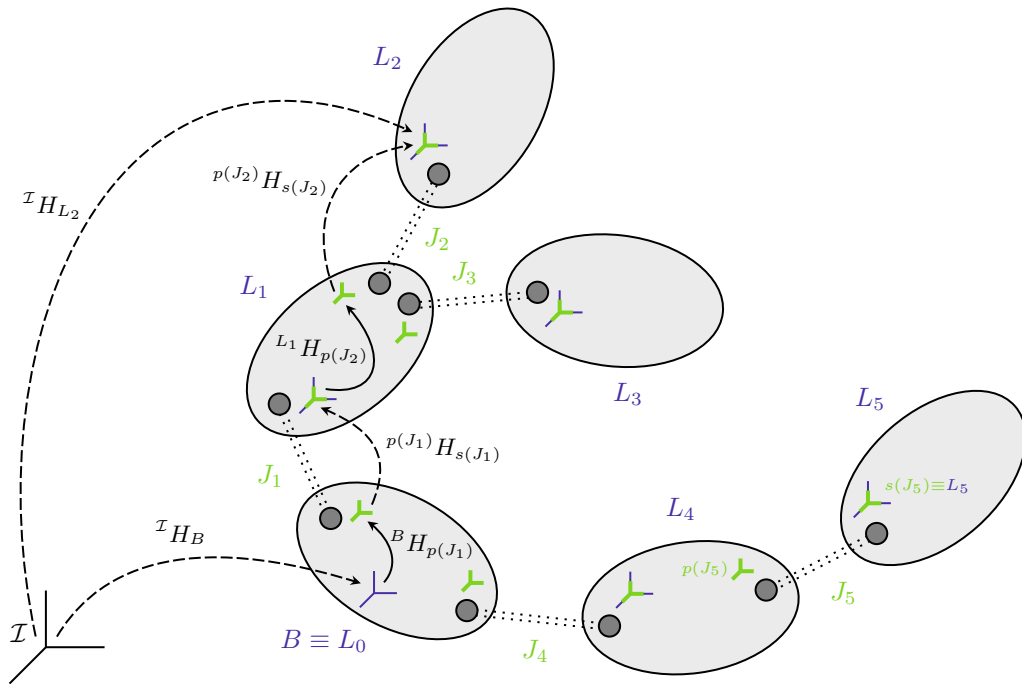


Figure 1.2. A multi-body system consisting of $n_l = 6$ links $\{L_0, \dots, L_5\}$ and $n_j = 5$ joints $\{J_1, \dots, J_5\}$. The link L_0 is chosen as floating base B . A blue frame L_i is attached to each link L_i . Each joint J_i is associated with a pair of green frames, the predecessor frame $p(J_i)$ and the successor frame $s(J_i)$, chosen to be coincident with the link frame L_i . Please look at J_5 for an explicit labeling of the correspondent frames. Solid and dashed arrows represent constant and time-varying transformations, respectively. The transforms involved in the computation of the forward kinematics of L_2 are illustrated.

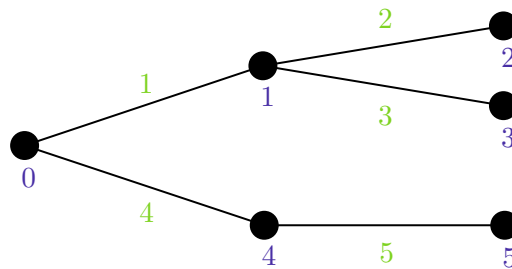


Figure 1.3. The kinematic tree representing the multi-body system in Figure 1.2. Links numbering (in blue) is such that each link has a higher number than its parent. Joints are numbered (in green) as their child link.

Given a choice for the base B , we further characterize the multi-body system and its kinematic tree as follows:

- For each link $L_i \in \mathcal{L}$, the *link numbering* function $\rho_l : \mathcal{L} \mapsto 0 \cup \mathbb{N}$ returns the number i associated to the link. The base B is associated to $\rho_l(B) = 0$. For the remaining links, any numbering such that each link has a higher number with respect to its parent works. Figures 1.2 and 1.3 illustrate, in blue, one possible link numbering for the considered structure.
- For each joint $J_i \in \mathcal{J}$, the *joint numbering* function $\rho_j : \mathcal{J} \mapsto \mathbb{N}$ returns the number i associated to the joint. Each joint is associated to the number of its child link. Figures 1.2 and 1.3 illustrate, in green, the joint numbering induced by the selected link numbering for the considered structure.
- Each link L_i has an associated homonym *link frame* L_i rigidly attached to it. Similarly, the free floating base B has the frame B attached to it. Link frames are depicted in blue in Figure 1.2.
- Each joint J_i has two associated frames, the *predecessor frame* $p(J_i)$ located on its parent link and the *successor frame* $s(J_i)$ located on its child link. In particular, the joint's successor frame is located over its child link frame. Joint frames are depicted in green in Figure 1.2. The complete frames' labeling is indicated for joint J_5 only, for the sake of clarity. Notice how, for each link but the base, the link frame and the successor frame of its parent joint overlap.
- For each link but the base, i.e., for each link $L_i \in \{\mathcal{L}/B\}$, the *parent link* function $\lambda_B : \{\mathcal{L}/B\} \mapsto \mathcal{L}$ maps each link to its parent. Referring to Figure 1.2, for instance, $\lambda_B(L_1) = B$ while $\lambda_B(L_5) = L_4$.
- Each link L_i is connected to the base B through a unique *path* $\pi_B(L_i)$, defined as the ordered sequence of links connecting B (excluded) to L_i (included) on the kinematic tree. Referring to Figures 1.2 and 1.3, for instance, $\pi_B(L_1) = [L_1]$, $\pi_B(L_2) = [L_1, L_2]$ and $\pi_B(L_3) = [L_1, L_3]$.

Parent-to-child link transform

For each joint J_i , the transformation ${}^{\lambda_B(L_i)}H_{L_i}$ from its child link L_i to its parent link $\lambda_B(L_i)$ can be retrieved by making use of the joint's predecessor and successor frames, i.e., $p(J_i)$ and $s(J_i)$, respectively. In particular, ${}^{\lambda_B(L_i)}H_{L_i}$ can be decomposed into the following series of three transformations

$${}^{\lambda_B(L_i)}H_{L_i}(s_i) = {}^{\lambda_B(L_i)}H_{p(J_i)} {}^{p(J_i)}H_{s(J_i)}(s_i) {}^{s(J_i)}H_{L_i}, \quad (1.2.2)$$

where ${}^{\lambda_B(L_i)}H_{p(J_i)}$ and ${}^{s(J_i)}H_{L_i}$ are constant transformations depending on the placement of $p(J_i)$ and $s(J_i)$, while ${}^{p(J_i)}H_{s(J_i)}(s_i)$ varies according to the joint configuration s_i . Since the joint's successor frame $s(J_i)$ is assumed to be located over its child link frame, ${}^{s(J_i)}H_{L_i} = I_4$ and Eq. (1.2.2) simplifies to

$${}^{\lambda_B(L_i)}H_{L_i}(s_i) = {}^{\lambda_B(L_i)}H_{p(J_i)} {}^{p(J_i)}H_{s(J_i)}(s_i), \quad (1.2.3)$$

where ${}^{\lambda_B(L_i)}H_{p(J_i)}$ and ${}^{p(J_i)}H_{s(J_i)}(s_i)$ are referred to as *tree transform* and *joint transform*, respectively. Figure 1.2 illustrates the decomposition of ${}^{\lambda_B(L_i)}H_{L_i}$ into a constant tree transform followed by a time-varying joint transform for joints J_1 and J_2 , omitting the dependency of the joint transform on the joint configuration s_i for the sake of clarity.

In the specific case of revolute joints considered in this thesis, the joint transform is obtained through the axis-angle formalism. A revolute joint J_i , characterized by a unitary-norm axis ${}_i a \in \{x \in \mathbb{R}^3 | x^\top x = 1\}$ and an angle $s_i \in \mathbb{R}$, provides for a rotation of s_i along ${}_i a$. Therefore, it induces a joint transform

$${}^{p(J_i)}H_{s(J_i)}(s_i) = \begin{bmatrix} {}^{p(J_i)}R_{s(J_i)}(s_i) & 0_{3 \times 1} \\ 0_{1 \times 3} & 1 \end{bmatrix}, \quad (1.2.4)$$

where, assuming ${}^{p(J_i)}H_{s(J_i)}(0) = I_4$, the rotation is given by the Rodrigues' formula

$${}^{p(J_i)}R_{s(J_i)}(s_i) = I_3 + \cos(s_i) {}_i a^\wedge + \sin(s_i) ({}_i a^\wedge)^2. \quad (1.2.5)$$

Parent-to-child link velocity

For each joint J_i , the relative velocity of its child link L_i with respect to its parent link $\lambda_B(L_i)$ is retrieved by differentiating Eq. (1.2.3) as

$$\frac{d}{dt} \left({}^{\lambda_B(L_i)}H_{L_i}(s_i) \right) = \frac{\partial}{\partial s_i} \left({}^{\lambda_B(L_i)}H_{L_i}(s_i) \right) \dot{s}_i. \quad (1.2.6)$$

Let us introduce a frame placeholder F standing for L_i , $\lambda_B(L_i)$ and $L_i[\lambda_B(L_i)]$ in the left-trivialized, right-trivialized and mixed velocity representations, respectively. Expressed as a 6D velocity, Eq. (1.2.6) writes then as

$${}^F v_{\lambda_B(L_i), L_i} = {}^F S_{\lambda_B(L_i), L_i}(s_i) \dot{s}_i \quad (1.2.7)$$

where the *joint motion subspace vector* ${}^F S_{\lambda_B(L_i), L_i}(s_i)$, in the different representations, is given by

$${}^{L_i} S_{\lambda_B(L_i), L_i}(s_i) = \left({}^{\lambda_B(L_i)}H_{L_i}^{-1}(s_i) \frac{\partial}{\partial s_i} \left({}^{\lambda_B(L_i)}H_{L_i}(s_i) \right) \right)^\vee, \quad (1.2.8)$$

$${}^{\lambda_B(L_i)} S_{\lambda_B(L_i), L_i}(s_i) = \left(\frac{\partial}{\partial s_i} \left({}^{\lambda_B(L_i)}H_{L_i}(s_i) \right) {}^{\lambda_B(L_i)}H_{L_i}^{-1}(s_i) \right)^\vee, \quad (1.2.9)$$

$${}^{L_i[\lambda_B(L_i)]} S_{\lambda_B(L_i), L_i}(s_i) = \left[\begin{array}{c} \frac{\partial}{\partial s_i} \left({}^{\lambda_B(L_i)}o_{L_i}(s_i) \right) \\ \left(\frac{\partial}{\partial s_i} \left({}^{\lambda_B(L_i)}R_{L_i}(s_i) \right) \left({}^{\lambda_B(L_i)}R_{L_i}(s_i) \right)^\top \right)^\vee \end{array} \right]. \quad (1.2.10)$$

In the specific case of revolute joints considered in this thesis, it can be shown that the left-trivialized, right-trivialized and mixed joint motion subspaces coincide, are constant and are given by [Traversaro, 2017, Lemma 3.6]

$${}^{L_i} S_{\lambda_B(L_i), L_i} = {}^{\lambda_B(L_i)} S_{\lambda_B(L_i), L_i} = {}^{L_i[\lambda_B(L_i)]} S_{\lambda_B(L_i), L_i} = \begin{bmatrix} 0_{3 \times 1} \\ {}_i a \end{bmatrix} = S_{\lambda_B(L_i), L_i}. \quad (1.2.11)$$

In this case, the 6D relative velocity of the child L_i of joint J_i with respect to its parent $\lambda_B(L_i)$, independently from the chosen representation, is therefore given by

$$v_{\lambda_B(L_i), L_i} = S_{\lambda_B(L_i), L_i} \dot{s}_i. \quad (1.2.12)$$

1.2.1 Multi-body kinematics

Given the definitions of the parent-to-child link transform ${}^{\lambda_B(L_i)}H_{L_i}$ in Eq. (1.2.3) and the parent-to-child link velocity $v_{\lambda_B(L_i),L_i}$ in Eq. (1.2.12), it is possible to examine the kinematics of the entire floating-base multi-body system.

Let us define first the floating-base system configuration q as

$$q := \left({}^{\mathcal{I}}H_B, s \right) \in \text{SE}(3) \times \mathbb{R}^n, \quad (1.2.13)$$

where n denotes the number of degrees of freedom of the system that matches the number of joints, i.e., $n = n_j$, under the assumption of revolute joints only. The system configuration is fully determined by the base pose ${}^{\mathcal{I}}H_B \in \text{SE}(3)$ and the *joint positions* $s \in \mathbb{R}^n$, also called *shape*.

Although the derivative of the system configuration is

$$\dot{q} := \left({}^{\mathcal{I}}\dot{H}_B, \dot{s} \right), \quad (1.2.14)$$

where $\dot{s} \in \mathbb{R}^n$ are the *joint velocities*, it is often more convenient – as in the case of a single rigid body – to represent the system velocity as a column vector. Selecting the left-trivialized velocity representation for the base, we introduce the left-trivialized floating-base system velocity as

$${}^B\nu := \begin{bmatrix} {}^B v_{\mathcal{I},B} \\ \dot{s} \end{bmatrix} = \begin{bmatrix} {}^B v_{\mathcal{I},B} \\ {}^B \omega_{\mathcal{I},B} \\ \dot{s} \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{I}}R_B^\top {}^{\mathcal{I}}\dot{o}_B \\ \left({}^{\mathcal{I}}R_B^\top {}^{\mathcal{I}}\dot{R}_B \right)^\vee \\ \dot{s} \end{bmatrix} \in \mathbb{R}^{6+n}. \quad (1.2.15)$$

Adopting instead the mixed velocity representation for the base, we introduce the mixed system velocity ${}^{B[\mathcal{I}]}\nu$ as

$${}^{B[\mathcal{I}]}\nu := \begin{bmatrix} {}^{B[\mathcal{I}]}v_{\mathcal{I},B} \\ \dot{s} \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{I}}\dot{o}_B \\ {}^{\mathcal{I}}\omega_{\mathcal{I},B} \\ \dot{s} \end{bmatrix} \in \mathbb{R}^{6+n}, \quad (1.2.16)$$

and the alternative definition of the free-floating system configuration q in the form

$$q := \left({}^{\mathcal{I}}o_B, {}^{\mathcal{I}}R_B, s \right) \in \mathbb{R}^3 \times \text{SO}(3) \times \mathbb{R}^n. \quad (1.2.17)$$

Link transform

The pose of a link L_i with respect to the inertial frame \mathcal{I} is a function of the floating-base system configuration q . Given the unique path $\pi_B(L_i)$ connecting L_i to the base B , the pose of L_i is defined as

$${}^{\mathcal{I}}H_{L_i}(q) = {}^{\mathcal{I}}H_B {}^B H_{L_i}(s) = {}^{\mathcal{I}}H_B \prod_{L_k \in \pi_B(L_i)} {}^{\lambda_B(L_k)}H_{L_k}(s_k). \quad (1.2.18)$$

In other words, the pose of L_i can be reconstructed by concatenating the base transform ${}^{\mathcal{I}}H_B$ with the ordered sequence of parent-to-child link transforms ${}^{\lambda_B(L_k)}H_{L_k}(s_k)$ of all the links $L_k \in \pi_B(L_i)$, defined as in Eq. (1.2.3).

We refer to Eq. (1.2.18) as the *forward kinematics* function Π . Given the floating-base system configuration q , Π returns the absolute pose of a link, i.e.,

$$\Pi : \text{SE}(3) \times \mathbb{R}^n \mapsto \text{SE}(3), \quad \Pi(q) = {}^{\mathcal{I}}H_{L_i}(q). \quad (1.2.19)$$

As an example, Figure 1.2 shows the forward kinematics for the pose of L_2 , computed as ${}^{\mathcal{I}}H_{L_2}(q) = {}^{\mathcal{I}}H_B {}^B H_{p(J_1)} {}^{p(J_1)} H_{s(J_1)}(s_1) {}^{L_1} H_{p(J_2)} {}^{p(J_2)} H_{s(J_2)}(s_2)$.

Link velocity

Assuming one-degree-of-freedom revolute joints, the velocity of link L_i with respect to the inertial frame is a function of the floating-base system velocity ν . In the following, we will express 6D velocities using the left-trivialized representation, i.e., we will relate the link velocity ${}^{L_i}v_{\mathcal{I},L_i}$ to the system velocity ${}^B\nu$. However, similar results hold if the velocities are expressed in different representations. Let us decompose the velocity of link L_i as

$${}^{L_i}v_{\mathcal{I},L_i} = {}^{L_i}v_{\mathcal{I},B} + {}^{L_i}v_{B,L_i} \quad (1.2.20a)$$

$$= {}^{L_i}X_B {}^B v_{\mathcal{I},B} + {}^{L_i}v_{B,L_i}, \quad (1.2.20b)$$

where the velocity transformation in Eq. (1.1.34) is used to express the left-trivialized base velocity ${}^B v_{\mathcal{I},B}$ in the frame L_i . By further developing the left-trivialized velocity ${}^{L_i}v_{B,L_i}$ of L_i with respect to B as sum of the parent-to-child link velocities of all the links $L_k \in \pi_B(L_i)$, we obtain

$${}^{L_i}v_{\mathcal{I},L_i} = {}^{L_i}X_B {}^B v_{\mathcal{I},B} + \sum_{L_k \in \pi_B(L_i)} {}^{L_i}v_{\lambda_B(L_k),L_k} \quad (1.2.21a)$$

$$= {}^{L_i}X_B {}^B v_{\mathcal{I},B} + \sum_{L_k \in \pi_B(L_i)} {}^{L_i}X_{L_k} {}^{L_k}v_{\lambda_B(L_k),L_k} \quad (1.2.21b)$$

$$= {}^{L_i}X_B {}^B v_{\mathcal{I},B} + \sum_{L_k \in \pi_B(L_i)} {}^{L_i}X_{L_k} S_{\lambda_B(L_k),L_k} \dot{s}_k, \quad (1.2.21c)$$

where the parent-to-child link velocity ${}^{L_k}v_{\lambda_B(L_k),L_k}$ is expressed in terms of the joint motion subspace vector (constant for revolute joints) as introduced in Eq. (1.2.12). In matrix form, ${}^{L_i}v_{\mathcal{I},L_i}$ can be written as

$${}^{L_i}v_{\mathcal{I},L_i} = \begin{bmatrix} {}^{L_i}X_B & {}^{L_i}J_{B,L_i}^{\dot{s}} \end{bmatrix} \begin{bmatrix} {}^B v_{\mathcal{I},B} \\ \dot{s} \end{bmatrix} = {}^{L_i}J_{\mathcal{I},L_i}^B {}^B \nu, \quad (1.2.22)$$

where ${}^{L_i}J_{\mathcal{I},L_i}^B \in \mathbb{R}^{6 \times (6+n)}$ is the *left-trivialized Jacobian* of link L_i and ${}^{L_i}J_{B,L_i}^{\dot{s}} \in \mathbb{R}^{6 \times n}$ is such that its k -th column is

$$({}^{L_i}J_{B,L_i}^{\dot{s}})_{(:,k)} = \begin{cases} {}^{L_i}X_{L_k} S_{\lambda_B(L_k),L_k} & \text{if } L_k \in \pi_B(L_i), \\ 0_6 & \text{otherwise.} \end{cases} \quad (1.2.23)$$

By applying Eq. (1.1.37) to Eq. (1.2.22), the velocity of link L_i can be expressed in mixed representation. In particular, you can relate the mixed link velocity ${}^{L_i}v_{\mathcal{I},L_i}^{[T]}$

to the mixed system velocity ${}^{B[\mathcal{I}]}v$ as

$${}^{L_i[\mathcal{I}]}v_{\mathcal{I},L_i} = {}^{L_i[\mathcal{I}]}X_{L_i} \begin{bmatrix} {}^{L_i}X_B & {}^{L_i}J_{B,L_i}^s \end{bmatrix} \begin{bmatrix} {}^B X_{B[\mathcal{I}]} & 0_{6 \times n} \\ 0_{n \times 6} & I_n \end{bmatrix} \begin{bmatrix} {}^{B[\mathcal{I}]}v_{\mathcal{I},B} \\ \dot{s} \end{bmatrix} \quad (1.2.24a)$$

$$= {}^{L_i[\mathcal{I}]}J_{\mathcal{I},L_i}^{B[\mathcal{I}]} {}^{B[\mathcal{I}]}v, \quad (1.2.24b)$$

where ${}^{L_i[\mathcal{I}]}J_{\mathcal{I},L_i}^{B[\mathcal{I}]}$ is the *mixed Jacobian* of link L_i .

1.2.2 Multi-body dynamics

Let us assume a mixed velocity representation and simplify the notation by referring to the mixed system velocity ${}^{B[\mathcal{I}]}v$ simply as v . Let us also assume that the interaction between the floating-base system and the environment takes place by exchanging n_c distinct 6D forces, with the k -th external 6D force f_k applied to the contact frame \mathcal{C}_k . In the simplified notation assuming mixed representations, f_k actually stands for the mixed force ${}^{C_k[\mathcal{I}]}f_k$ measured in the frame having the same origin of \mathcal{C}_k and oriented as \mathcal{I} - see Eq. (1.1.41).

Since the configuration space is not a vector space, rather than deriving the free-floating system dynamics through the classical Euler-Lagrange equations [Wieber et al., 2016] we employ the Euler-Poincaré formalism [Marsden and Ratiu, 2010, Section 13.5]. Such a derivation, which is omitted here but can be found in [Traversaro, 2017, Section 3.5, Theorem 3.2], leads to the following *equations of motion*

$$M(q)\dot{v} + C(q, v)v + G(q) = \begin{bmatrix} 0_{6 \times n} \\ I_n \end{bmatrix} \tau_s + \sum_{k=1}^{n_c} J_{C_k}^\top f_k, \quad (1.2.25)$$

where $M \in \mathbb{R}^{(n+6) \times (n+6)}$ is the mass matrix, $C \in \mathbb{R}^{(n+6) \times (n+6)}$ accounts for Coriolis and centrifugal effects, and $G \in \mathbb{R}^{n+6}$ contains the gravity term. The vector $\tau_s \in \mathbb{R}^n$ contains the internal actuation torques. The Jacobian J_{C_k} , standing for ${}^{C_k[\mathcal{I}]}J_{\mathcal{I},C_k}^{B[\mathcal{I}]}$ in the simplified notation assuming mixed representations, is the mixed Jacobian of the contact frame \mathcal{C}_k - see Eq. (1.2.24).

By stacking the Jacobians and contact forces, Eq. (1.2.25) can be rearranged as

$$M(q)\dot{v} + C(q, v)v + G(q) = \begin{bmatrix} 0_{6 \times n} \\ I_n \end{bmatrix} \tau_s + J_C^\top f, \quad (1.2.26)$$

where

$$J_C(q) = \begin{bmatrix} J_{C_1}(q) \\ \vdots \\ J_{C_{n_c}}(q) \end{bmatrix}, \quad f = \begin{bmatrix} f_1 \\ \vdots \\ f_{n_c} \end{bmatrix}. \quad (1.2.27)$$

Moreover, a set of *holonomic constraints* is assumed to act on the system in combination with the n_c rigid contacts. In particular, the k -th link in contact with the environment (e.g., the support foot of a humanoid robot interacting with the ground) is required to remain fixed as long as the contact is active. In *Pfaffian form*, this translates into a set of holonomic constraints prescribing 6D velocities equal to zero for the links in contact, i.e.,

$$J_{C_k}(q)v = 0. \quad (1.2.28)$$

Notice that Eq. (1.2.28) can be differentiated as

$$\dot{J}_{C_k} \nu + J_{C_k} \dot{\nu} = 0, \quad (1.2.29)$$

obtaining a dependency of the constraint on $\dot{\nu}$. Taken together, Eq. (1.2.26) and Eq. (1.2.29) provide the dynamical equations describing the motion of a floating-base system that instantiates rigid contacts with the environment.

1.2.3 Centroidal dynamics

Let us denote the mass of each link L_i of the multi-body system with $m_i \in \mathbb{R}$. The link center of mass and its 3D inertia matrix, both expressed in L_i , are denoted by ${}^{L_i}c_i \in \mathbb{R}^3$ and ${}^{L_i}\mathbb{I}_i \in \mathbb{R}^{3 \times 3}$, respectively. Then, the 6D inertia matrix of link L_i , expressed in L_i , is defined as [Traversaro, 2017, Eq. (2.67)]

$${}^{L_i}\mathbb{M}_{L_i} := \begin{bmatrix} m_i I_3 & -(m_i {}^{L_i}c_i)^\wedge \\ (m_i {}^{L_i}c_i)^\wedge & {}^{L_i}\mathbb{I}_i \end{bmatrix}, \quad (1.2.30)$$

and is a constant matrix. Moreover, the 6D momentum of link L_i with respect to \mathcal{I} , expressed in the base frame B , is defined as

$${}^B h_{\mathcal{I}, L_i} := {}^B X^{L_i} {}^{L_i}\mathbb{M}_{L_i} {}^{L_i} v_{\mathcal{I}, L_i}, \quad (1.2.31)$$

where ${}^B X^{L_i}$ is the force coordinate transformation from Eq. (1.1.40).

If we denote with m the total mass of the multi-body system, i.e.,

$$m := \sum_i m_i, \quad (1.2.32)$$

then the system center of mass x_{CoM} , expressed in the inertial frame \mathcal{I} , is defined as the weighted average of the centers of mass of all the links

$$x_{\text{CoM}} := {}^{\mathcal{I}} H_B \sum_i \frac{m_i}{m} {}^B H_{L_i} {}^{L_i} c_i. \quad (1.2.33)$$

Let us introduce the frame \bar{G} , having its origin located on x_{CoM} and oriented as the inertial frame \mathcal{I} . The *centroidal momentum* $\bar{G}h \in \mathbb{R}^6$ is then given by the sum of all the link 6D momenta, projected on \bar{G}

$$\bar{G}h = \begin{bmatrix} \bar{G}h^p \\ \bar{G}h^\omega \end{bmatrix} = \bar{G} X^B \sum_i {}^B h_{\mathcal{I}, L_i} \quad (1.2.34a)$$

$$= \bar{G} X^B \sum_i {}^B X^{L_i} {}^{L_i}\mathbb{M}_{L_i} {}^{L_i} v_{\mathcal{I}, L_i} \quad (1.2.34b)$$

$$= J_{CMM} \nu, \quad (1.2.34c)$$

where the passage from Eq. (1.2.34b) to Eq. (1.2.34c) makes the dependency of $\bar{G}h$ on the robot velocity ν explicit by the use of Eq. (1.2.22) and $J_{CMM} \in \mathbb{R}^{(6 \times n)}$ is the *Centroidal Momentum Matrix* (CMM) [Orin and Goswami, 2008], while $\bar{G}h^p \in \mathbb{R}^3$ and $\bar{G}h^\omega \in \mathbb{R}^3$ are the linear and angular momentum, respectively. In particular, the linear component of the centroidal momentum $\bar{G}h^p$ is related to the center of mass velocity \dot{x}_{CoM} via

$$\bar{G}h^p = m \dot{x}_{\text{CoM}}. \quad (1.2.35)$$

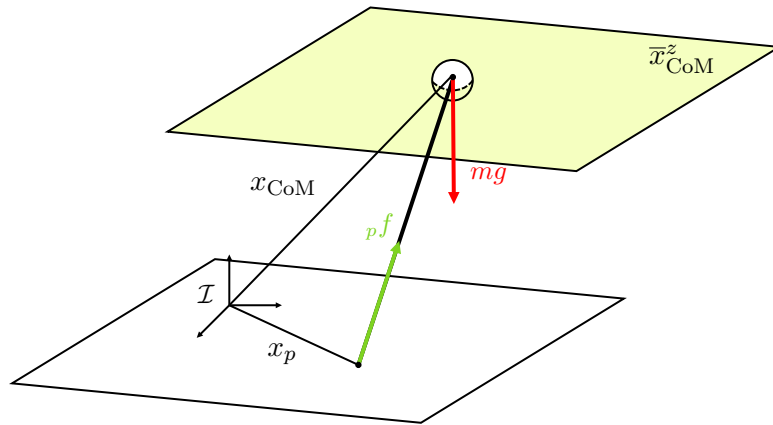


Figure 1.4. Representation of the Linear Inverted Pendulum (LIP) model. The external force ${}_p f$ and the gravity mg acting on the system are highlighted in green and red, respectively. The CoM position x_{CoM} is assumed to belong to the yellow plane at constant height \bar{x}_{CoM}^z .

The rate of change of the centroidal momentum balances the external forces acting on the multi-body system [Orin et al., 2013; Wensing and Orin, 2016]. This consideration leads to the formulation of the centroidal momentum dynamics, or simply *centroidal dynamics*, which takes the form

$$\bar{G}\dot{h} = \sum_{k=1}^{n_c} \bar{G}X^{C_k} c_k f_k + m\bar{g} \quad (1.2.36a)$$

$$= \sum_{k=1}^{n_c} \begin{bmatrix} \mathcal{I}R_{C_k} & 0_{3 \times 3} \\ (\mathcal{I}o_{C_k} - x_{\text{CoM}})^\wedge \mathcal{I}R_{C_k} & \mathcal{I}R_{C_k} \end{bmatrix} c_k f_k + m\bar{g}, \quad (1.2.36b)$$

where the adjoint matrix $\bar{G}X^{C_k}$ transforms the external force $c_k f_k$ from the application frame C_k to \bar{G} and $\bar{g} = [0, 0, -g, 0, 0, 0]^\top$ denotes the 6D gravity acceleration vector.

1.3 Simplified models

To lower the computational burden arising from a complex dynamics modeling that relies on the accurate knowledge of many parameters, a series of simplified models have been proposed in the legged locomotion literature. This section briefly introduces some of the most widely employed ones, along with related concepts of particular relevance for bipedal locomotion.

1.3.1 Linear inverted pendulum

Consider a biped supporting its body on one leg. The Linear Inverted Pendulum (LIP) model [Kajita et al., 2001, 2003] allows to approximate its dynamics by considering an inverted pendulum with negligible inertia, having a point mass on its top edge and making a rigid contact with the ground at its bottom edge – see Figure 1.4. The point mass represents the robot center of mass, connected through a massless leg to a point foot in contact with the ground.

Let $x_{\text{CoM}} \in \mathbb{R}^3$ and $x_p \in \mathbb{R}^3$ denote the position of the robot CoM and of the contact point with respect to the inertial frame \mathcal{I} , respectively. Assume that:

1. A single external force ${}_p f \in \mathbb{R}^3$ acts on the system, applied in x_p and parallel to the pendulum.
2. The gravity vector g acts on the mass point m along the negative z axis of \mathcal{I} .
3. The CoM height remains constant, i.e., $e_3^\top(x_p - x_{\text{CoM}}) = \bar{x}_{\text{CoM}}^z$, which also implies that $e_3^\top \dot{x}_{\text{CoM}} = 0$ and $e_3^\top \ddot{x}_{\text{CoM}} = 0$ hold.
4. The centroidal angular momentum is constantly zero, i.e., $\bar{G}h^\omega = 0$, implying that also $\bar{G}\dot{h}^\omega = 0$ holds.

Then, the centroidal dynamics in Eq. (1.2.36) applied to the LIP model becomes

$$m\ddot{x}_{\text{CoM}} = {}_p f + mg, \quad (1.3.1a)$$

$$(x_p - x_{\text{CoM}})^\wedge {}_p f = 0, \quad (1.3.1b)$$

where Eq. (1.3.1b) encodes the constant angular momentum constraint and, along with hypothesis 3, allows to write the components of ${}_p f$ as

$${}_p f_x = m\omega^2 e_1^\top (x_{\text{CoM}} - x_p), \quad (1.3.2a)$$

$${}_p f_y = m\omega^2 e_2^\top (x_{\text{CoM}} - x_p), \quad (1.3.2b)$$

$${}_p f_z = -m|g|, \quad (1.3.2c)$$

where $\omega = \sqrt{\frac{|g|}{\bar{x}_{\text{CoM}}^z}}$ is the time constant of the LIP model. By replacing Eq. (1.3.2) into Eq. (1.3.1), we obtain

$$\ddot{x}_{\text{CoM}} = \begin{bmatrix} e_1^\top \\ e_2^\top \\ 0_{1 \times 3} \end{bmatrix} \omega^2 (x_{\text{CoM}} - x_p). \quad (1.3.3)$$

Being this dynamics different from zero in the planar coordinates only, it is convenient to rearrange it by considering $x_p \in \mathbb{R}^2$ and defining $x_{\text{LIP}} \in \mathbb{R}^2$ as

$$x_{\text{LIP}} = \begin{bmatrix} x_{\text{LIP}}^x \\ x_{\text{LIP}}^y \end{bmatrix} = \begin{bmatrix} e_1^\top \\ e_2^\top \end{bmatrix} x_{\text{CoM}}. \quad (1.3.4)$$

We then obtain the LIP dynamics equation

$$\ddot{x}_{\text{LIP}} = \omega^2 (x_{\text{LIP}} - x_p), \quad (1.3.5)$$

a second-order linear dynamics that is completely decoupled in the x and y coordinates. Focusing on the x_{LIP}^x component – with the same considerations holding for x_{LIP}^y – we can define $x_1 = x_{\text{LIP}}^x$ and $x_2 = \dot{x}_{\text{LIP}}^x$, and express Eq. (1.3.5) in the state-space representation as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \omega^2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \omega^2 \end{bmatrix} x_p^x = A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + Bx_p^x. \quad (1.3.6)$$

Being the eigenvalues of A equal to $\lambda_{1,2} = \pm\omega$, the single equilibrium of the system

$$\begin{bmatrix} x_{\text{LIP}_e}^x \\ x_{\text{LIP}_e}^y \\ 0 \end{bmatrix} = \begin{bmatrix} x_p^x \\ 0 \end{bmatrix} \quad (1.3.7)$$

is unstable. The LIP model has also been extended to the case of a finite-sized foot [Kajita et al., 2001; Koolen et al., 2012]. In this case, x_p corresponds to the position x_{ZMP} of the Zero Moment Point (ZMP) [Vukobratović and Borovac, 2004] introduced in the next section.

1.3.2 Zero moment point

Consider a rigid body, with a frame $B = (o_B, [B])$ attached, making contact with the external environment through a contact domain $\Omega \in \mathbb{R}^3$. The contact domain Ω is associated to:

- A continuous force distribution $\rho : \Omega \mapsto \mathbb{R}^3$, that associates to each point ${}^B x \in \Omega$ the force $\rho({}^B x)$;
- A continuous torque distribution $\sigma_{o_B} : \Omega \mapsto \mathbb{R}^3$, that associates to each point ${}^B x \in \Omega$ the torque $\sigma_{o_B}({}^B x)$ computed with respect to the origin o_B of B as

$$\sigma_{o_B}({}^B x) = {}^B x \times \rho({}^B x) \quad (1.3.8)$$

The equivalent left-trivialized contact 6D force ${}^B f$ acting on the body is then obtained by integrating the force and the torque contributions over the contact domain [Caron et al., 2015]

$${}^B f = \begin{bmatrix} {}^B f \\ {}^B \mu \end{bmatrix} = \begin{bmatrix} \int_{\Omega} \rho \, d\Omega \\ \int_{\Omega} \sigma_{o_B} \, d\Omega \end{bmatrix}. \quad (1.3.9)$$

Consider the mixed frame $F[B]$ having its origin in o_F and oriented as B . The Zero Moment Point (ZMP) x_{ZMP} is defined as the point $o_F \in \Omega$ (if it exists) such that

$$e_1^\top {}_{F[B]}\mu = e_2^\top {}_{F[B]}\mu = 0, \quad (1.3.10)$$

namely, the point $o_F \in \Omega$ such that the horizontal component of ${}_{F[B]}\mu$ is equal to zero. If it exists, the *local* ZMP, i.e., the ZMP expressed in the body frame B , is given by [Vukobratović and Borovac, 2004]

$${}^B x_{\text{ZMP}} = \begin{bmatrix} -\frac{{}^B \mu^y}{{}^B f^z} \\ \frac{{}^B \mu^x}{{}^B f^z} \end{bmatrix}, \quad (1.3.11)$$

where the superscripts extract specific vector coordinates.

The ZMP is used to describe whether the body can tilt. If $o_F \in \Omega$, the contact domain Ω does not change, and therefore the contact dynamic balance is ensured. If $o_F \notin \Omega$, the rigid body is instead free to rotate around one of its edges. This non-tilting condition is known as ZMP condition [Arakawa and Fukuda, 1997] or ZMP stability criterion [Li et al., 1993] and has been widely used along with the

LIP model as a bipedal stability criterion [Kajita et al., 2003; Hirai et al., 1998]. In particular, Eq. (1.3.5) combined with the ZMP definition leads to the CoM dynamics

$$\ddot{x}_{\text{LIP}} = \omega^2(x_{\text{LIP}} - {}^{\mathcal{I}}H_B^B x_{\text{ZMP}}). \quad (1.3.12)$$

In the presence of a floating-base system, more links can be in contact with the environment. If the contact domains of the n_c links L in contact belong to the same plane, we define the *global ZMP* as

$${}^{\mathcal{I}}x_{\text{ZMP}_g} = \sum_{k=1}^{n_c} {}^{\mathcal{I}}H_{L_k} {}^{L_k}x_{\text{ZMP}_k} \frac{{}^{L_k}f_k^z}{f^z} \quad (1.3.13)$$

where f^z is the sum of all the contact forces acting along the z axis

$$f^z = \sum_{k=1}^{n_c} {}^{L_k}f_k^z \quad (1.3.14)$$

Let us recall that the global ZMP is defined in the convex hull¹ of all the points in the contact domains, also referred to as *support polygon*. In this case, the ZMP stability criterion requires the ZMP to remain strictly inside the support polygon.

1.3.3 Divergent component of motion

Consider a multi-body system of total mass m making n_c contacts with the environment. Let $\bar{G} := (x_{\text{CoM}}, [Z])$ denote a frame having its origin in the CoM of the multi-body system and oriented as the inertial frame \mathcal{I} . The CoM dynamics is

$$\ddot{x}_{\text{CoM}} = \frac{1}{m}(mg + \bar{G}f), \quad (1.3.15)$$

where g is the gravity vector and $\bar{G}f$ represents the sum of all the external forces acting on the system, expressed in \bar{G} .

The three-dimensional Divergent Component of Motion (DCM) [Englsberger et al., 2011, 2013, 2015], denoted by ξ , is defined as

$$\xi = x_{\text{CoM}} + b\dot{x}_{\text{CoM}}, \quad (1.3.16)$$

where $b > 0$ is the time-constant of the DCM dynamics. Therefore, the DCM is defined as a point that is located at a certain distance from of the CoM, along the CoM velocity direction. Eq. (1.3.16) can be rearranged to derive the CoM dynamics

$$\dot{x}_{\text{CoM}} = -\frac{1}{b}(x_{\text{CoM}} - \xi), \quad (1.3.17)$$

showing that the CoM has a stable first-order dynamics, i.e., it follows the DCM.

By differentiating Eq. (1.3.16) and inserting Eq. (1.3.15) and Eq. (1.3.17), the DCM dynamics can be written as

$$\dot{\xi} = \dot{x}_{\text{CoM}} + b\ddot{x}_{\text{CoM}} \quad (1.3.18a)$$

$$= -\frac{1}{b}(x_{\text{CoM}} - \xi) + \frac{b}{m}(mg + \bar{G}f). \quad (1.3.18b)$$

¹The convex hull of a set of points is the smallest convex region that contains all the points.

Since $\bar{G}f$ directly influences the DCM dynamics, one can design external forces that are appropriate for a stable locomotion of the multi-body system. To simplify the design, the external forces can be defined through a force-to-point transform

$$\bar{G}f = \alpha(x_{\text{CoM}} - x_{\text{eCMP}}), \quad (1.3.19)$$

where $\alpha > 0$ is a constant and x_{eCMP} is the so-called Enhanced Centroidal Moment Pivot (eCMP) point [Englsberger et al., 2015]. Replacing $\bar{G}f$ from Eq. (1.3.19) into Eq. (1.3.18), the DCM dynamics writes as

$$\dot{\xi} = \left(\frac{b\alpha}{m} - \frac{1}{b} \right) x_{\text{CoM}} + \frac{1}{b} \xi - \frac{b\alpha}{m} x_{\text{eCMP}} + bg, \quad (1.3.20)$$

showing that x_{CoM} and ξ are in general coupled. However, by choosing $\alpha = \frac{m}{b^2}$ the DCM dynamics becomes decoupled from the CoM dynamics, in the form

$$\dot{\xi} = \frac{1}{b} \xi - \frac{1}{b} x_{\text{eCMP}} + bg. \quad (1.3.21)$$

Let us further simplify the DCM dynamics by defining the Virtual Repellent Point (VRP) [Englsberger et al., 2015] as

$$x_{\text{VRP}} = x_{\text{eCMP}} + b^2g, \quad (1.3.22)$$

and noticing that the VRP differs from the eCMP only in the z component, by a vertical displacement $b^2|g|$. Using the VRP, the DCM dynamics simplifies as

$$\dot{\xi} = \frac{1}{b} (\xi - x_{\text{VRP}}), \quad (1.3.23)$$

showing that the DCM has an unstable first-order dynamics, i.e., it is "pushed" away by the VRP on a straight line (while the CoM is following it).

The 3D-DCM dynamics equations hold for general free-floating robot models. However, let us consider the case in which the multi-body system is approximated by the LIP model and b is chosen as the inverse of the LIP model time constant ω , i.e., $b = 1/\omega$. If $\xi_{\text{LIP}} \in \mathbb{R}^2$ are the planar coordinates of the DCM, i.e.,

$$\xi_{\text{LIP}} = \begin{bmatrix} e_1^\top \\ e_2^\top \end{bmatrix} \xi, \quad (1.3.24)$$

it is possible to prove that the ground projection of the VRP, by definition equal to the ground projection of the eCMP, coincides with the ZMP [Romualdi, 2022, Section 4.4.1]. Therefore, the planar DCM and CoM dynamics, under the LIP model assumptions, take the form

$$\dot{\xi}_{\text{LIP}} = \omega (\xi_{\text{LIP}} - x_{\text{ZMP}}), \quad (1.3.25)$$

$$\dot{x}_{\text{LIP}} = -\omega (x_{\text{LIP}} - \xi_{\text{LIP}}). \quad (1.3.26)$$

Under the LIP assumptions, the DCM dynamics corresponds therefore to the unstable part of the LIP dynamics highlighted in Eq. (1.3.6). In this case, the DCM is equivalent to the Capture Point [Pratt et al., 2006, 2012; Koolen et al., 2012] and has also been referred to as Extrapolated Center of Mass (xCoM) [Hof, 2008].

Chapter 2

Robot Learning

In the previous chapter, we described how to derive kinematic and dynamical models of floating-base robots. This chapter introduces the basics of *reinforcement learning* and *supervised deep learning*, two subfields of machine learning that we exploit to design the controller in Chapter 4 and the planner in Chapter 5, respectively. For more in-depth coverage of these topics, the reader is referred to the works on which most of the material presented in this chapter is based, i.e., [Bishop, 2006; Goodfellow et al., 2016; Rumelhart et al., 1986; Sutton and Barto, 2018; Schulman et al., 2015, 2017]. More in detail, the chapter is organized as follows.

We address supervised deep learning in Section 2.1, starting from the description of deep feedforward neural networks in Section 2.1.1. Gradient-based training algorithms for feedforward neural networks are reviewed in Section 2.1.2. The back-propagation algorithm, enabling an efficient computation of the gradients required by the gradient-based training algorithms, is illustrated in Section 2.1.3.

The basics of reinforcement learning are covered in Section 2.2. The key concepts characterizing the reinforcement learning setting are introduced in Section 2.2.1. Its formalization using Markov decision processes is tackled in Section 2.2.2. Finally, the policy-gradient algorithms suitable for reinforcement learning agents operating in high-dimensional continuous action spaces are detailed in Section 2.2.3.

2.1 Supervised deep learning

Within the broad field of *machine learning*, aiming at developing algorithms that improve their performances at given tasks by learning from experience or *data*, the nested hierarchical models studied in *deep learning* have seen a tremendous growth in popularity and usefulness in response to the recent noticeable increase of available data and computational power.

As for any other machine learning algorithm, developing a deep learning algorithm involves the design of a certain *model*, the definition of a *loss function* measuring the predictive performances of the model, and the implementation of a *training algorithm* to minimize the selected loss function.

In this brief introduction, we present a combination of model family, loss function and training algorithm that is suitable for tackling *supervised* deep learning, and in particular the *multiple regression* task of interest for this thesis.

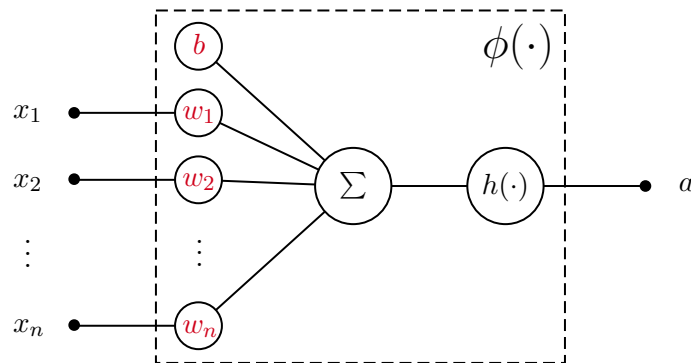


Figure 2.1. Schematic representation of a neuron, the computational unit $\phi(\cdot)$ enclosed by the dashed line mapping a vector input x into a scalar activation a . Within a neuron, a linear combination of the input components x_i with the neuron weights w_i as coefficients, added to the bias b , is mapped to the activation a by the nonlinear activation function $h(\cdot)$. The trainable parameters, i.e., the weights w and the bias b , are highlighted in red.

2.1.1 Feedforward neural networks

Feedforward neural networks – the class of machine learning models at the core of deep learning – can be thought of as function approximators designed to generalize to previously unseen data.

Such models are called *neural* since they are loosely inspired by neuroscience, meaning that their composing units can be interpreted as playing a role analogous to the role of a neuron in a biological system. They are known as *networks* because they are typically structured as a sequence of multiple groups of neurons, referred to as layers, through which the input information flows in order to produce the desired output. Finally, they are termed *feedforward* since the flow of information through the network is one-directional, from the input to the output, with no feedback interconnections among neurons.

Artificial neuron

The elementary unit composing feedforward neural networks is an evolution of the *perceptron*, often referred to as artificial *neuron* [Rosenblatt, 1958; Minsky and Papert, 1969]. Given an input vector $x \in \mathbb{R}^n$, a neuron is defined as the computational unit $\phi : \mathbb{R}^n \mapsto \mathbb{R}$ returning the scalar output or *activation* $a \in \mathbb{R}$ given by

$$a = \phi(x) = h\left(w^\top x + b\right) = h\left(\sum_{i=1}^n w_i x_i + b\right), \quad (2.1.1)$$

where $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$ are the *weights* and *bias*, respectively, while $h : \mathbb{R} \mapsto \mathbb{R}$ is the *activation function*. The neuron weights w and bias b control the affine transformation of the input x , whose result is then mapped to the output y by the nonlinear activation function h . While the weights w and bias b constitute trainable parameters, the activation function h , once selected, usually remains fixed. Figure 2.1 illustrates a schematic representation of a neuron.

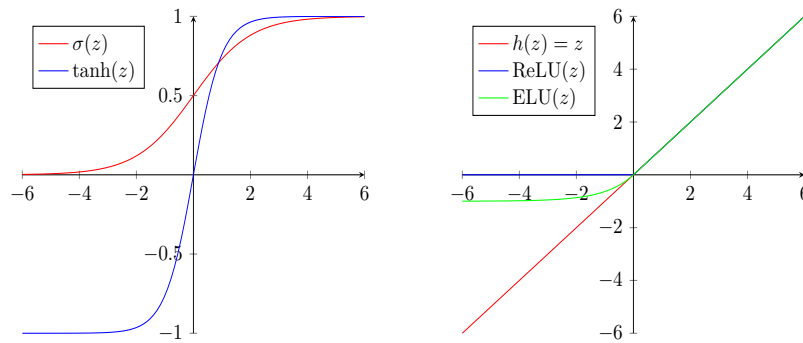


Figure 2.2. Common activation functions. On the left, the logistic sigmoid $\sigma(z)$ compared to the *hyperbolic tangent* $\tanh(z)$. On the right, the *linear unit* compared to the *rectified linear unit* $\text{ReLU}(z)$ and the *exponential linear unit* $\text{ELU}(z)$ (with $\alpha = 1$). The three activation functions coincide for non-negative inputs.

Activation function

The activation function embedded in the artificial neuron – see Eq. (2.1.1) – enables it to perform *nonlinear* transformations, playing therefore a crucial role in the function approximation capabilities of feedforward neural networks. Many functions, recently also trainable ones [Scardapane et al., 2019; Apicella et al., 2021], have been proposed in the literature as nonlinear activation functions [Dubey et al., 2022].

For the purpose of this thesis, let us consider fixed (i.e., non-trainable) activation functions. If $z = w^\top x + b$ is the affine transformation of the input x performed within the neuron, common choices of activation functions $h(z)$ are the *logistic sigmoid*

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad (2.1.2)$$

the *hyperbolic tangent*

$$\tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}, \quad (2.1.3)$$

and the *rectified linear unit* or ReLU [Nair and Hinton, 2010]

$$\text{ReLU}(z) = \begin{cases} z & \text{if } z > 0, \\ 0 & \text{otherwise,} \end{cases} \quad (2.1.4)$$

a piecewise linear function that coincides for positive arguments with the dummy *linear unit* defined by the identity function

$$h(z) = z. \quad (2.1.5)$$

Such activation functions are shown in Figure 2.2, along with the so-called *exponential linear unit* or ELU [Clevert et al., 2016], a generalization of ReLU defined as

$$\text{ELU}(z) = \begin{cases} z & \text{if } z > 0, \\ \alpha(e^z - 1) & \text{otherwise,} \end{cases} \quad (2.1.6)$$

where the default value for the α hyperparameter is $\alpha = 1$.

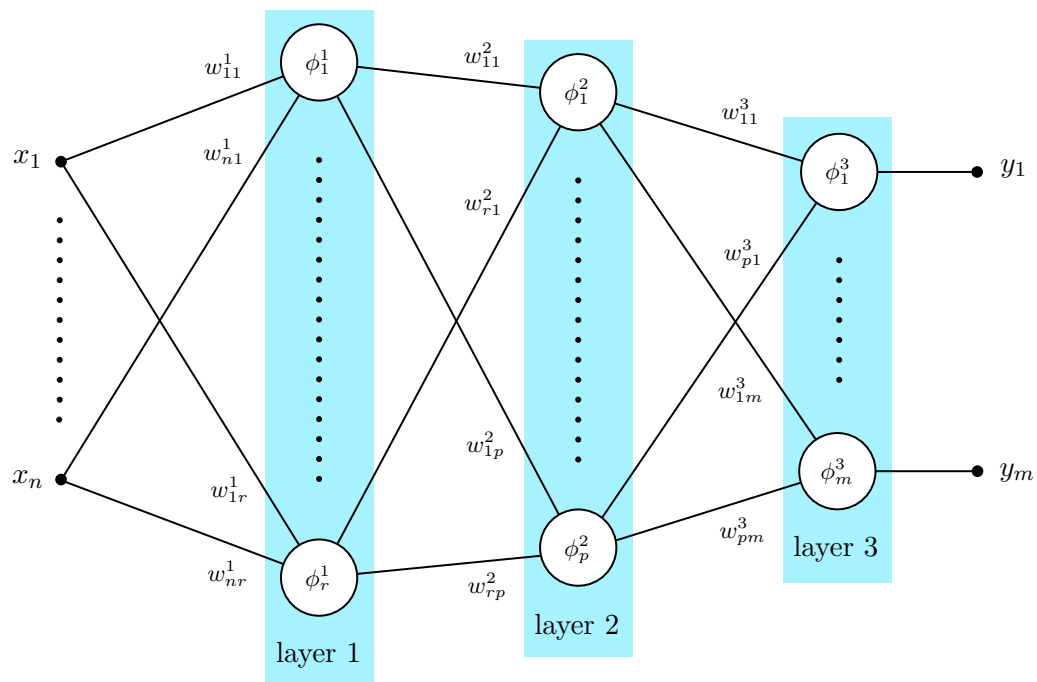


Figure 2.3. Graph representation of a three-layer feedforward neural network mapping the input $x \in \mathbb{R}^n$ to the output $y \in \mathbb{R}^m$. Each node in the graph is a neuron $\phi(\cdot)$. Weights w are associated to the graph edges. Groups of neurons belonging to the same layer are highlighted in blue.

Notice that all the aforementioned activation functions but ReLU are differentiable, a key property for the exploitation of the gradient-based learning algorithms described in Section 2.1.2. In practice, the non-differentiability of ReLU, limited to the singular point $z = 0$, can be however safely disregarded [Goodfellow et al., 2016].

Multi-layer perceptron

Multiple neurons, organized in groups concatenated sequentially, compose a feedforward neural network, also referred to as *multi-layer perceptron*. A multi-layer perceptron approximates a function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ mapping its input $x \in \mathbb{R}^n$ into the output $y \in \mathbb{R}^m$. The groups of neurons chained one after the other in the neural network structure are known as *layers*. The number of neurons in each layer defines the *width* of that layer, while the number of sequentially-chained layers defines the *depth* of the neural network.

Figure 2.3 provides a schematic representation of a generic *three-layer* feedforward neural network in the form of a graph. In this example, the three subsequent layers have a width of r , p and m , respectively. They are therefore composed of r , p and m neurons, respectively, with $r \neq p \neq m$. All the layers but the last one are called *hidden layers*, the last one is referred to as *output layer*.

The i -th neuron of the l -th layer is represented in the graph by the node ϕ_i^l . This is the very same computational unit described in Eq. (2.1.1) and depicted in Figure 2.1, characterized by a bias $b_i^l \in \mathbb{R}$, an activation $h_i^l : \mathbb{R} \mapsto \mathbb{R}$ and a set of

weights $w_i^l \in \mathbb{R}^d$, where d is the dimension of the neuron input. Given the input $x_i^l \in \mathbb{R}^d$, ϕ_i^l returns the scalar activation $a_i^l \in \mathbb{R}$.

Since the network structure is such that the activation a_i^l of the i -th neuron in the l -th layer acts as a component of the input x_j^{l+1} for the j -th neuron in the $(l+1)$ -th layer, the graph representation in Figure 2.3 associates the neuron weights w to the edges connecting neurons in subsequent layers. In particular, the weight w_{ij}^l is associated to the graph edge connecting the i -th neuron of the $(l-1)$ -th layer to the j -th neuron of the l -th layer.

For the sake of simplicity, let us assume that the same activation function h is used for all the neurons in the hidden layers. Moreover, let us consider an output layer with the linear activation function in Eq. (2.1.5). The network output $y \in \mathbb{R}^m$ is obtained by combining the operations performed by the neurons at the different network layers. In particular, the k -th output component y_k for the network in Figure 2.3 is given by

$$y_k = a_k^3 = \sum_{j=1}^p w_{jk}^3 a_j^2 + b_k^3 \quad (2.1.7a)$$

$$= \sum_{j=1}^p w_{jk}^3 h \left(\sum_{i=1}^r w_{ij}^2 a_i^1 + b_j^2 \right) + b_k^3 \quad (2.1.7b)$$

$$= \sum_{j=1}^p w_{jk}^3 h \left(\sum_{i=1}^r w_{ij}^2 h \left(\sum_{q=1}^n w_{qi}^1 x_q + b_i^1 \right) + b_j^2 \right) + b_k^3, \quad (2.1.7c)$$

where x_q is the q -th component of the network input $x \in \mathbb{R}^n$. Let us define the weights and biases of each layer in matrix form as

$$W_1 = \begin{bmatrix} w_{11}^1 & \cdots & w_{n1}^1 \\ \vdots & \ddots & \vdots \\ w_{1r}^1 & \cdots & w_{nr}^1 \end{bmatrix} \in \mathbb{R}^{r \times n}, \quad b_1 = \begin{bmatrix} b_1^1 \\ \vdots \\ b_r^1 \end{bmatrix} \in \mathbb{R}^r, \quad (2.1.8)$$

$$W_2 = \begin{bmatrix} w_{11}^2 & \cdots & w_{r1}^2 \\ \vdots & \ddots & \vdots \\ w_{1p}^2 & \cdots & w_{rp}^2 \end{bmatrix} \in \mathbb{R}^{p \times r}, \quad b_2 = \begin{bmatrix} b_1^2 \\ \vdots \\ b_p^2 \end{bmatrix} \in \mathbb{R}^p, \quad (2.1.9)$$

$$W_3 = \begin{bmatrix} w_{11}^3 & \cdots & w_{p1}^3 \\ \vdots & \ddots & \vdots \\ w_{1m}^3 & \cdots & w_{pm}^3 \end{bmatrix} \in \mathbb{R}^{m \times p}, \quad b_3 = \begin{bmatrix} b_1^3 \\ \vdots \\ b_m^3 \end{bmatrix} \in \mathbb{R}^m. \quad (2.1.10)$$

With an abuse of notation for the activation function h , assumed to be applied component-wise on its argument, Eq. (2.1.7) can be rearranged in matrix form as

$$y = W_3 h(W_2 h(W_1 x + b_1) + b_2) + b_3, \quad (2.1.11)$$

that makes clear the three-stages computation through which the output $y \in \mathbb{R}^m$ is retrieved in the feedforward neural network in Figure 2.3.

Notice that the network architecture in Figure 2.3 is *fully-connected* or *dense*, meaning that all the neurons of the l -th layer are connected to all the neurons of the $(l+1)$ -th layer. Therefore, all the parameters in the set $\theta = \{W_1 \in \mathbb{R}^{r \times n}, W_2 \in \mathbb{R}^{p \times r}, W_3 \in \mathbb{R}^{m \times p}, b_1 \in \mathbb{R}^r, b_2 \in \mathbb{R}^p, b_3 \in \mathbb{R}^m\}$ represent trainable parameters for the network. Although a wide variety of extensions to the fully-connected architecture has been proposed in the literature, we limit this brief introduction to the fully-connected neural networks of interest for this thesis.

Universal approximation theorem

The function approximation capabilities of feedforward neural networks rely upon the *universal approximation theorem* [Cybenko, 1989; Hornik et al., 1989].

The theorem states that a feedforward neural network with an output layer characterized by a linear activation such as the one in Eq. (2.1.5) and a hidden layer with a nonlinear activation of the kind of the logistic sigmoid in Eq. (2.1.2) is able to approximate *any* continuous function¹ from one finite-dimensional space to another, provided that the hidden layer is composed of *enough* neurons. The theorem has been extended to a broader class of activation functions, including the widely used ReLU in Eq. (2.1.4) [Leshno et al., 1993].

In other words, the theorem states that no matter the continuous function to be approximated there exists a large-enough two-layer feedforward neural network capable of *representing* it. However, this does not guarantee that the training algorithm – see Section 2.1.2 – will be able to find the actual values of the network trainable parameters ensuring the desired approximation accuracy. Moreover, no indications are given on the actual width of the hidden layer, that could also be unfeasibly large for the desired function. In practice, it could be more convenient to represent the same function by *deeper* architectures – such as the three-layer neural network in Figure 2.3 – with more layers but a reduced number of neurons per layer and, therefore, a possibly lower overall number of parameters to be trained.

2.1.2 Neural networks training

Training algorithms for feedforward neural networks aim at finding values for their trainable parameters θ such that the network generalizes well, i.e., it achieves the best possible approximation accuracy on unseen data. Depending on the network architecture and the function to be approximated, several approaches are possible [Goodfellow et al., 2016].

In the following, we briefly describe the training algorithm suitable for the *multiple regression* task of interest for this thesis, targeting the approximation of a function $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ mapping an input $x \in \mathbb{R}^n$ into an output $y \in \mathbb{R}^m$. Given the task, we consider feedforward neural networks with an output layer whose neurons are characterized by the linear activation function – see Eq. (2.1.5).

¹In particular, the universal approximation theorem refers to *Borel measurable* functions, which include the continuous functions on a closed and bounded subset of \mathbb{R}^n of interest for this overview.

Maximum likelihood estimation

From a probabilistic perspective, a feedforward neural network parameterized by θ produces an output $y \in \mathbb{R}^m$, given the input $x \in \mathbb{R}^n$, by estimating the conditional probability $p(y|x; \theta)$.

Consider a supervised training dataset of N datapoints composed of the inputs $X = \{x^1, \dots, x^N\}$, with $x^i \in \mathbb{R}^n$, and the correspondent observed targets $Y = \{y^1, \dots, y^N\}$, with $y^i \in \mathbb{R}^m$. The parameters θ such that the observed data is most likely, i.e., the parameters θ obtained by following the Maximum Likelihood Estimation (MLE) approach, are given by

$$\theta_{MLE} = \arg \max_{\theta} p(Y|X; \theta). \quad (2.1.12)$$

By assuming the N datapoints independent and identically distributed, i.e., i.i.d., Eq. (2.1.12) can be decomposed into

$$\theta_{MLE} = \arg \max_{\theta} \prod_{i=1}^N p(y^i|x^i; \theta), \quad (2.1.13)$$

and, by taking the logarithm (that does not change the argmax, being a monotonically increasing function), further developed as

$$\theta_{MLE} = \arg \max_{\theta} \sum_{i=1}^N \log p(y^i|x^i; \theta). \quad (2.1.14)$$

The so-called *log-likelihood* in Eq. (2.1.14) can be then maximized to seek optimal values for θ . Rather than maximizing the log-likelihood, the neural networks literature often considers the (equivalent) minimization of the negative log-likelihood, choosing therefore as *loss function*

$$J(\theta) = - \sum_{i=1}^N \log p(y^i|x^i; \theta), \quad (2.1.15)$$

whose specific form depends on the definition of the parametric family of probability distributions $p(y|x; \theta)$.

Now, assume the neural network to predict the mean of an isotropic m -dimensional Gaussian distribution, meaning that

$$p(y|x; \theta) = \mathcal{N}(y; \hat{y}(x; \theta), \sigma^2 I_m), \quad (2.1.16)$$

where $\Sigma = \sigma^2 I_m \in \mathbb{R}^{m \times m}$ is the fixed diagonal Gaussian covariance while its θ -dependent mean is the network prediction $\hat{y}(x; \theta) \in \mathbb{R}^m$ – see for instance Eq. (2.1.11). Under this assumption, the negative log-likelihood takes the form

$$J(\theta) = -N \log \sigma - \frac{N}{2} \log(2\pi) - \frac{1}{2\sigma^2} \sum_{i=1}^N \left\| \hat{y}^i(x^i; \theta) - y^i \right\|^2. \quad (2.1.17)$$

Once constants and terms that do not depend on θ are discarded, it is clear that minimizing Eq. (2.1.17) is equivalent to minimizing the Mean Squared Error (MSE)

$$J_{MSE}(\theta) = \frac{1}{N} \sum_{i=1}^N \left\| \hat{y}^i(x^i; \theta) - y^i \right\|^2. \quad (2.1.18)$$

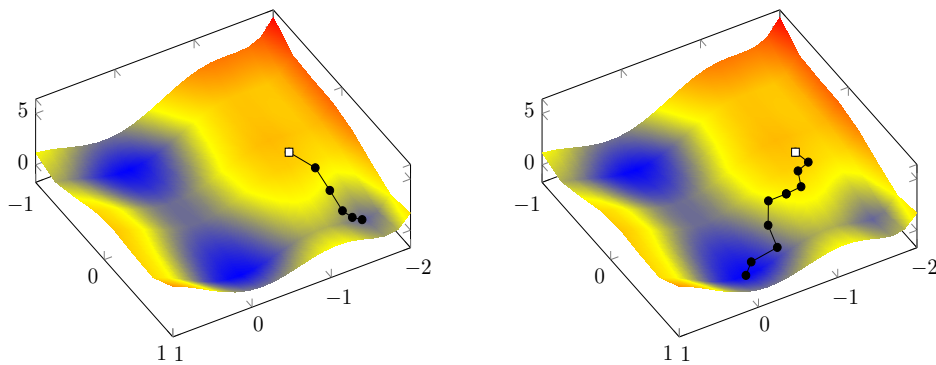


Figure 2.4. Optimization paths resulting from an execution of gradient descent (left) and SGD (right) on a two-dimensional loss function, from the same initial point (\square). Gradient descent smoothly follows the negative gradient direction, reaching a local minimum. Thanks to the variability of the update performed on the minibatches, in this particular execution SGD escapes the local minimum and reaches the global one.

Therefore, the MSE represents a suitable loss function to be minimized when training a feedforward neural network with linear output activations that is assumed to predict the mean of an isotropic multivariate Gaussian with fixed covariance as a solution for a multiple regression task. The same approach can be easily extended, with other output activations, for more general and even θ -dependent covariances [Goodfellow et al., 2016; Bishop, 2006].

To improve the network generalization performances, $J_{\text{MSE}}(\theta)$ is often minimized along with the *weight decay*² regularization term $J(w)$ encouraging smaller L2-norm for the network weights w . The final loss becomes therefore

$$J(\theta) = J_{\text{MSE}}(\theta) + J(w) \quad (2.1.19a)$$

$$= J_{\text{MSE}}(\theta) + \lambda w^\top w, \quad (2.1.19b)$$

where λ is the parameter weighting the relative contribution of the regularization term $J(w)$ to the overall loss function $J(\theta)$.

Stochastic gradient descent

The non-linearity of feedforward neural networks leads to the minimization of non-convex loss functions, an optimization problem that can be addressed by iterative numerical procedures. Despite coming with no convergence guarantees, such procedures applied to deep neural networks proved to be effective in driving the loss function to reasonably low values, quickly enough to be useful.

Given a loss function $J(\theta)$ and an initial value θ_0 for its parameters, *gradient descent* aims to iteratively decrease $J(\theta)$ by taking small steps in the direction of its negative gradient – see Figure 2.4 (left). This results in the iterative update of the current estimate of θ by means of the update rule

$$\theta_{i+1} = \theta_i - \eta \nabla_{\theta} J(\theta), \quad (2.1.20)$$

where $\eta \in \mathbb{R}_+$ is the *learning rate*, a positive scalar determining the size of the step.

²For many other regularization techniques for deep neural networks, out of scope for this brief overview, please refer to [Goodfellow et al., 2016, Chapter 7].

Gradient descent applied to deep neural networks, however, is particularly challenging due to the many local minima and flat regions of the loss functions to be minimized, in which the gradient vanishes, invalidating the iterative update rule. Moreover, since $J(\theta)$ is defined on the entire training set, each update of the parameters θ requires the entire training set to be processed for the evaluation of $\nabla_{\theta}J(\theta)$ – gradient descent is indeed a *batch* method.

An approximation of gradient descent, known as *stochastic gradient descent*³ (SGD), obtains an estimate $\nabla_{\theta}\hat{J}(\theta)$ of the gradient by randomly sampling from the training set a *minibatch* of k samples $\{x^i, y^i\}$, with $i \in \{1, \dots, k\}$, and taking the average gradient on the minibatch only. This results in the iterative update rule

$$\theta_{i+1} = \theta_i - \eta \nabla_{\theta} \hat{J}(\theta) \quad (2.1.21a)$$

$$= \theta_i - \frac{\eta}{k} \sum_{i=1}^k \nabla_{\theta} J(x^i, y^i; \theta), \quad (2.1.21b)$$

where $J(x^i, y^i; \theta)$ is the per-sample loss, e.g., for the MSE

$$J_{MSE}(x^i, y^i; \theta) = \left\| \hat{y}^i(x^i; \theta) - y^i \right\|^2. \quad (2.1.22)$$

The minibatch size k is fixed and typically relatively small. Therefore, no matter the dimension N of the training set, SGD has a fixed computation time per update that allows for feasible execution even with very large training sets. Moreover, since in general stationary points of the entire training set are not stationary points of the k -dimensional minibatches, SGD is more likely to escape from the local minima of the overall loss function – see Figure 2.4.

Applied to non-convex loss functions, SGD is strongly affected by the initialization of the parameters θ , which can determine whether, how quickly, and where the training converges. Initialization heuristics for feedforward neural networks suggest zero or small positive values as initial values for the network biases b , and small random values for the network weights w . In particular, the *normalized initialization* [Glorot and Bengio, 2010] for the weights of a network mapping $x \in \mathbb{R}^n$ into $y \in \mathbb{R}^m$ suggests to sample them from the uniform distribution

$$w \sim U\left(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}}\right). \quad (2.1.23)$$

Another key parameter for SGD is the learning rate η in Eq. (2.1.21), that is usually gradually decreased over time not to prevent convergence of the algorithm. A simple heuristic decreases η linearly from an initial value η_0 to a final value η_{τ} in τ iterations, so that for the first $k < \tau$ iterations the learning rate is

$$\eta_k = (1 - \alpha)\eta_0 + \alpha\eta_{\tau}, \quad (2.1.24)$$

with $\alpha = k/\tau$, while for the remaining $k \geq \tau$ iterations the learning rate $\eta_k = \eta_{\tau}$ remains constant. More advanced algorithms such as AdaGrad [Duchi et al., 2011] or ADAM [Kingma and Ba, 2015] use separate learning rates for each optimization parameter and automatically adapt the learning rates throughout the learning process, leading to an improved robustness. However, no best algorithm emerges when comparing different optimizers over a wide range of tasks [Schaul et al., 2014].

³Also referred to as *stochastic gradient method*, since it is not guaranteed to descend at each step even for convex loss functions.

2.1.3 Backpropagation algorithm

Neural networks training requires computing the gradients of complicated loss functions – see Eq. (2.1.21) for SGD. The backpropagation algorithm provides a simple and computationally efficient procedure to numerically evaluate these gradients [Rumelhart et al., 1986].

Let us consider the generic three-layer feedforward neural network in Figure 2.3. For the sake of simplicity, let us assume linear activation functions for all the neurons in the output layer and identical non-linear activation functions $h(\cdot)$ for all the neurons in the hidden layers.

Recall that in this architecture the i -th neuron of the l -layer is denoted by ϕ_i^l and constitutes the computational unit

$$a_i^l = \phi_i^l(x_i^l) = h(z_i^l) = h\left(\sum_{j=1}^k w_{ji}^l a_j^{l-1} + b_i^l\right), \quad (2.1.25)$$

where k is the width of the $(l-1)$ -th layer whose neurons ϕ_j^{l-1} are connected to ϕ_i^l , with the weights w_{ji}^l associated to the correspondent connections.

Consider the per-sample gradient $\nabla_{\theta} J(x^i, y^i; \theta)$ in Eq. (2.1.21) to be evaluated at the iterative update of SGD for each sample $\{x^i, y^i\}$ in the selected minibatch. For the sake of clarity, let us simplify the notation and refer to the current sample as $\{x, y\}$ and to its associated per-sample gradient as $\nabla_{\theta} J$. The backpropagation algorithm provides a procedure, based on the chain rule of calculus, to efficiently evaluate this gradient in two alternate stages that involve a forward and a backward flow of information through the network, respectively.

Forward propagation

The first stage of the backpropagation algorithm is the so-called *forward propagation*. It simply consists in feeding the network with the current input and letting the information propagate forward through the network layers till the computation of the network prediction along with its associated loss.

In particular, given the current value of θ and the input $x \in \mathbb{R}^n$, Eq. (2.1.7) shows how to compute each component of the network prediction $\hat{y}(x; \theta) \in \mathbb{R}^m$ – in the following simply referred to as \hat{y} – via the computation of the intermediate activations $a_i^l \in \mathbb{R}$ of each neuron ϕ_i^l in the architecture.

Comparing $\hat{y} \in \mathbb{R}^m$ with the current target $y \in \mathbb{R}^m$, the per-sample loss $J(x, y; \theta)$ can be retrieved. Let us assume here the MSE loss with no regularization terms from Eq. (2.1.18), leading to the per-sample loss

$$J_{MSE}(x, y; \theta) = \|\hat{y} - y\|^2 = \sum_{k=1}^m (\hat{y}_k - y_k)^2, \quad (2.1.26)$$

where m is the number of neurons in the output layer, each returning one component of the network prediction $\hat{y} \in \mathbb{R}^m$.

Backward propagation

The second stage of the backpropagation algorithm is the *backward propagation*, hence its name. It consists in evaluating the partial derivatives in the per-sample gradient $\nabla_{\theta} J$ starting from the output layer and propagating the information backward through the network, by taking advantage of the chain rule of calculus.

Recall that $\nabla_{\theta} J$ is by definition the vector of partial derivatives of the loss with respect to each parameter, i.e., each weight and bias, in the network. For the network in Figure 2.3, it is defined as

$$\nabla_{\theta} J = \left[\frac{\partial J}{\partial w_{11}^1} \cdots \frac{\partial J}{\partial w_{pm}^3} \frac{\partial J}{\partial b_1^1} \cdots \frac{\partial J}{\partial b_m^3} \right]^{\top}. \quad (2.1.27)$$

Notice that the per-sample loss J depends on the weight w_{ji}^l only through the computation taking place at the neuron ϕ_i^l described in Eq. (2.1.25). Therefore, for the chain rule of calculus, the partial derivative of the per-sample loss with respect to a weight w_{ji}^l is given by

$$\frac{\partial J}{\partial w_{ji}^l} = \frac{\partial J}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ji}^l}. \quad (2.1.28)$$

Similarly, for the bias b_i^l we have

$$\frac{\partial J}{\partial b_i^l} = \frac{\partial J}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial z_i^l}{\partial b_i^l}. \quad (2.1.29)$$

Considering the MSE loss in Eq. (2.1.26), for an output neuron ϕ_i^3 in Figure 2.3 with linear activation function Eq. (2.1.28) translates into

$$\frac{\partial J}{\partial w_{ji}^3} = \frac{\partial J}{\partial a_i^3} \frac{\partial a_i^3}{\partial z_i^3} \frac{\partial z_i^3}{\partial w_{ji}^3} = \frac{\partial J}{\partial a_i^3} \frac{\partial a_i^3}{\partial z_i^3} a_j^2 \quad (2.1.30a)$$

$$= \frac{\partial J}{\partial a_i^3} a_j^2 \quad (2.1.30b)$$

$$= 2(\hat{y}_i - y_i) a_j^2, \quad (2.1.30c)$$

where a_j^2 is the activation of the j -th neuron of the second layer ϕ_j^2 connected to the i -th neuron of the output layer ϕ_i^3 through the weight w_{ji}^3 . Accordingly, Eq. (2.1.29) translates into

$$\frac{\partial J}{\partial b_i^3} = \frac{\partial J}{\partial a_i^3} \frac{\partial a_i^3}{\partial z_i^3} \frac{\partial z_i^3}{\partial b_i^3} = \frac{\partial J}{\partial a_i^3} \frac{\partial a_i^3}{\partial z_i^3} \quad (2.1.31a)$$

$$= \frac{\partial J}{\partial a_i^3} \quad (2.1.31b)$$

$$= 2(\hat{y}_i - y_i). \quad (2.1.31c)$$

For a hidden neuron ϕ_i^l in Figure 2.3 with activation function h , instead, Eq. (2.1.28) can be developed as

$$\frac{\partial J}{\partial w_{ji}^l} = \frac{\partial J}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ji}^l} = \frac{\partial J}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l} a_j^{l-1} \quad (2.1.32a)$$

$$= \frac{\partial J}{\partial a_i^l} h'(z_i^l) a_j^{l-1}, \quad (2.1.32b)$$

where a_j^{l-1} is the activation of the j -th neuron of the $(l-1)$ -th layer ϕ_j^{l-1} connected to the i -th neuron of the l -th layer ϕ_i^l through the weight w_{ji}^l .

Since the per-sample loss J depends on the activation a_i^l of the neuron ϕ_i^l through all the activations a_k^{l+1} of the n_{l+1} neurons ϕ_k^{l+1} of the $(l+1)$ -layer to which ϕ_i^l is connected, the first term in Eq. (2.1.32b) can be further developed by making use of the chain rule as

$$\frac{\partial J}{\partial a_i^l} = \sum_{k=1}^{n_{l+1}} \frac{\partial J}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial a_i^l} \quad (2.1.33a)$$

$$= \sum_{k=1}^{n_{l+1}} \frac{\partial J}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l} \quad (2.1.33b)$$

$$= \sum_{k=1}^{n_{l+1}} \frac{\partial J}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} w_{ik}^{l+1} \quad (2.1.33c)$$

$$= \sum_{k=1}^{n_{l+1}} \frac{\partial J}{\partial a_k^{l+1}} h'(z_k^{l+1}) w_{ik}^{l+1}. \quad (2.1.33d)$$

Notice how the partial derivative of the per-sample loss J with respect to a_i^l depends on its n_{l+1} partial derivatives with respect to a_k^{l+1} , already known if the computation starts from the output layer and proceeds backward.

Finally, Replacing Eq. (2.1.33d) into Eq. (2.1.32b), the expression of Eq. (2.1.28) for the hidden neuron ϕ_i^l becomes

$$\frac{\partial J}{\partial w_{ji}^l} = h'(z_i^l) a_j^{l-1} \sum_{k=1}^{n_{l+1}} \frac{\partial J}{\partial a_k^{l+1}} h'(z_k^{l+1}) w_{ik}^{l+1}. \quad (2.1.34)$$

Through similar computations, the expression of Eq. (2.1.29) for the hidden neuron ϕ_i^l is given by

$$\frac{\partial J}{\partial b_i^l} = h'(z_i^l) \sum_{k=1}^{n_{l+1}} \frac{\partial J}{\partial a_k^{l+1}} h'(z_k^{l+1}) w_{ik}^{l+1}. \quad (2.1.35)$$

In summary, all the components of the per-sample gradient $\nabla_{\theta} J$ in Eq. (2.1.27) can be evaluated by computing first Eqs. (2.1.30) and (2.1.31) for each neuron of the output layer and then Eqs. (2.1.34) and (2.1.35) for each neuron of each hidden layer, proceeding backwards.

The resulting per-sample gradient $\nabla_{\theta} J$, summed up for all the samples in the selected minibatch, is then exploited to perform SGD – see Eq. (2.1.21).

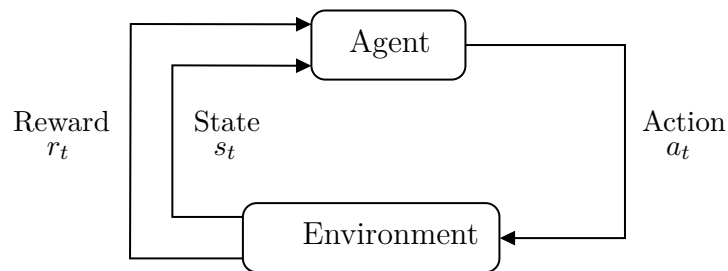


Figure 2.5. A sketch of the interaction loop taking place between the agent and the environment in a RL setting. Given the environment state s_t at time t , the learning agent selects an action a_t whose application influences the next environment state and results in the collection of the reward r_t .

2.2 Reinforcement learning

Reinforcement learning (RL) is the machine learning paradigm dealing with decision-making agents that interact with an uncertain environment with the aim of achieving well-defined long-term goals or tasks [Sutton and Barto, 2018]. To improve its task-related performances over time, a RL agent can take actions that have an influence on the environment and then judge its progress by sensing information from the environment. The loop arising from this repeated interaction between the agent and the environment is sketched in Figure 2.5.

In the following, we detail the fundamental concepts required to formalize a general RL problem and then focus on the specific knowledge relevant to understand the contribution of this thesis related to RL. In particular, for the application of RL to the robotics domain addressed in this work, we recall *deep* reinforcement learning (DRL) algorithms for *continuous* control, i.e., policy-gradient algorithms that optimize a stochastic policy operating in a continuous action space and represented with a deep neural network – see Section 2.1.

2.2.1 RL basics

In the RL setting depicted in Figure 2.5, the *agent* and the *environment* represent abstract entities whose boundary varies depending on the given problem. No matter the definition of the learning problem, the RL paradigm assumes both the environment and the agent to be characterized by certain essential features that allow to properly formalize their interaction, as detailed in the remaining of this section. The trial-and-error process arising from the agent-environment interaction, through which the agent accumulates useful experience to progressively improve towards its task, is a distinctive feature of the RL paradigm.

Notice that the entire RL paradigm relies on the so-called *reward hypothesis*, i.e., the assumption that any task can be formulated in terms of the maximization of a cumulative *scalar* signal, the reward. Although at a first glance this may seem a significantly limiting constraint, in practice the RL framework has proven flexible and powerful enough to be even regarded as a suitable problem formulation for artificial general intelligence [Legg and Hutter, 2007; Silver et al., 2021].

Environment

In the RL setting depicted in Figure 2.5, the environment is the abstract entity responsible for the evolution of the system the agent interacts with, and for the generation of the reward the agent seeks to maximize over time.

At time t , the *state* $s_t \in \mathcal{S}$ fully characterizes the environment dynamics. Once a certain state s_t has been reached, the future evolution of the environment is independent from its previous states s_{t-i} , with $i \in \mathbb{N}_+$, $i < t$. If \mathcal{S} is a non-Euclidean space, we define the *observation* $o \in \mathcal{O} = \mathbb{R}^n$ as the output of the function $o : \mathcal{S} \mapsto \mathcal{O}$ that maps states into elements of an Euclidean space.

The environment state s_t (or the observation o_t) is available for the agent to select its action $a_t \in \mathcal{A}$, whose choice results into the environment evolving to the state s_{t+1} . In *stochastic* settings, this evolution takes place according to the *state-transition probability function* $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \Pr[\mathcal{S}]$, i.e.,

$$s_{t+1} \sim \mathcal{P}(s \mid s_t, a_t). \quad (2.2.1)$$

If \mathcal{S} includes special states, called *terminal* states, that can be associated to a failure or a success for the task the agent is trying to learn (e.g., winning or losing a game), such a task is referred to as an *episodic* task. In this case, the agent-environment interaction naturally breaks into episodes starting at the randomly-sampled initial state s_0 and ending at a terminal state s_T , with the episode *duration* T potentially varying between episodes. On the contrary, tasks for which no terminal states determining failure or success exist are referred to as *continuing* tasks. In continuing tasks, the agent-environment interaction proceeds seamlessly, with no time limit, i.e., $T = \infty$ for continuing tasks⁴.

The environment is also in charge of generating the *immediate reward* $r_t \in \mathbb{R}$, a scalar feedback signal that represents the basis for the agent to evaluate its choice of actions with respect to the given task. The immediate reward is the output of the *reward function* $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$, i.e.,

$$r_t = \mathcal{R}(s_t, a_t, s_{t+1}). \quad (2.2.2)$$

Notice that the immediate reward r_t depends upon the current action a_t but also upon the current state s_t , which is the result of all the previous actions performed by the agent. In other words, the actions performed by the agent not only affect the immediate reward but also – through future environment states – all the subsequent rewards. Such an indirect impact of actions on *delayed* rewards is a crucial feature of the RL setting, along with the assumption of both the state-transition probability function \mathcal{P} and the reward function \mathcal{R} being unknown.

To consider the impact of actions on delayed rewards, it is particularly useful to introduce the *return* G_t at time t , that is, the actual quantity the agent seeks to maximize. For episodic tasks, the return G_t is defined as the sum of the immediate rewards from the current time till the end of the episode, i.e.,

$$G_t = r_t + r_{t+1} + r_{t+2} + \dots + r_T = \sum_{k=0}^{T-t} r_{t+k}. \quad (2.2.3)$$

⁴In some cases, continuing tasks are then truncated at a finite time T , but such T is only defined in terms of maximum length of the episode and is not related to the state s_T reached by the agent at time T , since there are no terminal states for continuing tasks.

Applied to continuing tasks, where $T = \infty$, the definition in Eq. (2.2.3) would lead to infinite returns. In this case, it is more appropriate to define the *discounted return* G_t at time t as an exponentially-weighted sum of immediate rewards over time, i.e.,

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \quad (2.2.4)$$

where $\gamma \in [0, 1)$ is the *discount factor*, whose exponential weights the sum of rewards. The discount factor γ , assumed to belong to $[0, 1)$ in order to ensure a finite discounted return G_t , determines the current value of future rewards. With γ approaching 0, future rewards give indeed a very little contribution to G_t , that coincides with the current reward r_t in the extreme case in which $\gamma = 0$. The more γ approaches 1, the more future rewards contribute to G_t . Notice that the discounted return G_t in Eq. (2.2.4) can also be expressed recursively as

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \dots) = r_t + \gamma G_{t+1}. \quad (2.2.5)$$

Finally, the return definitions for the episodic and continuing tasks, i.e., Eq. (2.2.3) and Eq. (2.2.4), respectively, can be unified by writing G_t as

$$G_t = \sum_{k=0}^T \gamma^k r_{t+k}, \quad (2.2.6)$$

with either $T = \infty$ (continuing tasks) or $\gamma = 1$ (episodic tasks), but not both, since $T = \infty$ and $\gamma = 1$ would lead to infinite returns.

Agent

In the RL setting depicted in Figure 2.5, the agent is the abstract entity that attempts to learn over time, by interacting with the environment, how to maximize the cumulative (discounted) reward or, more precisely, the expected (discounted) return. The agent is the decision maker, i.e., the RL algorithm that, depending on its learning objectives and the information about the environment it has available, is characterized as follows.

At time t , the agent selects the action $a_t \in \mathcal{A}$ to perform. This action is chosen according to the agent's behaviour function, that is, the *policy*. A *stochastic* policy is defined as the probability distribution $\pi : \mathcal{S} \mapsto \text{Pr}[\mathcal{A}]$ mapping states into probabilities of selecting each possible action. Given the current state s_t , the action a_t is then sampled from π as

$$a_t \sim \pi(a|s_t). \quad (2.2.7)$$

In the *high-dimensional continuous* state and action spaces of interest for this thesis, a common choice for π is the isotropic m -dimensional Gaussian distribution

$$\pi(a|s_t) = \mathcal{N}(a; \mu(s_t), \sigma^2 I_m), \quad (2.2.8)$$

where $\Sigma = \sigma^2 I_m \in \mathbb{R}^{m \times m}$ is the Gaussian fixed diagonal covariance while $\mu(s_t) \in \mathbb{R}^m$ is the Gaussian state-dependent mean to be learned by the agent, with m being the dimension of the action space. In particular, if the policy is modeled as a deep

neural network parameterized by θ that takes as input the environment state s_t and produces as output the mean of the isotropic Gaussian $\mu_\theta(s_t)$, as in Eq. (2.1.16), we describe the policy as

$$\pi_\theta(a|s_t) = \mathcal{N}(a; \mu_\theta(s_t), \sigma^2 I_m), \quad (2.2.9)$$

where the dependency of π and μ on θ is denoted by the subscript.

Another relevant concept for a RL agent is the *value* of either a state s_t or a state-action pair (s_t, a_t) . The value of a state (or a state-action pair), defined with respect to a policy π , expresses how good it is for the agent to be in that state (or to perform that action in that state). In other words, it estimates the cumulative future reward obtained from that state (or by performing that action in that state), assuming that the policy π is followed from there on. Whereas the reward, provided by the environment, gives the agent a *short-term* feedback about being in a certain state, the value of a state (or a state-action pair), learned by the agent itself, encodes the *long-term* utility of that state (or of performing that action in that state). For instance, a defense move in a game, which gives zero or low immediate reward, can be motivated by the value of the state it allows to reach (which may help chances of winning later, that is the actual goal of the agent playing the game).

Despite providing potentially useful information to take decisions, state values (or, more-commonly, state-action pair values) are not taken into account by all the RL agents. An agent can indeed be classified as policy-based or value-based, with actor-critic agents standing in the middle, as follows:

- *Value-based* agents aim to learn state-action pair values and derive their policy implicitly from them. These agents do not scale well to high-dimensional state spaces and are not compatible with continuous actions, therefore their application in the robotics domain is severely limited.
- *Policy-based* agents directly parameterize a policy and search for its optimal parameters, completely ignoring the estimation of state-action pair values. These agents naturally handle continuous actions and scale well to high-dimensional state spaces, which let them gain popularity in robotics research.
- Finally, *actor-critic* agents aim to learn the policy (actor) along with the state values (critic), both parameterized, typically as deep neural networks, and sometimes also sharing parameters. By combining the advantages of policy-based or *actor-only* and value-based or *critic-only* agents, actor-critic agents often represent a remarkably suitable option when applying RL to the robotics domain.

Furthermore, RL agents may leverage a *model* of the environment they interact with. In this case, they are classified as *model-based* agents. Such a model allows the agent to predict the effects of its actions. An accurate model can be used to efficiently guide the policy search and easily adapt to new tasks (in state-action regions close to those visited by the agent) without additional experience. However, incorrect models end up biasing what the agent learns. On the other hand, agents that do not use any model of the environment and entirely learn from experience are referred to as *model-free* agents. Despite typically requiring a large amount of experience to learn properly, model-free agents are more suitable for scenarios in which the environment dynamics is too complex to be modeled accurately.

Agent-environment interaction

The interaction loop in Figure 2.5 between the agent and the environment results in a sequence of states and actions from the initial state s_0 to the state s_T ⁵. We denote this sequence as the *trajectory* τ – often also called *rollout* – given by

$$\tau = (s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T), \quad (2.2.10)$$

where the initial state s_0 is randomly sampled from the *initial state distribution* $\rho_0 : \mathcal{S} \mapsto \Pr(\mathcal{S})$, i.e.,

$$s_0 \sim \rho_0(s). \quad (2.2.11)$$

At time t , given the state s_t , the agent selects the action a_t according to its policy $\pi(a|s_t)$. The selected action results into the evolution of the environment at the state s_{t+1} and the collection of the reward r_t ⁶. Therefore, trajectories can also be expressed in terms of state-action-reward tuples as

$$\tau = \{(s_0, a_0, r_0), (s_1, a_1, r_1), \dots, (s_{T-1}, a_{T-1}, r_{T-1}), (s_T, \emptyset, 0)\}, \quad (2.2.12)$$

where the last element contains no action and zero reward since the agent-environment interaction ends at s_T (the last reward is r_{T-1}).

Eq. (2.2.6) applied to the trajectory τ leads to the definition of the discounted return $G(\tau)$ associated to τ , i.e.,

$$G(\tau) = \sum_{t=0}^T \gamma^t r_t, \quad (2.2.13)$$

where in the specific case of the truncated continuing tasks of interest for this thesis we assume a finite T and $\gamma < 1$.

A key feature of the RL setting is that the agent learns how to optimize its policy *while* interacting with the environment, i.e., while collecting trajectories τ_i in the form of Eq. (2.2.12) and their associated returns $G(\tau_i)$ computed as in Eq. (2.2.13). Trajectories $\{\tau_0, \dots, \tau_N\}$ and their associated returns $\{G(\tau_0), \dots, G(\tau_N)\}$ represent the experience that the agent can exploit to optimize its policy with respect to the task once N trajectories have been executed, with N increasing as the learning process progresses. Since the policy π varies over time as an effect of the optimization process, the trajectories τ_i included in the stored experience are associated to different policies. The ability to use data gathered using different policies differentiate on-policy and off-policy training algorithms for the agent. In particular, *on-policy* algorithms optimize the current policy by using data collected according to that very same policy, while *off-policy* algorithms can exploit data generated by following any arbitrary policy in the same environment. Since after each policy optimization step all the previously-collected data become off-policy, solely off-policy algorithms actually learn from the entire experience accumulated over time, while on-policy algorithms learn from recently-sampled trajectories only. Therefore, off-policy algorithms are characterized by a higher sample efficiency with respect to on-policy ones.

⁵The state s_T is a terminal state for episodic tasks, while it represents the state at which the agent-environment interaction is truncated for truncated continuing tasks.

⁶In the RL literature, the reward generated after performing action a_t in state s_t can be denoted either as r_t or r_{t+1} . We adopt the former notation here.

2.2.2 Markov decision processes

The agent-environment interaction that characterizes the RL setting can be formalized by making use of the *Markov Decision Processes* (MDPs). These classical formulations of sequential decision-making problems capture indeed all the essential features of the RL setting described in the previous section. Given

- The set of valid states \mathcal{S} ;
- The set of valid actions \mathcal{A} ;
- The state-transition probability function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \Pr[\mathcal{S}]$;
- The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$;
- The initial state distribution $\rho_0 : \mathcal{S} \mapsto \Pr(\mathcal{S})$;
- The discount factor γ ;

we define a MDP as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma \rangle$. Being Markov processes, MDPs satisfy the *Markov property*, stating that future states are conditionally independent from past states given the current state. This translates into the dynamics of the MDP being completely defined by the state-transition probability function $\mathcal{P}(s_{t+1} | s_t, a_t)$ that relates the next state s_{t+1} to the previous state s_t and action a_t only, with no dependency on the past states.

We have introduced in the previous section the concept of value of a state or state-action pair. Within the MDP formulation, the value is defined in terms of expected return. Given the policy π , the *state-value function* $V^\pi(s)$ for policy π associates to the state s the expected return from s assuming that the agent follows the policy π . It is defined as

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t \right]. \quad (2.2.14)$$

Similarly, the *action-value function* $Q^\pi(s, a)$ for policy π associates to the state-action pair (s, a) the expected return from s assuming that the agent performs a and then follows the policy π . It is defined as

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t, a_t] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t, a_t \right]. \quad (2.2.15)$$

Moreover, from the state-value and action-value functions we define the *advantage function* $A^\pi(s, a)$ for policy π as

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s). \quad (2.2.16)$$

This function provides a relative measure of the quality of the action a in state s with respect to the average action quality in s . If the action a performs better than average in the state s , then $Q^\pi(s, a) > V^\pi(s)$ and therefore a positive advantage $A^\pi(s, a) > 0$ is associated to the state-action pair (s, a) . On the contrary, worse-than-average actions a in the state s correspond to a negative advantage $A^\pi(s, a) < 0$.

State-value and action-value functions allow to compare policies. A policy π' is considered better than or equal to a policy π , i.e., $\pi' \geq \pi$, if and only if $V^{\pi'}(s) \geq V^\pi(s)$ for all the states $s \in \mathcal{S}$. Let us denote by π^* the *optimal policy* of a MDP, i.e., the policy that is better than or equal to all the other policies. There exists always at least one π^* and it may be not unique. Then the optimal state-value function $V^*(s)$ for policy π^* is defined as

$$V^*(s) = \max_{\pi} V^\pi(s). \quad (2.2.17)$$

Accordingly, the optimal action-value function $Q^*(s, a)$ for policy π^* is defined as

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a). \quad (2.2.18)$$

Bellman expectation equations

By applying the recursive expression of the return in Eq. (2.2.5) to the definition of the state-value function in Eq. (2.2.14), we get a recursive relationship for the state-value function known as *Bellman expectation equation for V^π* . It relates the value of a state s_t with the value of all its possible successor states as

$$V^\pi(s_t) = \mathbb{E}_\pi[G_t|s_t] \quad (2.2.19a)$$

$$= \mathbb{E}_\pi[r_t + \gamma G_{t+1}|s_t] \quad (2.2.19b)$$

$$= \sum_{a_t} \pi(a_t|s_t) \sum_{s_{t+1}} \mathcal{P}(s_{t+1}|s_t, a_t) \left[r_t + \gamma \mathbb{E}_\pi[G_{t+1}|s_{t+1}] \right] \quad (2.2.19c)$$

$$= \sum_{a_t} \pi(a_t|s_t) \sum_{s_{t+1}} \mathcal{P}(s_{t+1}|s_t, a_t) [r_t + \gamma V^\pi(s_{t+1})]. \quad (2.2.19d)$$

Similarly, the *Bellman expectation equation for Q^π* is a recursive relationship between the value of a state-action pair (s_t, a_t) and the value of all its possible successor state-action pairs, in the form

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|s_t, a_t] \quad (2.2.20a)$$

$$= \mathbb{E}_\pi[r_t + \gamma G_{t+1}|s_t, a_t] \quad (2.2.20b)$$

$$= \sum_{s_{t+1}} \mathcal{P}(s_{t+1}|s_t, a_t) \left[r_t + \gamma \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1}) \mathbb{E}_\pi[G_{t+1}|s_{t+1}, a_{t+1}] \right] \quad (2.2.20c)$$

$$= \sum_{s_{t+1}} \mathcal{P}(s_{t+1}|s_t, a_t) \left[r_t + \gamma \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1}) Q^\pi(s_{t+1}, a_{t+1}) \right]. \quad (2.2.20d)$$

The Bellman expectation equations for V^π and Q^π are descriptions of the systems to be solved to get V^π and Q^π , respectively. V^π and Q^π represent indeed the unique solutions to their correspondent Bellman expectation equations, and can be even retrieved in closed form under specific assumptions.

Bellman optimality equations

The Bellman expectation equations also hold for the optimal state-value function V^* in Eq. (2.2.17) and the optimal action-value function Q^* in Eq. (2.2.18). However, being V^* and Q^* optimal, their Bellman expectation equations do not depend on any policy, since in the optimal case the best policy is assumed to be followed at any state. In particular, by replacing the first term in Eq. (2.2.19d) with the action maximization performed by the optimal policy, the Bellman expectation equation for V^* – known as *Bellman optimality equation for V^** – takes the form

$$V^*(s_t) = \max_{a_t} \sum_{s_{t+1}} \mathcal{P}(s_{t+1}|s_t, a_t) [r_t + \gamma V^*(s_{t+1})]. \quad (2.2.21)$$

Similarly, by replacing the term averaging the actions according to the policy π in Eq. (2.2.20d) with the action maximization performed by the optimal policy, the *Bellman optimality equation for Q^** is given by

$$Q^*(s_t, a_t) = \sum_{s_{t+1}} \mathcal{P}(s_{t+1}|s_t, a_t) \left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \right]. \quad (2.2.22)$$

In principle, Eq. (2.2.21) and Eq. (2.2.22) can be explicitly solved to get V^* and Q^* , respectively. Notice that the knowledge of Q^* provides a direct way to compute an optimal policy π^* . Any policy selecting the action a_t that maximizes $Q^*(s_t, a_t)$ at s_t is indeed an optimal policy.

In most real cases of interest, however, it is not possible to derive an optimal policy by explicitly solving the Bellman optimality equations. In these cases, Bellman optimality equations can only be solved *approximately*. For high-dimensional state and action spaces, approximations of V^* and Q^* can be maintained as parameterized functions such as deep networks – see Section 2.1 – trained to match the collected experience. Several algorithms have been proposed to this purpose, such as SARSA [Rummery and Niranjan, 1994] or Q-learning [Watkins, 1989].

Deriving policies from value functions, even approximated, is troublesome when dealing with continuous action spaces, as it is often the case for the application of RL to the robotics domain. The selection of the action a_t maximizing $Q^*(s_t, a_t)$ at s_t is indeed problematic for continuous action spaces. In this setting, directly optimizing parameterized policies rather than deriving optimal policies from parameterized value functions is more convenient, as illustrated in the next section.

2.2.3 Policy gradient methods

Policy search methods directly operate with parameterized policies π_θ . Among the large variety of methods classified as policy search [Deisenroth et al., 2013], in this brief overview we will focus on *policy gradient* methods [Sutton et al., 2000], i.e., model-free methods that exploit gradient ascent to optimize the policy parameters θ with respect to a certain performance function $J(\theta)$. Being model-free, these methods make use of sampled trajectories only – and no models of the environment – to optimize the policy. Also known as *likelihood-ratio* policy gradient methods, they update the policy parameters θ such that higher-rewarding trajectories become more likely under the updated policy.

Vanilla policy gradient

Consider a stochastic policy π_θ parameterized by θ and differentiable with respect to its parameter θ , e.g., the Gaussian policy modeled as a deep neural network in Eq. (2.2.9). By following π_θ , trajectories τ in the form of Eq. (2.2.12) can be sampled, along with their associated returns $G(\tau)$.

The probability of sampling a trajectory τ under π_θ – with τ of length T – is given by the product of the probabilities of passing through the T states sequentially visited along τ , i.e.,

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T-1} \mathcal{P}(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t), \quad (2.2.23)$$

where the state-transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$ is unknown.

The *performance function* $J(\theta)$ to be maximized by the agent over the trajectories τ for the reward maximization objective is then defined as

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau)] = \int_{\tau} P(\tau|\theta) G(\tau) d\tau, \quad (2.2.24)$$

and the corresponding maximization problem takes the form

$$\theta^* = \arg \max_{\theta} J(\theta). \quad (2.2.25)$$

This can be solved via *gradient ascent* using the update rule

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta} J(\theta), \quad (2.2.26)$$

where $\alpha \in \mathbb{R}_+$ is the learning rate determining the size of the step in the direction of the gradient. In particular, the *policy gradient* $\nabla_{\theta} J(\theta)$ can be expressed as

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \int_{\tau} P(\tau|\theta) G(\tau) d\tau \quad (2.2.27a)$$

$$= \int_{\tau} \nabla_{\theta} P(\tau|\theta) G(\tau) d\tau \quad (2.2.27b)$$

$$= \int_{\tau} P(\tau|\theta) \nabla_{\theta} \log P(\tau|\theta) G(\tau) d\tau \quad (2.2.27c)$$

$$= \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_{\theta} \log P(\tau|\theta) G(\tau) \right], \quad (2.2.27d)$$

where the passage from Eq. (2.2.27a) to Eq. (2.2.27b) is due to the Leibniz integral rule for differentiation under the integral sign, while the passage from Eq. (2.2.27b) to Eq. (2.2.27c) relies on the *log derivative trick* stating that, given a function $g(x)$, as a direct consequence of the chain rule applied to calculate $\nabla_x \log g(x)$, i.e.,

$$\nabla_x \log g(x) = \frac{1}{g(x)} \nabla_x g(x), \quad (2.2.28)$$

the gradient $\nabla_x g(x)$ can be expressed as

$$\nabla_x g(x) = g(x) \nabla_x \log g(x). \quad (2.2.29)$$

Thanks to the log derivative trick, the policy gradient is therefore expressed as an expectation. The log derivative trick also allows – by exploiting the properties of the logarithm functions – to further expand the term $\nabla_{\theta} \log P(\tau|\theta)$ in Eq. (2.2.27d) as

$$\nabla_{\theta} \log P(\tau|\theta) = \nabla_{\theta} \log \left[\rho_0(s_0) \prod_{t=0}^{T-1} \mathcal{P}(s_{t+1}|s_t, a_t) \pi_{\theta}(a_t|s_t) \right] \quad (2.2.30a)$$

$$= \nabla_{\theta} \left[\log \rho_0(s_0) + \sum_{t=0}^{T-1} \log \mathcal{P}(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t) \right] \quad (2.2.30b)$$

$$= \nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(a_t|s_t) \quad (2.2.30c)$$

$$= \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t), \quad (2.2.30d)$$

where the passage from Eq. (2.2.30b) to Eq. (2.2.30c) removes all the terms that do not depend on θ and are therefore irrelevant for the gradient computation. Finally, by replacing Eq. (2.2.30d) into Eq. (2.2.27d), the policy gradient takes the form

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[G(\tau) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \right], \quad (2.2.31)$$

which can be interpreted as the expectation of the gradient of the policy log-likelihood over τ weighted by the trajectory return $G(\tau)$. This expectation, unlike Eq. (2.2.27d), does not depend on the unknown state-transition probability. However, it still requires the evaluation of all the possible trajectories $\tau \sim \pi_{\theta}$ and cannot therefore be used in practice.

For practical usage, $\nabla_{\theta} J(\theta)$ is then approximated with its empirical average over a finite batch \mathcal{D} of sampled trajectories, i.e., with its *approximate gradient* \hat{g}

$$\hat{g} = \hat{\mathbb{E}}_t \left[G(\tau) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \right] \quad (2.2.32a)$$

$$= \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} G(\tau) \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t). \quad (2.2.32b)$$

With the approximate \hat{g} in place of $\nabla_{\theta} J(\theta)$, the update rule in Eq. (2.2.26) can be directly exploited for the optimization of the policy parameters θ via stochastic gradient ascent, and represents the core of the so-called *vanilla policy gradient* or REINFORCE algorithm [Williams, 1992].

A well-known drawback of vanilla policy gradient methods is that they are extremely sensitive to the selection of the learning rate α – see Eq. (2.2.26). In particular, too small an α results in slow convergence but too large an α leads to diverging updates. However, the instability characterizing the training process of vanilla policy gradient methods in presence of large policy updates is mitigated in several ways by more advanced policy gradients algorithms.

Proximal policy optimization

For more advanced policy gradient methods, alternative expressions of the policy gradient $\nabla_{\theta} J(\theta)$ and its approximate version \hat{g} , with terms weighting the policy log-likelihood other than $G(\tau)$, have been proposed. In its most general form, the policy gradient can be written as

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \Psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right], \quad (2.2.33)$$

where Ψ_t represents a generic log-likelihood weighting term. A common choice for Ψ_t is the advantage function $A_t = A^{\pi_{\theta}}(s_t, a_t)$ - see Eq. (2.2.16). Using the advantage as a weighting term for the policy log-likelihood results indeed in an increased likelihood for actions that perform better than average, and a correspondent decreased likelihood for worse-than-average actions. In this case, the approximate policy gradient \hat{g} takes the form

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \hat{A}_t \right], \quad (2.2.34)$$

where \hat{A}_t is an estimator of the unknown advantage A_t , e.g., the *generalized advantage estimator* (GAE) [Schulman et al., 2016].

The advantage-based \hat{g} is in practice obtained by defining a loss function whose gradient coincides with \hat{g} and then exploiting automatic differentiation. The most common solution for such a loss is the policy gradient (PG) loss function

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_{\theta}(a_t | s_t) \hat{A}_t \right], \quad (2.2.35)$$

while another popular option, derived using importance sampling (IS), is the *surrogate* loss function [Schulman et al., 2015]

$$L^{IS}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right], \quad (2.2.36)$$

where θ_{old} are the pre-update policy parameters and $r_t(\theta)$ is the *likelihood ratio* between the post-update and the pre-update policies.

Inspired by the *natural policy gradient* algorithms [Amari, 1998; Kakade, 2001], the Trust Region Policy Optimization (TRPO) algorithm [Schulman et al., 2015] proposes to optimize the surrogate loss in Eq. (2.2.36) while constraining the policy change from one update to another, i.e., maintaining the policy within a *trust region* during the update. The constraint on the policy update is formulated in terms of the Kullback-Leibler (KL) divergence between the pre-update and post-update policies, and the resulting optimization problem takes the form

$$\max_{\theta} \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right] \quad \text{s.t.} \quad \hat{\mathbb{E}}_t \left[\text{KL}[\pi_{\theta_{\text{old}}}(a | s_t), \pi_{\theta}(a | s_t)] \right] \leq \delta, \quad (2.2.37)$$

where δ is a hyperparameter. By exploiting the conjugate gradient algorithm to solve the above constrained optimization, TRPO guarantees monotonic policy improvement, mitigating therefore the issue characterizing vanilla policy gradient algorithms in presence of large policy updates.

Furthermore, the Proximal Policy Optimization (PPO) algorithm [Schulman et al., 2017] achieves comparable or even better empirical performances than TRPO with a much simpler implementation. This algorithm proposes two variations of the surrogate loss in Eq. (2.2.36) aiming to tackle the issues characterizing vanilla policy gradient algorithms in the presence of large policy updates. The proposed variations of the surrogate loss can be used separately or combined together, leading in both cases to an unconstrained optimization.

In particular, the *clipped* surrogate loss takes the minimum between the usual surrogate loss and a version of the surrogate loss with the likelihood ratio $r_t(\theta)$ clipped in the interval $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a hyperparameter. It is defined as

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right], \quad (2.2.38)$$

where taking the minimum ensures that the clipping has an effect only when the unclipped loss is not beneficial.

On the other hand, the *adaptive KL penalty* surrogate loss is defined as the soft version of the TRPO optimization in Eq. (2.2.37), which makes use of a penalty rather than a constraint and takes therefore the form

$$L^{KL PEN}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t - \beta \text{KL}[\pi_{\theta_{\text{old}}}(a|s_t), \pi_{\theta}(a|s_t)] \right], \quad (2.2.39)$$

where the coefficient β is adapted in such a way to achieve a target KL divergence d^* at each policy update. Specifically, given the estimated KL divergence

$$d = \hat{\mathbb{E}}_t \left[\text{KL}[\pi_{\theta_{\text{old}}}(a|s_t), \pi_{\theta}(a|s_t)] \right], \quad (2.2.40)$$

at each policy update the coefficient β is set as

$$\begin{aligned} \beta &\leftarrow \beta/2, & \text{if } d < d^*/1.5, \\ \beta &\leftarrow \beta \times 2, & \text{if } d > d^* \times 1.5, \end{aligned} \quad (2.2.41)$$

where the hardcoded parameters 2 and 1.5 are chosen heuristically.

Finally, the clipped surrogate loss and the adaptive KL penalty loss, when used together, take the form

$$\begin{aligned} L^{CLIP+KL PEN}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right. \\ \left. - \beta \text{KL}[\pi_{\theta_{\text{old}}}(a|s_t), \pi_{\theta}(a|s_t)] \right]. \end{aligned} \quad (2.2.42)$$

Chapter 3

State of the Art and Thesis Context

This chapter presents the state of the art in the fields of Bipedal Locomotion and Character Animation relevant for this thesis, contextualized at the end of the chapter. More in detail, the chapter is organized as follows.

The state of the art in Bipedal Locomotion is illustrated in Section 3.1, focusing on the three subsequent layers of a hierarchical control architecture. Methods employed in the outer trajectory optimization layer are described in Section 3.1.1. Solutions for the central simplified model control layer are discussed in Section 3.1.2. Techniques adopted in the inner whole-body control layer are introduced in Section 3.1.3. In addition, Section 3.1.4 covers alternative approaches to bipedal locomotion based on reinforcement learning.

The state of the art in Character Animation is illustrated in Section 3.2. Kinematic motion synthesis techniques aiming at realistic character animation without necessarily satisfying physics constraints are covered in Section 3.2.1. Physics-based motion synthesis methods operating instead on dynamically-simulated characters are covered in Section 3.2.2.

Finally, the thesis contributions are contextualized with respect to the limitations and criticalities of the state of the art in Section 3.3, while the robotic platforms used for their experimental validation are described in Section 3.4.

3.1 State of the art in Bipedal Locomotion

Designed to mimic human shape and behaviour, humanoid robots are characterized by an articulated mechanical structure with many degrees of freedom. They are supposed to interact with challenging environments to pursue a large variety of tasks. Given their ambitious multi-purpose objective, such robotic platforms come with a series of intrinsic and distinctive complexities.

Bipeds, compared to other morphologies such as quadrupeds [Poulakakis et al., 2005], are inherently unstable. Their sparse mass distribution and narrow support surface, defined by relatively small and close feet in contact with the ground, require dynamic balancing for effective locomotion. Being underactuated [Spong, 1998], humanoids cannot affect their global pose without exploiting contacts with the external

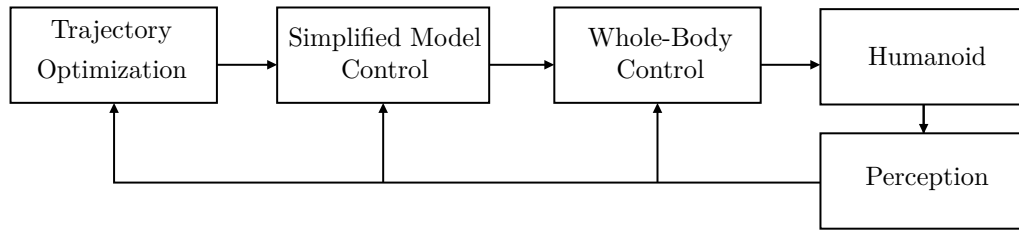


Figure 3.1. Three-layer hierarchical control architecture commonly adopted for humanoid locomotion. Each layer provides inputs to the subsequent one, down to the actual references for the humanoid. Perception is exploited to close the loop at different levels.

environment. Handling contacts is already non-trivial in structured environments, where unlike wheeled robots [Borst et al., 2009] humanoids are expected to fully exploit their legged structure and resemble human motion capabilities, which could also mean climbing stairs or jumping over holes. Unstructured environments, due to their unpredictability, make this interaction even more tricky. In addition, humanoids are redundant [Wieber et al., 2016; Siciliano et al., 2008] with respect to the locomotion task, typically involving their lower body while leaving free their upper body motion. All these issues and many others, e.g., the low efficiency of the robot sensor and actuation system, make bipedal locomotion an extremely challenging research area within the broader field of humanoid robotics.

A recent approach to tackle bipedal locomotion comes from the DARPA Robotics Challenge, and consists in breaking down the complexity of the problem by means of a hierarchical control architecture composed of several layers [Feng et al., 2015]. The three main layers of this architecture, commonly adopted in the bipedal locomotion literature [Carpentier et al., 2017; Romualdi et al., 2018, 2020], are referred to as *trajectory optimization*, *simplified model control* and *whole-body control* layers – see Figure 3.1. Each layer generates references for the subsequent layer by processing inputs from the robot and the environment. Inner layers usually exploit more complex models to compute their output, on a shorter time horizon to satisfy computational constraints. If external disturbances act on the robot, all the layers are involved in adjusting the trajectory and letting the robot keep walking without falling.

3.1.1 Trajectory optimization layer

Given the specification of some walking target, the *trajectory optimization* layer is in charge of computing the sequence of robot footsteps to proceed towards the target, i.e., the locations and timings of the contacts that the robot is going to make with the ground. Depending on the assumptions on the robot model and the environment made at this stage, the complexity of the footsteps generation varies considerably. For instance, assuming flat terrain and modeling the robot as a simple unicycle [Truong et al., 2010; Morin and Samson, 2008] fast solutions to the walking pattern generation problem can be retrieved [Dafarra et al., 2018]. However, too simple models impose constraints on the footstep sequence, compromising therefore the execution of highly dynamic motions which often require unconstrained footsteps placement. On the other hand, several planners considering the centroidal dynamics and the full robot kinematics have proved successful in different scenarios [Dai et al., 2014; Herzog et al., 2015; Fernbach et al., 2018; Dafarra et al., 2020].

Certain solutions to the trajectory optimization problem also allow for the integration of reactive strategies [Shafiee et al., 2019; Bombile and Billard, 2017]. When the robot undergoes the effect of external disturbances, these methods adjust the contacts accordingly so that the robot maintains its balance. Different strategies are available for push-recovery, from ankle-based to hip-based as well as stepping strategies [Stephens, 2007], often truly effective only when combined together [McGreavy et al., 2020; Jeong et al., 2019]. However, due to the hand-crafted models required by each of these strategies, ensuring robust transitions from one to another turns out to be particularly challenging and requires careful tuning.

Based on the amount of information on the contacts they assume to be provided, the methods for trajectory optimization can be classified in the four main categories illustrated below from the most to the least dependent on contact information.

Predefined contact locations and timings. Methods assuming contact locations and timings provided by an external contact planner aim at generating the remaining quantities, i.e., centroidal quantities and possibly body postures. To solve the problem online, either the CoM dynamics alone is considered [Caron and Kheddar, 2016] or a convex relaxation of the angular momentum dynamics to be minimized [Ponton et al., 2016]. Given the footsteps, online whole-body posture generators exploit Differential Dynamic Programming (DDP) [Budhiraja et al., 2018; Giraud-Esclasse et al., 2020], even in combination with model predictive control [Dantec et al., 2021; Mastalli et al., 2022]. The assumption of having predefined contact locations and timings plays a major role in enabling online solution for these planners.

Predefined contact sequence. Methods assuming only the contact sequence being predefined [Carpentier et al., 2016; Caron and Pham, 2017; Winkler et al., 2018] aim at planning the contact locations and timings. This is often the case for bipedal locomotion, since a biped robot can be often assumed to produce a periodic alternate contact sequence, in which a contact with the right foot is followed by another one with left foot, and so on. This simplification, however, does not reduce the computational effort enough to enable online planning.

Mixed-integer methods. These methods model contact activations through integer variables that determine where to establish contacts [Deits and Tedrake, 2014; Mirjalili et al., 2018; Mason et al., 2018] and in which time instant [Ibanez et al., 2014; Aceituno-Cabezas et al., 2018], making use of mixed integer programming tools. Although enhancing modeling capabilities, the exploitation of integer variables characterizing these methods strongly affects the computational performances, especially in the case of multiple contacts and a subsequent high number of associated integer variables to handle. Therefore, mixed-integer methods cannot be used online.

Complementarity-free methods. These methods model contacts explicitly, letting the planner decide sequence, locations and timings of the footsteps. Using such a complete modeling, with no a priori information about the contacts, these methods allow to generate remarkably complex motions [Dafarra et al., 2020, 2022b]. However, their computational time exceeds the time horizon for which they look for a feasible solution, preventing therefore their usage online.

3.1.2 Simplified model control layer

Given the footsteps sequence, timings and locations from the trajectory optimization layer, the *simplified model control* layer is responsible for finding feasible trajectories for the robot center of mass (CoM). The computational burden arising from the search of feasibility regions for the CoM is usually balanced by the adoption of simplified models to characterize the robot dynamics.

Assuming constant CoM height and angular momentum, it is possible to exploit the well-known Linear Inverted Pendulum (LIP) model [Kajita et al., 2001] – see Section 1.3.1 – to design simple and effective controllers. The decomposition of the LIP dynamics into a stable and an unstable component [Pratt et al., 2006; Hof, 2008; Takenaka et al., 2009; Engelsberger et al., 2011; Koolen et al., 2012; Pratt et al., 2012] leads to walking stabilization strategies built around the unstable component, denoted by a series of different names ranging from the Extrapolated Center of Mass (xCoM) [Hof, 2008] to the instantaneous Capture Point (CP) [Pratt et al., 2012; Koolen et al., 2012] and the Divergent Component of Motion (DCM) [Takenaka et al., 2009]. Initially formulated in 2D, the DCM has been extended to the 3D case as well [Engelsberger et al., 2013, 2015] – see Section 1.3.3.

The aforementioned simplified models gained popularity also in combination with the Zero Moment Point (ZMP) [Vukobratović and Borovac, 2004] – see Section 1.3.2 – as a walking stability criterion [Caron and Kheddar, 2017], also extended to non-coplanar contacts [Caron et al., 2017]. Nowadays, these models are widely adopted for both position-controlled robots [Shafiee-Ashtiani et al., 2017b; Kamioka et al., 2018; Leng et al., 2020; Ramuzat et al., 2021] and torque-controlled robots [Stephens and Atkeson, 2010; Pratt et al., 2012; Dafarra et al., 2016; Griffin and Leonessa, 2016; Engelsberger et al., 2018a,b].

The main advantage of both the LIP and DCM models is that of being linear models, as a result of the assumption of constant CoM height and angular momentum. The DCM model has been demonstrated to remain linear even in the case of varying CoM height [Engelsberger et al., 2013], although a time-varying pendulum natural frequency needs to be considered for generic CoM height trajectories [Hopkins et al., 2014]. The hypothesis of constant CoM height is also relaxed in [Koolen et al., 2016], where an extension of the LIP model named variable-height inverted pendulum (VHIP) is introduced.

By taking advantage of these simplified models, various successful instantaneous [Hopkins et al., 2014; Engelsberger et al., 2015, 2018a,b] and predictive [Wieber, 2006; Diedam et al., 2008; Griffin and Leonessa, 2016; Bombile and Billard, 2017; Dafarra et al., 2018] controllers have been designed, also providing references for the footstep locations [Joe and Oh, 2018; Shafiee-Ashtiani et al., 2017a] and timings [Griffin et al., 2017; Khadiv et al., 2016]. Finally, model predictive controllers using the LIP model have been shown to produce CoM trajectories guaranteed to be stable [Scianca et al., 2016, 2020; De Simone et al., 2017; Smaldone et al., 2019, 2020, 2021, 2022].

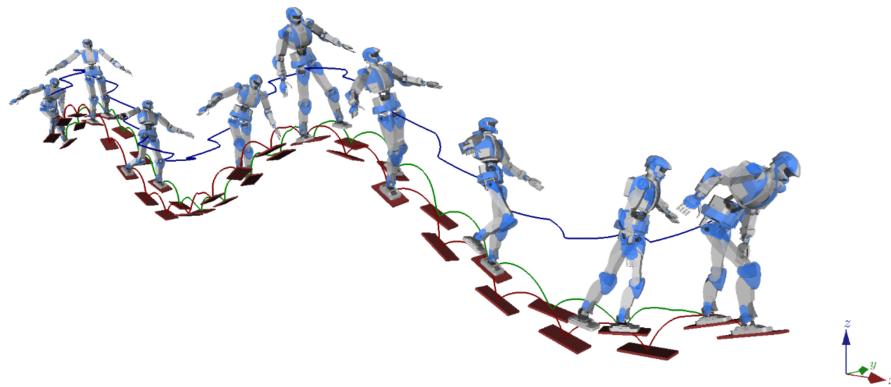


Figure 3.2. Extremely complex example of walking pattern generation for humanoid locomotion over uneven terrain. The CoM and the swing foot trajectories are highlighted in color. Image taken from [Caron and Pham, 2017].

3.1.3 Whole-body control layer

Given the reference trajectories from the simplified model control layer, the whole-body control layer is in charge of ensuring their tracking by generating the references that are directly sent to the robot – see Figure 3.1. Depending on the available robot control modes, these references can be in the form of joint positions, velocities or torques. With respect to position-controlled robots, torque-controlled robots are intrinsically compliant under the action of external disturbances [Fahmi et al., 2019; Mesesan et al., 2019], allowing for safe interactions with the environment [Ott et al., 2011; Saab et al., 2013] as well as with human collaborators [Romano et al., 2018; Tirupachuri et al., 2020].

Within the hierarchical architecture depicted in Figure 3.1, the whole-body control layer is the layer running at higher frequency. In this layer, instantaneous algorithms on whole-body robot models are employed rather than advanced techniques such as model predictive control on simplified models. To meet the computational constraints imposed by the high frequency, the whole-body optimization problems solved in this layer are often framed as Quadratic Programming (QP) problems. The layer itself is therefore often defined as whole-body QP control layer.

Being humanoids redundant with respect to the locomotion task, whole-body controllers adopted in this layer are often organized in a hierarchical *stack-of-tasks* formulation, with strict or weighted hierarchies. Strict hierarchies aim to solve each task in the null-space of its higher priority tasks [Park and Khatib, 2006; Wensing and Orin, 2013; Nava et al., 2016; Pucci et al., 2016; Padois et al., 2017]. As a result, tasks do not interfere with each other. Weighted hierarchies incorporate all the tasks in a cost function, whose terms are weighted to encode priorities [Lee and Goswami, 2012; Bouyarmane and Kheddar, 2018; Schuller et al., 2021]. With this approach, all the tasks compete in the search for the solution according to their associated weight. Particular care is needed in the definition of a well-posed cost function and tuning of the task weights, for which also automatic procedures have been proposed [Modugno et al., 2016a,b]. In practice, it is common to combine strict hierarchies for high-priority tasks with weighted hierarchies for low-priority tasks, addressed by using the redundant degrees of freedom with respect to the high-priority tasks.

3.1.4 Reinforcement learning based approaches

Alternative approaches to bipedal locomotion aim at sidestepping the computational complexities of the hierarchical control architecture presented in the previous sections by leveraging learning-based techniques, mainly RL methods – see Section 2.2. The application of RL algorithms to the control of simulated robots has been proven capable of providing model-free end-to-end successful policies for a wide variety of high-dimensional tasks, including legged locomotion, in [Lillicrap et al., 2016]. Afterwards, extensive research in the field has led to outstanding results, especially for what concerns quadrupedal locomotion [Hwangbo et al., 2019; Bin Peng et al., 2020; Lee et al., 2020; Yang et al., 2020b]. Recently, a RL-based controller enabled a quadruped robot to successfully complete a 1-hour-long hike in the Alps in the time recommended for human hikers [Miki et al., 2022].

RL-based bipedal locomotion needs further development to achieve performances comparable to RL-based quadrupedal locomotion. However, a series of RL-based solutions for bipedal locomotion can be found in the literature demonstrating promising results for both simulated and real bipeds.

As regards simulated bipeds, different balancing and push-recovery strategies have been shown to emerge from a single policy controlling the robot lower body joints if the training process is guided by a carefully-shaped locomotion-oriented reward [Yang et al., 2017, 2018]. A similar approach, additionally leveraging motion capture data as postural regularizers in the reward function and a peculiar neural network architecture to represent the policy, extends to robust locomotion in simulation [Yang et al., 2020a]. By incorporating model information in the RL setting, feasible walking gaits for simulated bipeds are obtained with no need for reference trajectories in the form of motion capture data [Castillo et al., 2020; Ordonez-Apraiez et al., 2022]. Given an external contact planner, feeding a policy with the two upcoming footsteps has been shown sufficient for achieving omnidirectional walking, turning in place and even climbing stairs in simulation, with no need for motion capture references but an appropriate curriculum learning strategy to progress toward tasks with an increasing level of complexity [Singh et al., 2022].

For what concerns RL policies deployed on real bipeds, [Xie et al., 2019] propose to combine imitation and reinforcement learning to retrieve locomotion policies resembling deterministic feasible motions produced by existing controllers. An extensive usage of domain randomization during training turns out to be pivotal to obtain robust dynamic behaviours on a real biped capable of recovering from significant external perturbations [Li et al., 2021]. Thanks to domain randomization techniques compensating for the differences between the simulated and the physical system, also a policy learning target dynamical motions in the form of motion capture data seamlessly transitions from training in simulation to executing on a reduced-height physical humanoid [Taylor et al., 2021]. A methodology based on curriculum learning that gradually increases the target velocity is used to train a single control policy for omnidirectional walking on a lightweight adult-size humanoid [Rodriguez and Behnke, 2021]. No curriculum learning nor domain randomization are instead employed for a cascade architecture achieving sustained walking gaits under external disturbances on a real humanoid, capable of adapting to challenging terrains not included in the training process [Castillo et al., 2021]. Recently, [Bloesch et al., 2022]

introduced an end-to-end procedure to train a locomotion policy for a small bipedal robot directly on hardware, with minimal human intervention, paving the way for alternative strategies for real-robot learning with respect to the challenging transfer of policies trained in simulation.

Finally, learning-based techniques other than RL – specifically, Gaussian processes – also allowed for an efficient synthesis of highly dynamical walking patterns for humanoid robots demonstrated on real platforms [Clever et al., 2017]. In this work, feasible movement primitives are learned from an offline-generated dataset of reference trajectories obtained by solving optimal control problems which also take into account the robot’s dynamical model.

3.2 State of the art in Character Animation

Synthesizing natural-looking motions for animated characters is a long-standing challenge in computer graphics. Besides having a realistic appearance, animations are often required to interactively follow user-specified goals or input signals driving their generation. Responsive realistic character animation has been tackled through a wide variety of methods that can be primarily divided into two streams, *kinematic* and *physics-based*, depending on whether the animation obeys some physics constraints. As opposed to robotics research aimed at developing methods eventually meant for real-world platforms, character animation is indeed intended for virtual-world deployment. Therefore, obtaining dynamically-consistent motions can be convenient, but it is not strictly required.

Both kinematic and physics-based methods can be further characterized according to their nature and target. Given a dataset of motion clips, parametric models trained on it can be used later to synthesize *novel* motions. On the contrary, non-parametric methods only look for effective ways of *re-playing* the given data. Methods for generating task-specific motions – even new ones – *close* to the reference data usually rely on supervised learning only. More general techniques that first learn a generic motion model from the data, regardless of the motion generation target, and then adapt it to even *remarkably different* tasks typically take advantage of reinforcement learning. The taxonomy of the literature overview provided in this section is summarized in Figure 3.3.

3.2.1 Kinematic motion synthesis

Kinematic motion synthesis methods animate characters with no regards for physics constraints. Since they do not operate in dynamically-simulated environments, to avoid designing the animation from scratch they typically exploit reference motion capture data and are therefore also known as *data-driven* methods – although physics-based methods taking advantage of reference motion capture data exist as well. The large variety of tools used for data-driven animation synthesis, including graphs, linear methods, kernel methods, and most recently neural networks, are briefly introduced by category in the following.

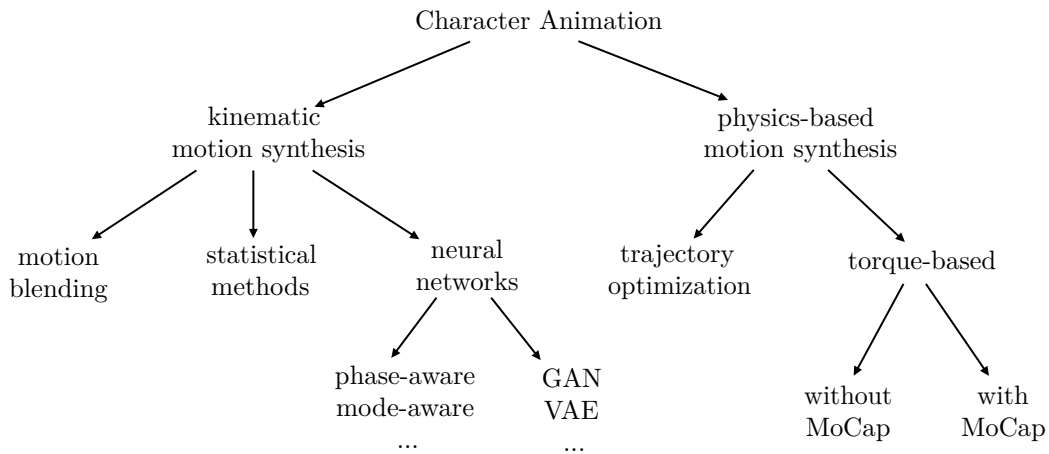


Figure 3.3. Taxonomy of the Character Animation literature introduced in this section. The two main streams of kinematic and physics-based motion synthesis are further categorized in subfields. Notice that often the distinction between branches is not sharp, and there exist methods contributing to more than one field.

Non-parametric motion blending methods. Given a dataset of motion clips, motion blending approaches attempt to concatenate them into a coherent task-oriented trajectory [Agrawal and van de Panne, 2016]. Blending enables seamless transitions between clips, that are directly played back from the original dataset by following some control logic often based on finite state machines. Data structures such as *motion graphs* [Arikan and Forsyth, 2002; Kovar et al., 2002; Lee et al., 2002; Beaudoin et al., 2008] have been alternatively proposed to model transitions between arbitrary clips. In this case, a pose distance metric characterizing clip transitions is used to build the graph. Then, coherent motion sequences are extracted by searching the graph. However, the discrete nature of graphs allows for transitions only at the end of a motion segment. Continuous representations such as *motion fields* [Lee et al., 2010b] cope with this issue by moving the blending at the individual frames level, i.e., in the high-dimensional field of character poses. *Motion Matching* [Clavet, 2016] follows the idea of motion fields to synthesize locomotion behaviours by searching for the best possible successor frame over the entire dataset of motion clips, given the current frame and a user-specified input. In particular, each motion generation step involves a nearest-neighbour search on feature vectors embedding the locomotion task – in the form of desired future root trajectory – to retrieve the best-matching frame to play back. Besides providing an extremely powerful and efficient solution for kinematic motion synthesis, this method was also employed as a lightweight kinematic controller to guide physics-animated systems [Bergamin et al., 2019; Hong et al., 2019], proving flexible and highly responsive. Its main drawback, i.e., the large memory usage required to store the original dataset of motion clips and its associated dataset of matching features, was drastically reduced by its learning-based extension [Holden et al., 2020]. However, as all the motion-blending-related methods, Motion Matching has by design no generalization capabilities outside the original dataset of motions, carefully selected and played back in a realistic sequence.

Statistical models. To develop animation systems that synthesize novel motions rather than simply playing back the existing ones, researchers have looked towards statistical methods and early machine learning approaches. Maximum a Posteriori (MAP) frameworks use a motion prior to regularize user-specified motion constraints [Chai and Hodgins, 2007; Min et al., 2009]. Linear methods such as local Principal Component Analysis (PCA) synthesize animation from low dimensional signals [Chai and Hodgins, 2005; Tautges et al., 2011]. Kernel-based approaches such as Radial Basis Functions (RBF) and Gaussian Processes (GP) overcome the limitations of linear character animators and consider non-linearities in the data [Mukai and Kuriyama, 2005; Wang et al., 2008; Mukai, 2011]. A generative model combining functional PCA to learn within each motion class and GP for the transitions between classes is proposed in [Min and Chai, 2012]. Gaussian Process Latent Variable Models (GPLVM) [Grochow et al., 2004; Levine et al., 2012] learn an underlying low-dimensional motion latent space and then animate characters in the latent space at runtime to meet the user-specified motion constraints, leading to better generalization performances. All these methods, however, suffer from scalability issues, both for their computational and memory cost.

Neural-network-based generative models. Conversely, neural-network-based generative models are able to scale with increasing data and learn from very large motion capture datasets while maintaining low memory and computational costs at runtime. In this case, the consistency of the generated motion is maintained by making use of *recurrent* or *autoregressive* models, both suitable for learning time-series data. The former keep an internal memory state, while the latter predict the next character pose using the previously-predicted pose blended with the user input. Conditional Restricted Boltzmann Machines (cRBM) were originally used to predict subsequent poses during locomotion [Taylor and Hinton, 2009]. Recurrent Neural Networks (RNN) have been demonstrated to successfully operate on complex human characters [Harvey and Pal, 2018] and effectively learn locomotion and basketball motions [Lee et al., 2018b]. A Long Short-Term Memory (LSTM) network is used to robustly synthesize high-quality motions between temporally-sparse animation keyframes [Harvey et al., 2020]. For offline motion synthesis, Convolutional Neural Networks (CNN) along the time domain [Holden et al., 2015, 2016] have been used to learn a latent motion manifold from task-relevant data, to which high-level user commands are mapped to drive the animation synthesis. Nevertheless, RNNs and other NN-based generative models are known to suffer from converging to an average pose – the so-called *dying-out* effect – when generating long motion sequences [Fragkiadaki et al., 2015].

Phase-aware and mode-aware NN-based models. In order to overcome over-smoothed motions or convergence to an average pose observed in deep-network-based generative models [Fragkiadaki et al., 2015], *phase-aware* and *mode-aware* neural networks have been proposed. Phase-Functioned Neural Networks (PFNN) [Holden et al., 2017] are phase-weighted mixtures of neural networks trained on bipedal locomotion data. At prediction time, the weights of PFNN are blended according to a cyclic phase function encoding the periodicity of the walking motion. The character animation system for PFNN is shown in Figure 3.4 (bottom left). This

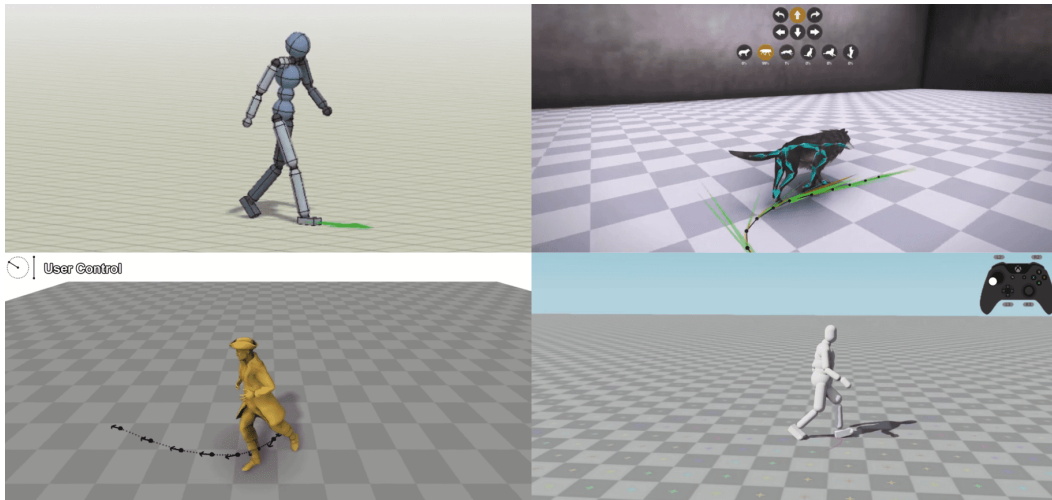


Figure 3.4. Examples of character animation systems. From top-left, clockwise: the DeepMimic character [Peng et al., 2018], the MANN quadruped [Zhang et al., 2018], the DReCon character [Bergamin et al., 2019], and the PFNN character [Holden et al., 2017]. Each system shows the associated user control interface.

resulted in a significant breakthrough for character animation, enabling remarkably natural motion and smooth transitions between arbitrary frames in different types of variable-terrain locomotion. However, training data need to be annotated with phase function values, which can be costly or unfeasible for the complex non-periodic motions that may arise in realistic unconstrained humanoid locomotion scenarios. [Starke et al., 2020] propose to automatically extract separate local motion phases for each character limb, enabling this approach for asynchronous motions that cannot be associated to a global phase. In Mode-Adaptive Neural Networks (MANN) [Zhang et al., 2018], the phase-related issues are solved by substituting the phase function with a gating network, which learns end-to-end how to effectively blend the network weights, and produces successful quadruped locomotion patterns on complex terrains. The quadruped animated using MANN is shown in Figure 3.4 (top right). This work builds upon the Mixture of Experts (MoE) paradigm [Nowlan and Hinton, 1990; Jacobs et al., 1991; Jordan and Jacobs, 1994], a classic machine learning approach with a series of extensions [Yuksel et al., 2012] where several experts specialize in problem subdomains assigned by the gating network as the training progresses. A similar approach has been extended to tasks involving interaction with objects and the environment such as carrying objects, sitting and opening doors [Starke et al., 2019], and enhanced towards a modular framework enabling iterative addition of control modules for different tasks with no need for retraining from scratch [Starke et al., 2021]. Besides the long training time shared by all the deep-learning-based methods, one drawback of phase-aware and mode-aware architectures – shared with Motion Matching – is the careful tuning required to properly handle user inputs at runtime. For locomotion and interaction tasks, this has been done by adding future trajectory data [Holden et al., 2017; Zhang et al., 2018] and object representations [Starke et al., 2019] to the network input features, respectively, but how to constrain more general motions remains an open problem.

Alternative NN-based models. Alternative solutions to the issues of standard deep-network-based generative models exist in the literature other than phase-aware and mode-aware networks. Generative Adversarial Networks (GANs) [Goodfellow et al., 2014] involve a motion generator trained in an adversarial setting to fool another network, the discriminator, trying to differentiate real motions from generated ones. Generative Adversarial Imitation Learning (GAIL) [Ho and Ermon, 2016] combines GANs with RL to address the challenge of data-driven reward design for imitation learning. GAN-based methods for physics-based motion synthesis are also proposed in [Peng et al., 2021]. Similarly, Variational Autoencoders (VAE) have been employed to model the motion manifold [Habibie et al., 2017; Ling et al., 2020], with a RL policy controlling the input of the VAE during interactive motion synthesis. Finally, Transformers [Vaswani et al., 2017] have been exploited for goal-conditioned high-quality motion synthesis from large unstructured datasets [Li et al., 2020].

3.2.2 Physics-based motion synthesis

Physics-based motion synthesis methods make use of physics simulations to generate motions [Geijtenbeek and Pronost, 2012]. The motion dynamics and the action space – either joint torques or muscles – are defined by the simulation, and the challenge is to design or learn controllers for the dynamically-simulated characters that, when successful, will implicitly produce motions with physical realism.

Similarly to bipedal locomotion in robotics – see Section 3.1 – physics-based motion synthesis can be addressed via *trajectory optimization* methods, often relying on simplified models, or learning-based methods, also known as *torque-based* approaches in reference to their direct decision-making on torque references with no intermediate optimization stages. Both trajectory-based and torque-based methods can take advantage of motion capture data, as regularizers towards natural-looking behaviours.

Trajectory optimization methods. A classical yet enduring approach to physics-based motion synthesis has been to manually design controllers using finite state machines and heuristic feedback rules [Yin et al., 2007; Coros et al., 2009, 2010; Lee et al., 2010a]. Despite leading to results for specific task-oriented applications, handcrafted controllers require deep domain knowledge and remain hard to generalize to different tasks. Trajectory optimization approaches make use of optimization techniques to enforce motion feasibility [Wampler and Popović, 2009; Levine and Popović, 2012; Wampler et al., 2014], typically by resorting to simplified models such as centroidal dynamics for computational efficiency. As for the control of humanoid robots, Quadratic Programming (QP) optimization [Da Silva et al., 2008; Macchietto et al., 2009] as well as Model Predictive Control (MPC) methods [Hong et al., 2019] have been used for physics-based motion synthesis. Complex multi-contact interactions between the character and the environment have been tackled by learning control graphs [Liu et al., 2016] or scheduling short motion fragments [Liu et al., 2010] with deep Q-learning, resulting into highly dynamic behaviours such as walking on a ball or skateboarding [Liu and Hodgins, 2017] as well as playing basketball [Liu and Hodgins, 2018]. Similar results are extended for muscle-actuated characters [Lee et al., 2019]. A common drawback of trajectory-based approaches is, however, that they are typically not suitable for real-time operation.

Torque-based methods, without motion capture data. Torque-based approaches overcome the limitations of trajectory-based optimization by taking advantage of model-free deep reinforcement learning. The core issue is then shifted to the definition of an appropriate reward function for the desired animation. Tracking-free methods do not exploit reference motion capture data for the definition of the reward. Their reward is simple and goal-oriented, e.g., proceed forward or maintain a certain speed, with marginal concern on the emerging behaviour. Robust locomotion performances across several tasks from such simple rewards were firstly shown to emerge in [Heess et al., 2017], encouraged by the interaction of the character with rich environments. Deep RL with simple goal-oriented rewards was also successfully applied to quadrupeds [Peng et al., 2015] and bipeds [Peng et al., 2016] control in 2D environments. In the absence of motion regularizers, the behaviours obtained by tracking-free methods show however asymmetric gaits and further artifacts which confer them low realism.

Torque-based methods, with motion capture data. To obtain more realistic motions, torque-based approaches using deep RL learn, among the various objectives, also to imitate reference expert motions. An imitation term is added to the reward function in DeepLoco [Peng et al., 2017] to implement walking style imitation in navigation tasks on a simplified armless biped. Imitation and goal-conditioned learning are also combined in DeepMimic [Peng et al., 2018], where the exploitation of short sequences of motion capture data as references leads to a remarkably large variety of natural-looking dynamically-consistent motion skills. The DeepMimic animated character is depicted in Figure 3.4 (top left). [Chentanez et al., 2018] build upon DeepMimic to train a recovery policy to be used if the character deviates significantly from the reference motion. Moreover, [Won et al., 2020; Wang et al., 2022] overcome the need for retraining at each new motion to be learned, that took hours or even days in the case of DeepMimic. Learning a family of motor skills from a single motion clip was proposed in [Lee et al., 2021; Li et al., 2022], while longer motions clips dynamically selected according to the desired motion speed, direction and style were adopted in [Bergamin et al., 2019], whose animated character is shown in Figure 3.4 (bottom right). Recently, tracking-based methods have further demonstrated their effectiveness and generality when combined with VAEs [Yin et al., 2021] or GANs [Xu and Karamouzas, 2021; Peng et al., 2022]. It is worth to mention, though, that careful reward design plays a fundamental role in all the works combining imitation learning with goal-directed learning via deep RL.

3.3 Thesis context

Given the concise overview on the state-of-the-art in Bipedal Locomotion (Section 3.1) and Character Animation (Section 3.2) provided in the former part of this chapter, we now detail the context of the thesis contributions which fall, to a large extent, at the intersection of the two aforementioned domains. The thesis contributions will be further elaborated in the three chapters of Part II, whose content is summarized in the following.

Chapter 4: Learning Whole-Body Push-Recovery Strategies

This chapter presents the design of a control policy for balancing and push-recovery of humanoid robots trained by model-free deep Reinforcement Learning (RL). As in [Yang et al., 2018], we develop a low-frequency high-level policy that, rather than directly generating joint torques, assumes the presence of low-level PID controllers for the joints. Differently from [Yang et al., 2018], where actions are defined as joint position targets, our policy commands *joint velocities*, subsequently integrated to get the references for the PID controllers. This is to guarantee continuous inputs for the low-level controllers even in the event of discontinuous actions selected by the policy. Moreover, as opposed to [Kim et al., 2019; Yang et al., 2018, 2020a] which investigate RL-based push recovery by controlling the humanoid lower body joints only, our method targets *high-dimensional whole-body* humanoid control. Our policy controls therefore all the humanoid joints but those of the hands, wrists and neck, which arguably play a minor role in balancing.

Compared to [Peng et al., 2017, 2018; Bergamin et al., 2019], no expert knowledge in the form of reference motion capture data is used in our approach to shape the agent behaviour. We investigate indeed the *emergence* of whole-body push-recovery strategies exclusively through autonomous agent exploration. Inspired by [Yang et al., 2018], we encourage successful exploration for the agent by defining reward components that incorporate domain knowledge in humanoid control to promote *transient* push-recovery strategies as well as *steady-state* balancing in the absence of external disturbances. We validate the proposed approach on the simulated iCub v2.7 humanoid, showing that our policy can withstand repeated applications of strong out-of-sample external pushes through a combination of ankle, step and momentum-based strategies (enabled by the control of the humanoid upper body), proving therefore robust and characterized by remarkable generalization capabilities.

Chapter 5: Learning Human-Like Whole-Body Trajectory Generators

In this chapter, we propose a data-driven whole-body trajectory generator of locomotion patterns for humanoid robots trained by deep Supervised Learning on motion capture data. Motivated by the responsive and natural-looking behaviours obtained by Mode-Adaptive Neural Networks (MANN) [Zhang et al., 2018] in the context of kinematic animation of quadruped characters, we adopt such a model for learning human-like locomotion trajectories for humanoid robots. Similarly to [Zhang et al., 2018; Bergamin et al., 2019], we collect an *unstructured* motion capture dataset that includes various locomotion modes and transitions between them. To retarget our motion capture dataset onto the given robot model, extending [Darvish et al., 2019], we propose a geometric approach that guarantees *kinematic feasibility* of the robot base motion.

Given the retargeted dataset, we train our MANN implementation on input and output features inherited from [Clavet, 2016; Holden et al., 2017; Zhang et al., 2018] that, being suitable for blending with a user-specified input, enable the exploitation of the selected autoregressive model for *interactive* trajectory generation. We validate the proposed approach on the iCub v2.7 humanoid model, demonstrating how our learning-based trajectory generator can *efficiently* provide kinematically-feasible

whole-body trajectories for the robot spanning a wide range of walking patterns and smooth transitions among them, while also exhibiting a certain degree of *human-likeness* inherited from the human-retargeted training data.

Chapter 6: Human-Like Whole-Body Control of Humanoid Robots

We present here the end-to-end system architecture, named ADHERENT, obtained by integrating the learning-based whole-body trajectory generator introduced in Chapter 5 with a state-of-the-art control architecture for humanoid locomotion [Romualdi et al., 2018, 2020] characterized by the three-layer hierarchical structure depicted in Figure 3.1. At the *trajectory optimization* layer, differently from [Fargasso et al., 2013; Cagnetti et al., 2016; Dafarra et al., 2018] which rely on a simple unicycle model to generate footsteps for the humanoid, we leverage the MANN-generated whole-body trajectories to extract contact locations and timings. This enables unconstrained footsteps placement from minimal references as in [Dafarra et al., 2020, 2022b], compared to which our approach only ensures kinematic feasibility and not dynamical consistency, but overcomes the computational complexity that prevents an online usage.

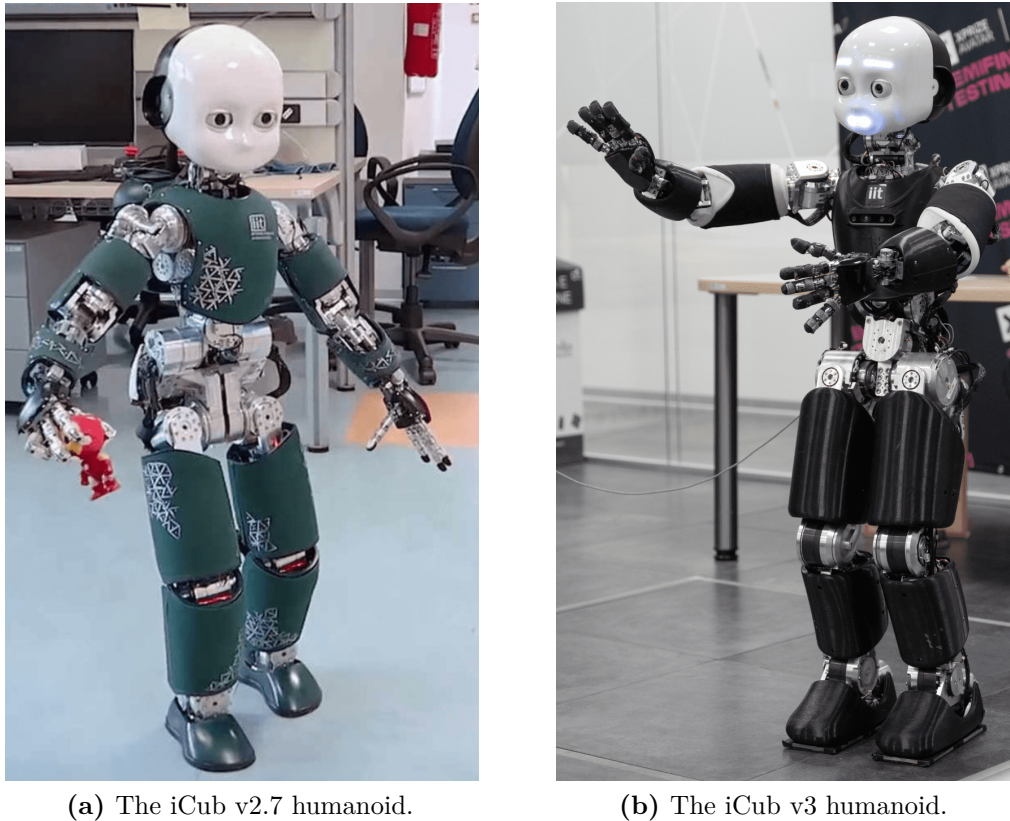
At the *whole-body QP control* layer, as opposed to [Righetti et al., 2011; Nava et al., 2016; Romualdi et al., 2018, 2020] that adopt a constant reference configuration for the low-priority postural task, we retrieve a *time-varying* whole-body postural from the MANN-generated trajectories. Such a postural, besides being coherent with the footstep plan, acts as a regularizer towards *human-like* walking motions. We validate the proposed ADHERENT architecture in terms of robustness and human-likeness with experiments on both the iCub v2.7 and the iCub v3 humanoids, showing therefore how our method easily adapts to different robots. Additionally, we investigate the modularity of ADHERENT with respect to the addition of new motion skills such as crouching.

3.4 The iCub humanoid robot

The iCub humanoid robot is an open-source state-of-the-art robotic platform developed at the Italian Institute of Technology, originally conceived for research in embodied artificial intelligence as part of the RobotCub European Project [Metta et al., 2010; Parmiggiani et al., 2012; Natale et al., 2017]. Since the first release of the iCub in 2006, more than 40 copies of this robot have been distributed worldwide, each with its own features depending on the release date, maintenance upgrades, and custom modifications, which identify the iCub version¹.

We present here the two specific versions of the iCub humanoid that have been used as validation platforms for the methods developed in this thesis, namely the iCub v2.7 and the recently-presented iCub v3 [Dafarra et al., 2022a], both depicted in Figure 3.5. Section 3.4.1 presents the iCub v2.7 humanoid and Section 3.4.2 illustrates the iCub v3 humanoid, while the software infrastructure employed to control and simulate all the iCub robots is introduced in Section 3.4.3.

¹Details on the different iCub versions are available at https://icub-tech-iit.github.io/documentation/icub_versions/.



(a) The iCub v2.7 humanoid.

(b) The iCub v3 humanoid.

Figure 3.5. The two versions of the iCub robot used as validation platforms for this thesis.

3.4.1 The iCub v2.7 humanoid

The iCub v2.7 humanoid, shown in Figure 3.5a, is a 104 cm tall robot weighing 33 kg, whose feet have a rounded shape delimited by a 19 cm long and 9 cm wide box. It possesses a total of 54 degrees of freedom, provided by its revolute joints distributed as follows: 4 in the head, 3 in the neck, 3 in the torso, 7 in each arm (3 in the shoulder, 1 in the elbow, 3 in the wrist), 9 in each hand, and 6 in each leg (3 in the hip, 1 in the knee, 2 in the ankle).

For the purpose of this thesis, a reduced set of joints that are the most relevant ones for the locomotion task is taken into account, namely 23 joints (i.e., all the joints but those of the head, neck, hands and wrists). These joints are all electrically actuated by brushless motors with harmonic drive transmissions and equipped with joint position encoders. Additionally, the torso and shoulder joints are mechanically coupled and driven by tendon mechanisms.

The robot is endowed with a series of sensors, of different nature, distributed across its body. It possesses six internal six-axis force-torque sensors, four attached to the base of each limb and the other two mounted in the feet, right below the ankles (see Figure 3.6a). It is also equipped with tactile sensors distributed on the torso, arms, legs and hands (both palm and fingertips), acting as an artificial skin [Cannata et al., 2008; Maiolino et al., 2013]. Since iCub v2.7 does not mount joint

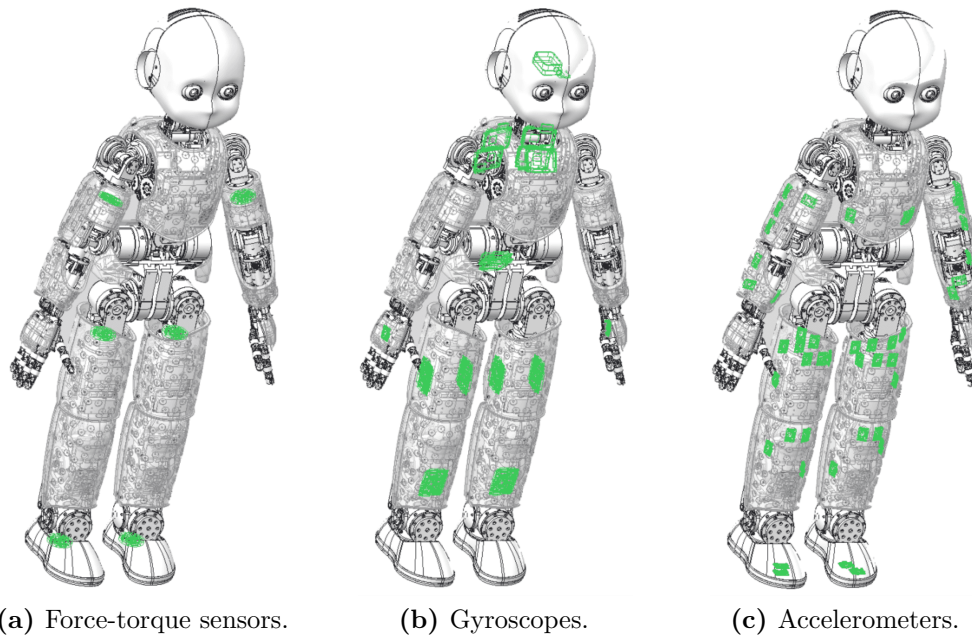


Figure 3.6. Distribution of (a) six-axis force-torque sensors and (b,c) inertial sensors, i.e., (b) gyroscopes and (c) accelerometers, on iCub v2.7.

torque sensors, both the internal torques and the external forces are estimated using the force-torque sensors and the skin [Fumagalli et al., 2012]. Moreover, a vast array of three-axis gyroscopes and three-axis accelerometers are spread over the robot structure, as illustrated in Figures 3.6b and 3.6c, respectively. Such a distributed inertial sensing setup is completed by two full-fledged Inertial Measurement Units, one on the head and the other one on the waist. Finally, two VGA cameras, a microphone and a speaker are located in the robot head.

More than 30 electronic boards operate the robot motors and sensors, connected by an Ethernet network in daisy chain. The boards provide three control strategies for the motors, namely position, velocity and torque. The robot is powered by either an external supplier, operating at 40V with about 3A current, or by a custom-made battery with a 9.3 Ah capacity included in a rigidly attached backpack, allowing for about 45 minutes of continuous usage.

The robot head is equipped with an on-board computer characterized by a 4-th generation Intel[®] Core i7 CPU @ 1.7 GHz and 8 GB of RAM, running Ubuntu Linux. Finally, the connection to the robot can be established via an Ethernet cable or a standard 5 GHz Wi-Fi network.

3.4.2 The iCub v3 humanoid

The iCub v3 humanoid, shown in Figure 3.5b, is the latest evolution of the iCub platform, and introduces novelties at the mechanics, actuation, electronics and sensing level [Dafarra et al., 2022a]. It is about 20 cm taller and 19 kg heavier than iCub v2.7, reaching a height of 125 cm and a weight of 52 kg. Its feet, composed by two separate rectangular sections each, are 25 cm long and 10 cm wide. The different dimensions of the two robots are shown in Figure 3.7.

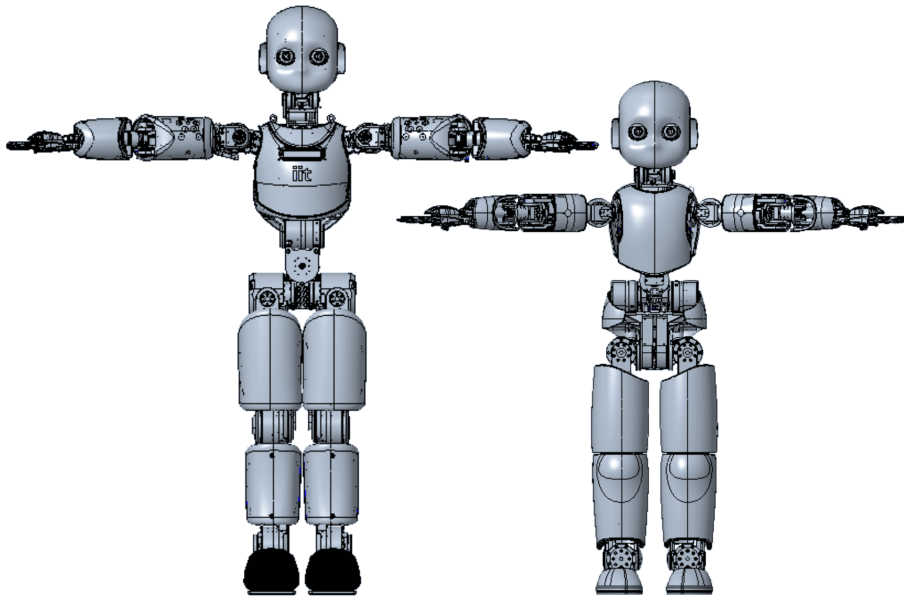


Figure 3.7. The iCub v3 (left) side to side to the iCub v2.7 (right).

The iCub v3 humanoid maintains the same number (54) and distribution of degrees-of-freedom of iCub v2.7. However, as opposed to iCub v2.7, its torso and shoulder joints are not coupled tendon-driven but serial direct mechanisms, which allows for a higher range of motion and greater mechanical robustness. Moreover, the increased size of the actuators on the legs (due to the increased overall robot weight requiring more powerful motors) leads to a different approach for the knee and ankle pitch joints, whose motor and actuator, rather than being on the same axis, are displaced and linked by belts. The 23 joints considered for this thesis as the most relevant for locomotion are all electrically actuated by three-phase brushless motors equipped with harmonic drive transmissions.

As regards sensing, iCub v3 extends the vast array of sensors of iCub v2.7 with higher resolution cameras for the eyes and two additional six-axis force-torque sensors. In particular, iCub v3 mounts eight six-axis force-torque sensors, of which six with a 45 mm diameter (F/T-45), as those of iCub v2.7, and two with a 58 mm diameter (F/T-58). The two F/T-58 are located in the middle of the robot thighs. Two F/T-45 are mounted at the shoulders, and two on each foot, connecting the two rectangular sections of each foot to its ankle assembly.

From the electronics point of view, iCub v3 shares with iCub v2.7 the architecture of distributed boards operating motors and sensors, connected via Ethernet. It is powered either by an external supplier or by a custom-made battery, which has a higher capacity of 10.05 Ah and is part of the torso assembly, rather than being located within an external backpack as for iCub v2.7.

The iCub v3 head is equipped with a more powerful on board computer, namely a 11-th generation Intel[®] Core i7 CPU @ 1.8 GHz and 16 GB of RAM, also running Ubuntu Linux. Additionally, the robot possess a NVIDIA[®] Jetson Xavier NX Module. Finally, as for iCub v2.7, the connection to the robot can be established through an Ethernet cable or wirelessly through a standard 5 GHz Wi-Fi network.

3.4.3 Software infrastructure

The core software infrastructure to control the robot is largely shared among the two aforementioned platforms, as well as among all the iCub robots. To control an iCub, the Yet Another Robot Platform (YARP) open-source middleware [Metta et al., 2006] is exploited. YARP consists of an abstraction layer whose main purpose is to let applications (modules) communicate, even if they run on different machines. Additionally, such an abstraction layer provides interfaces to interact with the robot physical devices no matter their actual implementation. Both sensor acquisition and motor control are therefore provided by YARP interfaces.

Besides the YARP middleware, software development on iCub can also rely on the iDynTree library [Nori et al., 2015], a library of robot dynamics algorithms for control, estimation and simulation, specifically designed for free-floating robots but also suitable for fixed-base ones. Written in C++, iDynTree provides Python and MATLAB bindings and is compatible with any robot described by a Universal Robot Description Format (URDF) model.

Finally, an ideal choice for developing software for the iCub robot is the Gazebo simulator [Koenig and Howard, 2004], an open-source simulator that efficiently handles complex multi-body systems and in particular supports the simulated version of the iCub robot. Since the simulated iCub comes with the corresponding YARP plugins [Mingo Hoffman et al., 2014] which allow to test the very same software on both the simulated and the real robot, software prototyping for iCub in Gazebo is transparent and therefore particularly convenient.

Part II
Contribution

Chapter 4

Learning Whole-Body Push-Recovery Strategies

In Part I, we introduced the background and presented the state of the art relevant for this thesis. This chapter illustrates, instead, the first contribution of the thesis, related to the application of deep reinforcement learning techniques to let general and robust whole-body balancing and push-recovery strategies for humanoid robots emerge in a simulation environment.

As opposed to classical control schemes often involving different specifically-designed controllers for handling different kinds of perturbation, and a tricky logic to switch from one to another, in this work we address balancing and push recovery via a single model-free whole-body control policy. To this purpose, we define a RL setting with a state space inspired by floating-base dynamics and a reward function shaped to guide the agent towards steady-state balancing while promoting the emergence of different transient push-recovery strategies. We validate the proposed approach in simulation on the iCub v2.7 humanoid robot – see Section 3.4.1 – demonstrating the emergence of robust momentum-based whole-body push-recovery strategies in addition to ankle, hip, and stepping ones.

More in detail, the chapter is organized as follows. With reference to the general RL agent-environment interaction loop depicted in Figure 2.5 and described in Section 2.2.1, we characterize the environment and the agent interacting with one another in our specific RL setting for balancing and push recovery of humanoid robots in Section 4.1 and Section 4.2, respectively. Extensive validation in simulation of the proposed approach for the tasks at issue is presented in Section 4.3. Finally, Section 4.4 concludes the chapter.

The content of this chapter partially appears in:

Ferigo, D., Camoriano, R., Viceconte, P. M., Calandriello, D., Traversaro, S., Rosasco, L., and Pucci, D. (2021). On the Emergence of Whole-body Strategies from Humanoid Robot Push-recovery Learning. *IEEE Robotics and Automation Letters*, 6(4):8561–8568.

Video <https://www.youtube.com/watch?v=FaOMtfYZiGA>

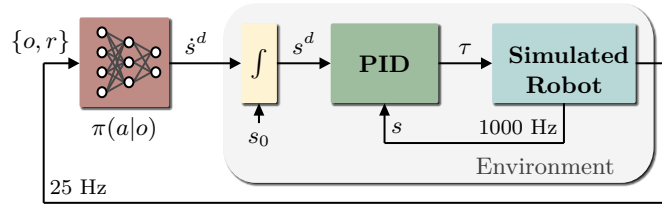


Figure 4.1. A sketch of the agent-environment interaction loop characterizing our RL setting. Given the observation o and the reward r , the agent provides desired joint velocities \dot{s}^d at 25 Hz. Such velocities are then integrated to retrieve reference joint positions s^d for the low-level PIDs, running at 1000 Hz as the physics simulation.

4.1 Environment

In the RL setting depicted in Figure 2.5, the environment comprises – by definition – everything outside the agent. It is characterized by its own dynamics and by the interface it exposes to the agent, which includes the state (for non-Euclidean states, the observation) and the reward. We define the state and the reward characterizing our environment in Section 4.1.1 and Section 4.1.2, respectively.

Since our agent generates reference joint velocities but we control the robot in position (see Section 3.4.1), in addition to its own dynamics our environment also includes a series of integrators in charge of retrieving the reference joint positions for the low-level PID controllers – see Figure 4.1. The physics simulation and the low-level PIDs run at 1000 Hz. Observations and rewards for the agent are instead produced at 25 Hz. The rationale behind such design choices for our agent-environment interaction loop is further explained in Section 4.2.

4.1.1 State definition

The state $x \in \mathcal{X}$ of our environment contains information about the kinematics and dynamics of the robot. In particular, it is defined as the tuple

$$x := \langle q, \nu, f_L, f_R \rangle \in \mathcal{X}, \quad (4.1.1)$$

where, given the robot base frame $B = (o_B, [B])$, q is the floating-base robot configuration defined in Eq. (1.2.17), namely,

$$q = \left({}^{\mathcal{I}}o_B, {}^{\mathcal{I}}R_B, s \right) \in \mathbb{R}^3 \times \text{SO}(3) \times \mathbb{R}^n, \quad (4.1.2)$$

and ν is the left-trivialized robot velocity defined in Eq. (4.1.3), namely,

$${}^B\nu = \begin{bmatrix} {}^Bv_{\mathcal{I},B} \\ {}^B\omega_{\mathcal{I},B} \\ \dot{s} \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{I}}R_B^\top {}^{\mathcal{I}}\dot{o}_B \\ \left({}^{\mathcal{I}}R_B^\top {}^{\mathcal{I}}\dot{R}_B \right)^\vee \\ \dot{s} \end{bmatrix} \in \mathbb{R}^{6+n}, \quad (4.1.3)$$

while f_L and f_R are the 6D forces applied to the left and right foot frames expressed in the inertial frame \mathcal{I} , i.e., ${}^{\mathcal{I}}f_L$ and ${}^{\mathcal{I}}f_R$, respectively (see Section 1.1.3).

Table 4.1. Components of the observation $o \in \mathcal{O}$, with the associated normalization ranges.

Name	Value	Set	Range	Unit
Joint positions	$o_s = s$	\mathbb{R}^{23}	$[s_{lb}, s_{ub}]$	rad
Joint velocities	$o_{\dot{s}} = \dot{s}$	\mathbb{R}^{23}	$[-\pi, \pi]$	rad/s
Base height	$o_h = {}^{\mathcal{I}}p_{\mathcal{B}}^z$	\mathbb{R}	$[0, 0.78]$	m
Base orientation	$o_R = ({}^{\mathcal{I}}\rho_B, {}^{\mathcal{I}}\phi_B)$	\mathbb{R}^2	$[-2\pi, 2\pi]$	rad
Contact configuration	$o_c = (c_L, c_R)$	$\{0, 1\}^2$	-	-
CoP forces	$o_f = (f_{L,CoP}^z, f_{R,CoP}^z)$	\mathbb{R}^2	$[0, 330]$	N
Feet positions	$o_F = ({}^{\mathcal{B}}p_L, {}^{\mathcal{B}}p_R)$	\mathbb{R}^6	$[0, 0.78]$	m
CoM velocity	$o_v = {}^{\bar{G}}v_{CoM}$	\mathbb{R}^3	$[0, 3]$	m/s

The observation $o \in \mathcal{O}$ computed from the non-Euclidean state $x \in \mathcal{X}$, i.e., the perceptual input actually available to the agent, is defined as the tuple

$$o := \langle o_s, o_{\dot{s}}, o_h, o_R, o_c, o_f, o_F, o_v \rangle \in \mathcal{O} := \mathbb{R}^{62}, \quad (4.1.4)$$

where:

- $o_s = s \in \mathbb{R}^{23}$ are the positions of the controlled joints (see Section 4.2.1) in radians, normalized component-wise within the joint hard limits $[s_{lb}, s_{ub}]$;
- $o_{\dot{s}} = \dot{s} \in \mathbb{R}^{23}$ are the velocities of the controlled joints, normalized component-wise in $[-\pi, \pi]$ rad/s;
- $o_h = {}^{\mathcal{I}}p_{\mathcal{B}}^z \in \mathbb{R}$ is the height of the robot base frame \mathcal{B} , normalized in $[0, 0.78]$ m, where 0.78 m represents a highly unlikely value to overcome for the base height of our robot, even in the case of extremely dynamic motions;
- $o_R = ({}^{\mathcal{I}}\rho_B, {}^{\mathcal{I}}\phi_B) \in \mathbb{R}^2$ is a tuple containing the roll and pitch angles of the robot base frame \mathcal{B} with respect to the inertial frame \mathcal{I} , normalized component-wise in $[-2\pi, 2\pi]$ rad;
- $o_c = (c_L, c_R) \in \{0, 1\}^2$ is a tuple of boolean values defining whether each foot is in contact with the ground, with no need for normalization;
- $o_f = (f_{L,CoP}^z, f_{R,CoP}^z) \in \mathbb{R}^2$ is a tuple containing the vertical forces applied to the local centers of pressure of the feet, normalized component-wise in $[0, 330]$ N, where, if m is the robot mass and g the gravitational constant, $mg = 330$ N is the nominal weight force of the robot;
- $o_F = ({}^{\mathcal{B}}p_L, {}^{\mathcal{B}}p_R) \in \mathbb{R}^6$ is a tuple containing the positions of the feet frames expressed in the base frame, whose norm is normalized in $[0, 0.78]$ m;
- $o_v = {}^{\bar{G}}v_{CoM} \in \mathbb{R}^3$ is the linear velocity of the CoM expressed in the mixed reference frame \bar{G} – see Section 1.2.3 – whose norm is normalized in $[0, 3]$ m/s.

The observation components including relevant information for the agent to distinguish promising states among those encountered throughout the training process are also summarized in Table 4.1.

Although this work is focused on the design and validation of learning methods for the emergence of whole-body balancing and push-recovery strategies in simulation and does not address sim-to-real transfer, it is worth to mention that we carefully select the state and observation so that they can be either measured or estimated online to promote policy transfer on real robots. We avoid indeed to include in the state definition in Eq. (4.1.1) measurements from sensors that are known to be noisy and values that cannot usually be estimated online with sufficient accuracy. For instance, the ground reaction forces included in the state can be estimated on-board on the iCub v2.7 humanoid [Nori et al., 2015].

4.1.2 Reward shaping

In our RL setting, the dynamic behaviours for balancing and push recovery are learned by the agent exclusively through autonomous exploration and interaction with the environment. No expert data, e.g., specialized motion capture data collected by human experts, are used to guide the agent exploration during training nor shape the resulting behaviours. In such cases, *reward shaping* plays a vital role. Defining a reward structure that encodes all the relevant performance indicators for the task at hand is indeed the only mechanism available for promoting successful agent exploration strategies. How to design in practice such a suitable reward remains however an open problem, often addressed empirically.

Inspired by previous works [Yang et al., 2017, 2018], we approach the reward design problem with a reward structure that we iteratively extend – exploiting domain knowledge in humanoid control – to allow high-dimensional whole-body strategies to emerge. The reward resulting from this heuristic process includes a series of terms categorized as regularizers, steady-state, and transient components, all detailed in Section 4.1.2. In particular, we report there the reward formulation that, among the ones we validated, led to the most satisfactory results, shown in Section 4.3. Given the duration of each individual training in our setup (around 2 days), we could not afford an exhaustive ablation study to select with enough confidence the minimal formulation of the reward guaranteeing the same performances and we were forced to rely on heuristic search guided by domain expertise only. However, thanks to our interpretable reward formulation illustrated in Section 4.1.2, we were able to shape the reward in such a way that each of its many components positively affects the total reward as training progresses – see Section 4.3.1.

Radial basis function kernels

Our heuristic search for suitable reward components and their effective combination to compute the total reward exploits the formulation of individual reward components using Radial Basis Function (RBF) kernels [Yang et al., 2017]. Given two vectors $x, x^* \in \mathbb{R}^k$, the RBF kernel is defined as the function

$$K(x, x^*) = e^{-\tilde{\gamma}\|x-x^*\|^2} \in [0, 1], \quad (4.1.5)$$

where $\tilde{\gamma}$ is the *kernel bandwidth*. RBF kernels measure similarities between their inputs. If its inputs x and x^* coincide, the RBF kernel returns a unitary output, i.e.,

$$K(x^*, x^*) = 1. \quad (4.1.6)$$

The sensitivity of the kernel, i.e., how different the inputs x and x^* must be for the kernel output to approach 0, can be tuned through $\tilde{\gamma}$. Let us introduce the pair (x_c, ϵ) , with $x_c, \epsilon \in \mathbb{R}^+$ and $\epsilon \ll 1$. By exploiting x_c and ϵ , we can parameterize $\tilde{\gamma}$ as

$$\tilde{\gamma} = \frac{\ln(\epsilon)}{x_c^2}. \quad (4.1.7)$$

This formulation ensures that, given an input x_m such that $\|x_m - x^*\| = x_c$, the kernel output is equal to ϵ , i.e.,

$$\|x_m - x^*\| = x_c \Rightarrow K(x_m, x^*) = \epsilon. \quad (4.1.8)$$

Assuming a constant ϵ , the kernel sensitivity only depends on x_c , referred to as the kernel *cutoff* parameter.

We exploit RBF kernels and the parameterization of $\tilde{\gamma}$ in Eq. (4.1.7) to formulate our individual reward components. If x is the *measurement* associated to a certain component and x^* the *target* value for that component, the kernel $K(x, x^*)$ provides indeed an estimate of the similarity between x and x^* , normalized in $[0, 1]$. For measurements x that are further from the target x^* than the cutoff x_c , the kernel returns an output smaller than ϵ , and is said to be inactive. The kernel becomes instead *active* for measurements x closer to the target x^* than the cutoff x_c , with its output approaching 1 as x gets closer to x^* . A suitable tuning of the cutoff x_c of the RBF kernel related to each reward component – depending on the scale and variance of its associated measurements – is therefore crucial to allow for a proper activation of the kernel during training.

Rather than defining the total reward $r \in \mathbb{R}$ returned by our environment simply as the weighted sum of individual reward components, i.e.,

$$r = \sum_i w_i r_i, \quad (4.1.9)$$

with r_i denoting each reward component and w_i its associated weight, we define r as the weighted sum of individual reward components processed by a RBF kernel, i.e.,

$$r = \sum_i w_i K(x_i, x_i^* | x_{c_i}), \quad (4.1.10)$$

where w_i , x_i and x_i^* are the weight, the current measurement and the target value of the i -th reward component, respectively, while x_{c_i} is the cutoff of its associated RBF kernel. With the RBF kernels normalizing the distance between x_i and x_i^* in $[0, 1]$ – no matter its original scale – for each reward component, Eq. (4.1.10) represents a weighted sum over similarly-scaled reward components, much more interpretable and easier to shape with respect to Eq. (4.1.9).

Monitoring, during training, the output of the kernels associated to the individual reward components via Eq. (4.1.10) enables indeed a useful reward shaping strategy. First, the cutoff parameters x_{c_i} are adjusted to have all the kernels properly activated. Then, the weights w_i are adjusted with the twofold aim of increasing the relevance of the reward components whose kernel output is still far from 1 (i.e., whose average measurement is still far from the target) and promoting reward components that are considered crucial for learning the task. In summary, this is how we retrieved the reward illustrated in the next section, which includes a series of terms individually selected based on domain knowledge and effectively combined using RBF kernels.

Reward definition

The terms summed up according to Eq. (4.1.10) to obtain the the total reward $r \in \mathbb{R}$ of our environment can be categorized as regularizers, steady-state, and transient. *Regularizers* are terms often used in optimal control for the minimization of control action and joint torques. *Steady-state* components help to obtain the balancing behaviour in the absence of external perturbations, and are active only during double support (DS) phases. Finally, *transient* components favor the emergence of push-recovery whole-body strategies.

In the following, we detail all the Regularizers (R), Steady-state (S) and Transient (T) terms composing the total reward r , which are also summarized in Table 4.2 along with their associated weights, targets and kernel cutoff parameters.

Joint torques r_τ (R). Joint torques applied by the PID controllers as shown in Figure 4.1 are penalized. Since the agent-environment interaction takes place at 25 Hz while the low-level PID controllers run at 1000 Hz, several torques τ_i , with $i \in \{1, \dots, 40\}$, are actuated between two subsequent reward computations. For each controlled joint, we define the joint torques reward component r_τ as the average

$$\tau_{step} = \frac{1}{40} \sum_{i=1}^{40} |\tau_i| \quad (4.1.11)$$

of all the applied torques' absolute values between two subsequent agent steps.

Joint velocities $r_{\dot{s}}$ (R). The control scheme employed by the agent – see Section 4.2 – ensures continuous reference joint positions. However, during training the learning algorithm explores the action space of joint velocities. To promote smoother trajectories, we penalize the norm of the latest action a – see Section 4.2. The joint velocities reward component $r_{\dot{s}}$ penalizing the norm of a corresponds to the minimization of the control effort.

Postural r_s (S). Whole-body humanoid controllers following a stack-of-tasks approach – see Section 3.1.3 – usually combine high and low priority tasks. The postural task is notably one of the most used [Nava et al., 2016], although it is usually assigned a low priority. We include a *fixed* postural reward term that helps to reach a target posture during balancing, otherwise realized in odd joint configurations corresponding to local minima found by the policy. The postural reward component r_s penalizes the mismatch between the sampled joint configuration s and the reference joint configuration \bar{s}_0 (postural), shown in the top-left corner frame of Figure 4.4.

CoM projection r_{CoM} (S). Statically balanced robots, in order to maintain stability, keep the ground projection of their CoM within their support polygon (SP), defined as the convex hull¹ (CH) of the contact points between their feet and the ground. With the same aim, we introduce a boolean component rewarding the agent if its CoM projection on the ground p_{CoM}^{xy} remains within its SP. For additional safety, we shrink the SP by a 2.5 cm margin all along its perimeter.

¹The convex hull of a set of points is the smallest convex region which contains them all.

Table 4.2. Components of the total reward r , with their associated targets and weights. For the reward terms processed by a RBF kernel, i.e., all the non-boolean terms but the negative reward related to the episode termination, the associated properly-tuned cutoff is reported. The table also indicates whether each component is active in single support (SS) and in double support (DS).

Name	Class	Symbol	Unit	Value x	Target x^*	Weight w	Cutoff x_c	SS	DS
Joint torques	R	r_τ	Nm	τ_{step}	0_n	5	10.0	✓	✓
Joint velocities	R	$r_{\dot{s}}$	rad/s	a	0_n	2	1.0	✓	✓
Postural	S	r_s	rad	s	\bar{s}_0	10	0.13		✓
CoM projection	S	r_{CoM}	-	$p_{CoM}^{xy} \in SP$	1	10	-		✓
Horizontal CoM velocity	S	r_v^{xy}	m/s	v_{CoM}^z	$\omega(p_{CoM}^{xy} - p_{SP})$	2	0.5		✓
Vertical CoM velocity	T	r_v^z	m/s	v_{CoM}^{xy}	0	2	1.0	✓	✓
Centroidal momentum	T	r_h	kg m ² /s	$\ \bar{G}h^p\ ^2 + \ \bar{G}h^\omega\ ^2$	0	1	50.0	✓	✓
Feet contact forces	T	$\{r_f^L, r_f^R\}$	N	$\{f_{L,CoP}^z, f_{R,CoP}^z\}$	$mg/2$	$\{4,4\}$	$mg/2$	✓	✓
Feet CoPs	T	$\{r_p^L, r_p^R\}$	m	$\{p_{L,CoP}, p_{R,CoP}\}$	$\{p_{L,CH}, p_{R,CH}\}$	$\{10,10\}$	0.3	✓	✓
Feet orientation	T	$\{r_o^L, r_o^R\}$	-	$\{\mathcal{I}r_L^z \cdot e_3, \mathcal{I}r_R^z \cdot e_3\}$	1	$\{1.5,1.5\}$	0.01	✓	✓
Feet in contact	T	r_c	-	$c_L \wedge c_R$	1	2	-	✓	✓
Links in contact	T	r_l	-	c_l	0	-10	-	✓	✓

Horizontal CoM velocity r_v^{xy} (S). We define a target horizontal velocity for the CoM as the vector pointing from the ground projection of the CoM p_{CoM}^{xy} to the center of the SP p_{SP} . In order to promote faster motions if the CoM is relatively close to the ground, the magnitude of the target horizontal CoM velocity is amplified by the time constant of the LIP model $\omega = \sqrt{\frac{g}{p_{\text{CoM}}^z}}$, where p_{CoM}^z is the CoM height and g is the gravitational constant – see Section 1.3.1. Rewarding a CoM horizontal velocity v_{CoM}^{xy} as close as possible to the aforementioned target encourages the motion of the CoM ground projection towards the center of the SP. In other words, this term promotes static balancing in DS (and is not active in SS, see Table 4.2).

Vertical CoM velocity r_v^z (T). We penalize vertical motions of the robot CoM, i.e., vertical CoM velocities v_{CoM}^z , promoting instead the exploitation of the horizontal component v_{CoM}^{xy} for balancing purposes.

Centroidal momentum r_h (T). Among the controlled joints detailed in Section 4.2 there are also joints belonging to the robot torso and its arms. Also the momentum generated by the upper body can therefore be exploited for balancing and push recovery. This term minimizes the sum of the norms of the linear and angular components of the robot centroidal momentum $\bar{c}h$ – see Section 1.2.3.

Feet contact forces $\{r_f^L, r_f^R\}$ (T). This reward term pushes the transient towards a steady-state pose in which the robot weight is distributed equally on the two feet. To do so, we reward vertical forces applied to the feet local CoP $\{f_{L,\text{CoP}}^z, f_{R,\text{CoP}}^z\}$ as close as possible to half of the robot nominal weight, i.e., $mg/2$.

Feet CoPs $\{r_p^L, r_p^R\}$ (T). Beyond equally-distributed contact forces at the feet CoPs, we also promote the local feet CoP positions $\{p_{L,\text{CoP}}, p_{R,\text{CoP}}\}$ to be located at the center of the convex hull (CH) of the correspondent foot $\{p_{L,\text{CH}}, p_{R,\text{CH}}\}$.

Feet orientation $\{r_o^L, r_o^R\}$ (T). The first training attempts exhibit feet tipping behaviours, i.e., policies maintaining balance without keeping the feet in stable contact with the ground. Targeting flat terrains only, we discourage feet tipping by promoting feet orientations such that the feet soles are parallel to the ground. If ${}^{\mathcal{I}}R_F = [{}^{\mathcal{I}}r_F^x \quad {}^{\mathcal{I}}r_F^y \quad {}^{\mathcal{I}}r_F^z]$ represents the orientation of the foot frame $F \in \{L, R\}$ with respect to the inertial frame \mathcal{I} , this reward component promotes the alignment of ${}^{\mathcal{I}}r_F^z$ with the z axis $e_3 \in \mathbb{R}^3$ of the inertial frame \mathcal{I} , pointing against gravity. We implement this alignment by rewarding the scalar products $\{{}^{\mathcal{I}}r_L^z \cdot e_3, {}^{\mathcal{I}}r_R^z \cdot e_3\}$ close to 1.

Feet in contact r_c (T). We encourage the robot feet to stay on the ground. In order to promote steps and increase movement freedom, rather than rewarding the whole foot surface to be in contact with the ground, we reward cases in which both feet have at least one contact with the ground, i.e., $c_L \wedge c_R = 1$, with c_L and c_R indicating contacts for the left and right foot, respectively.

Links in contact r_l (T). If any link but the feet is in contact with the ground, we consider the balancing task failed and terminate the episode with a reward of -10 .

4.1.3 Episode specifications

Given the state $x \in \mathcal{X}$ and the observation $o \in \mathcal{O}$ defined in Section 4.1.1 along with the reward $r \in \mathbb{R}$ illustrated in Section 4.1.2, our balancing and push-recovery task is structured as a *continuing* task² truncated according to certain early termination conditions or after $T = 150$ s. Therefore, we refer to *episodes* as agent-environment interactions *truncated* due to maximum length or failure. In the following, we characterize the episodes occurring in our RL setting in terms of the initial state selection and the domain randomization performed at the beginning of each episode, the external perturbations applied throughout the episodes, and the early termination criteria which determine their conclusion.

The initial state x_0 in which the agent begins each episode is sampled from an *initial state distribution* $\rho_0(x) : \mathcal{X} \rightarrow \Pr(\mathcal{X})$. We observe indeed that sampling x_0 , and in particular the initial joint positions s_0 and velocities \dot{s}_0 , from a distribution with small variance positively affects the policy exploration without degrading the learning performance. Therefore, at the beginning of each episode, we sample the j -th joint initial position $s_{j,0}$ from the Gaussian distribution $\mathcal{N}(\mu = \bar{s}_{j,0}, \sigma = 10\text{deg})$, where $\bar{s}_{j,0}$ is the j -th joint reference also used for the postural reward r_s , and its initial velocity $\dot{s}_{j,0}$ from $\mathcal{N}(\mu = 0, \sigma = 90\text{deg/s})$. To avoid the feet from interpenetrating the ground in x_0 , we increase the initial base height ${}^{\mathcal{I}}o_{B,0}^z$ so that the robot starts the episode with no feet in contact with the ground, which also encourages the agent to learn how to land and deal with impacts.

At the beginning of each episode, we implement the following *domain randomization*. To account for possible model inaccuracies, the masses of the k robot links are sampled from k normal distributions $\mathcal{N}_i(\mu = m_i, \sigma = 0.2m_i)$, where m_i is the nominal mass of the i -th link in the robot model. To address a wider range of materials for the robot feet and the ground, we randomize the Coulomb friction μ_c of the feet by sampling it from the uniform distribution $\mathcal{U}(0.5, 3)$. Finally, since the simulation does not include the real dynamics of the actuators, we further encourage policy robustness by randomizing the delay applied to the position references fed to the PID controllers, sampled from $\mathcal{U}(0, 20)$ ms.

Throughout the episodes, we promote exploration beyond the choice of the initial state and favor the emergence of push-recovery strategies by applying *external perturbations* in the form of 3D forces to the robot base frame. Such forces have a fixed magnitude of 200 N and duration of 200 ms, resulting into a 40 Ns impulse which, normalized on the 33kg-weighting iCub v2.7, acts as a 1.21 m/s push per unitary mass. Their direction is sampled from a uniform spherical distribution. Also their frequency is sampled from a uniform distribution, with an average of 5 simulated seconds between two subsequent force applications.

Despite the continuing nature of the balancing and push-recovery task, we define early-termination criteria for the episode to stop as soon as the state reaches a subspace from which it is not possible to recover or uninteresting to keep exploring. The state space of interest for our work is where the robot is *almost* standing on its feet, therefore we terminate the episodes as soon as it falls on the ground. A fall is detected in the event of any link but the feet making contact with the ground.

²For the balancing and push-recovery task, there exist no terminal states such that the task can be considered accomplished. The best the agent can do is indeed to keep balancing over and over.

4.2 Agent

In our RL setting depicted in Figure 4.1, given the observation $o \in \mathcal{O}$ from the environment, the agent selects the action a , characterized in Section 4.2.1, according to its policy $\pi(a|o)$, parameterized and trained as reported in Section 4.2.2.

Inspired by the results obtained in previous works [Yang et al., 2017, 2018], we design a low-frequency agent interacting with the environment at 25 Hz. Although increasing the policy rate would likely provide a more responsive behaviour and possibly help to transfer the policy to real platforms, given the acceptable performances observed at 25 Hz and the multitude of parameters characterizing our setup, we decide to keep the policy rate unchanged and tune other parameters instead.

4.2.1 Action definition

In our nested structure, the agent’s policy generates an action $a \in \mathbb{R}^{23}$ composed of the reference velocities \dot{s}^d for a large subset of the robot joints, referred to as *controlled joints*. The set of controlled joints includes joints belonging to the robot legs, torso, and arms. Hands, wrists, and neck, which arguably play a minor role in balancing, are not included in the set of controlled joints and are simply maintained locked in their natural positions, i.e., at 0 deg. For each controlled joint, the policy provides a reference joint velocity bounded in $[-180, 180]$ deg/s. The policy commands each joint independently, and is not subject to symmetry constraints. The output of the policy is then integrated to get the reference joint positions s^d which are actually fed to the low-level PID controllers.

Commanding joint velocities rather than joint positions prevents target joint positions from being too distant from each other in consecutive control steps. Especially at training onset, directly commanding joint positions would lead to highly jumpy references that the low-level PID controllers would not be able to track at all, affecting therefore the capability of the policy to discover the actual effect of the selected action on the environment state transition from x_t to x_{t+1} . Commanding joint velocities subsequently integrated to get the actual joint position references allows, instead, to use a stochastic policy that generates discontinuous actions while maintaining continuous inputs to the low-level PID controllers with no need for additional filters.

Finally, let us clarify why we opt in the first place for the design of a policy for a position-controlled rather than a torque-controlled robot. First, high-level action parameterizations including local feedback, such as PID targets, have been shown to lead to higher policy performance and learning speed with respect to low-level actions such as torque references across several locomotion tasks [Peng and van de Panne, 2017]. Then, despite not addressing sim-to-real transfer in this work, we take into account the current capabilities of our target platform, the iCub v2.7 humanoid illustrated in Section 3.4.1, which suggest a position-control approach for dynamic push-recovery. Although past works [Nori et al., 2015] on this platform have shown interesting results for balancing in torque control, more recent works [Dafarra et al., 2018; Romualdi et al., 2020] targeting scenarios characterized by richer contacts with the environment, e.g., walking, report instead bottlenecks for torque control preventing high performances for highly-dynamic tasks.

Table 4.3. PPO, SGD, and distributed training parameters.

Parameter	Value
Clip parameter ϵ	0.3
Discount factor γ	0.95
GAE parameter λ	1.0
Learning rate α	0.0001
Batch size	10000
Minibatch size	512
Number of SGD epochs	32
Number of parallel workers	32

4.2.2 Policy representation and training

We define the agent as an *actor-critic* – see Section 2.2.1 – parameterizing both its stochastic policy $\pi(a|o)$, which selects the action a to take given the observation o , and its value function $\hat{V}(o)$, which estimates the average return when starting from the observation o and then following the policy π , as deep feedforward neural networks – see Section 2.1.1.

In particular, we represent the policy and the value function using two distinct feedforward neural networks composed of two fully-connected hidden layers, with 512 and 128 units each, followed by a linear output layer. The hidden units use the ReLU activation function in Eq. (2.1.4), while the output layer the linear activation function in Eq. (2.1.5). The networks do not share any layer.

We select *Proximal Policy Optimization* (PPO) – see Section 2.2.3 – as training algorithm, in its variant which includes both the clipped surrogate loss and the adaptive KL penalty loss, expressed by Eq. (2.2.42). We set the clip hyperparameter $\epsilon = 0.3$. As advantage function estimator \hat{A}_t , we choose the Generalized Advantage Estimator (GAE) [Schulman et al., 2016], with a discount factor $\gamma = 0.95$ and a decay parameter $\lambda = 1.0$.

Since the chosen PPO algorithm scales gracefully to a setup where the batch samples are collected from multiple workers in parallel, we exploit a *distributed setup* for training. Our distributed setup is composed of 32 parallel *workers* with an independent copy of the environment, and one *trainer*. The workers run on CPU resources only, while the trainer has access to the GPU for accelerating the optimization process. The trainer is indeed in charge of performing a policy optimization step every 10000 transitions collected by the parallel workers. In particular, once a batch of 10000 on-policy transitions has been collected, the trainer optimizes the neural networks parameterizing the policy and the value function using Stochastic Gradient Descent (SGD) – see Section 2.1.2. The optimization is repeated for 32 epochs per batch, using minibatches of 512 samples each, with a learning rate $\alpha = 0.0001$. Each training execution is interrupted once it reaches 20 millions of agent steps, roughly equivalent to 7 days of simulated experience.

All the agent training parameters, empirically found given the impossibility to conduct an exhaustive grid-search in our setup because of the considerable duration of each training instance, are also summarized in Table 4.3.

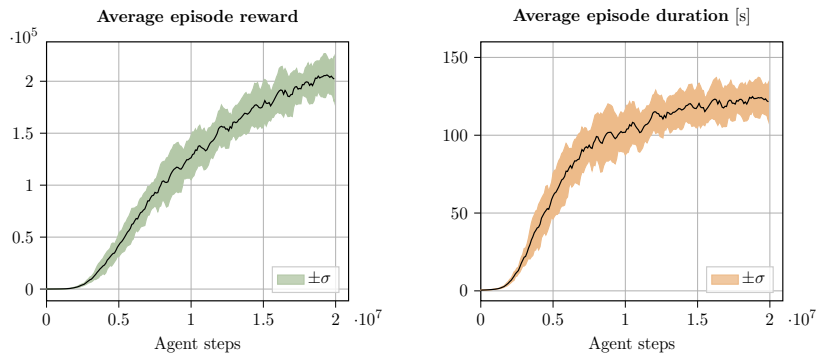


Figure 4.2. Average learning curves over 11 independent training executions. Mean and variance of the episode reward (left) and duration (right) across 20 millions agent steps performed for each training instance are reported. As the training proceeds, both curves increase, with the average episode duration getting closer to its maximum of 150 s.

4.3 Results

This section illustrates the results obtained by training the agent detailed in Section 4.2 in the environment described in Section 4.1 according to the training specifications reported in Section 4.2.2. The proposed approach is validated in simulation on the iCub v2.7 humanoid model – see Section 3.4.1.

First, let us detail the frameworks and tools we exploit for the practical implementation of our RL setting. To simulate the environment dynamics, we use the Gazebo Sim simulator embedded into the gym-ignition framework [Ferigo et al., 2020], compatible with OpenAI Gym [Brockman et al., 2016]. As physics engine, we enable the default one for Gazebo Sim, i.e. DART [Lee et al., 2018a]. For calculating rigid-body dynamics quantities, using an accurate model of the free-floating system dynamics from Eq. (1.2.26), we take advantage of the iDynTree library [Nori et al., 2015]. Finally, to train the agent in a distributed fashion we adopt the high-performance PPO implementation available within the RLLib library [Liang et al., 2018], built on top of the Ray framework [Moritz et al., 2018].

4.3.1 Reward shaping

Figure 4.2 reports the learning curves of the average episode reward and duration over $N = 11$ independent training instances for the selected agent-environment configuration, run with different random seeds. As thoroughly illustrated in [Henderson et al., 2018], high variance in performances due to the stochasticity of the environment and the learning process is a major concern with deep RL agents and therefore results averaged on a small number of trials, e.g., $N < 5$, or worse on the top- N trials only, are potentially misleading. No matter the high training time required for the $N = 11$ training instances, i.e., around 20 days in total, we decide to extensively test the agent performances without discarding any training instance. Both average episode reward (Figure 4.2, left) and duration (see Figure 4.2, right) across the N trials exhibit consistent growth and low variance as the training proceeds. In particular, episode duration approaches its maximum of 150 s towards the end of the training instances, set to 20 million agent steps.

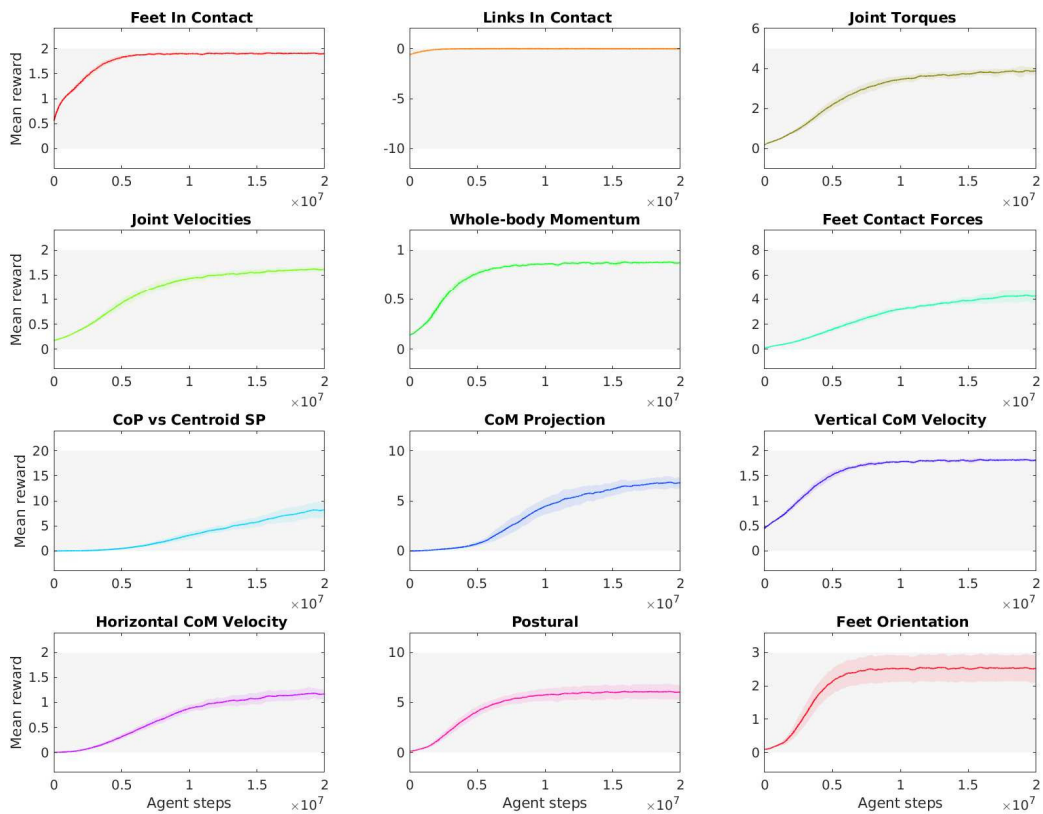


Figure 4.3. Average component-wise episode reward over 11 independent training executions. For each reward component, weighted by the correspondent weight from Table 4.2, mean and average across the 20 millions agent steps of each training instance are reported. Light gray areas represent upper and lower bounds of each weighted reward component.

Besides monitoring the evolution of the total reward r reported in Figure 4.2, as training progresses we also inspect the evolution of the individual reward components that contribute to r according to Eq. (4.1.10). As explained in Section 4.1.2, such a *component-wise* inspection of the reward is crucial to select helpful reward terms while simultaneously tuning their associated RBF kernel cutoff x_{c_i} and weight w_i . We report in Figure 4.3 the evolution over $N = 11$ independent training instances of the average reward for the 12 individual terms composing our total reward r . For each component, the average output of its associated RBF kernel is shown, weighted by the correspondent weight w_i (see Table 4.2). In the best case, each *weighted* average reward component would reach the associated w_i , e.g., $w_i = 2$ for the horizontal CoM velocity component or $w_i = 8$ for the feet contact forces component (4 per foot). The links in contact component, related to the episode termination and characterized by a negative weight, is instead expected to reach 0 as maximum. Figure 4.3 shows as in our RL configuration, while training progresses, each reward term consistently increases with low variance, and some terms approach their maximum value. This confirms that each selected component positively contributes to the total reward, resulting therefore helpful to guide the agent exploration towards the task.

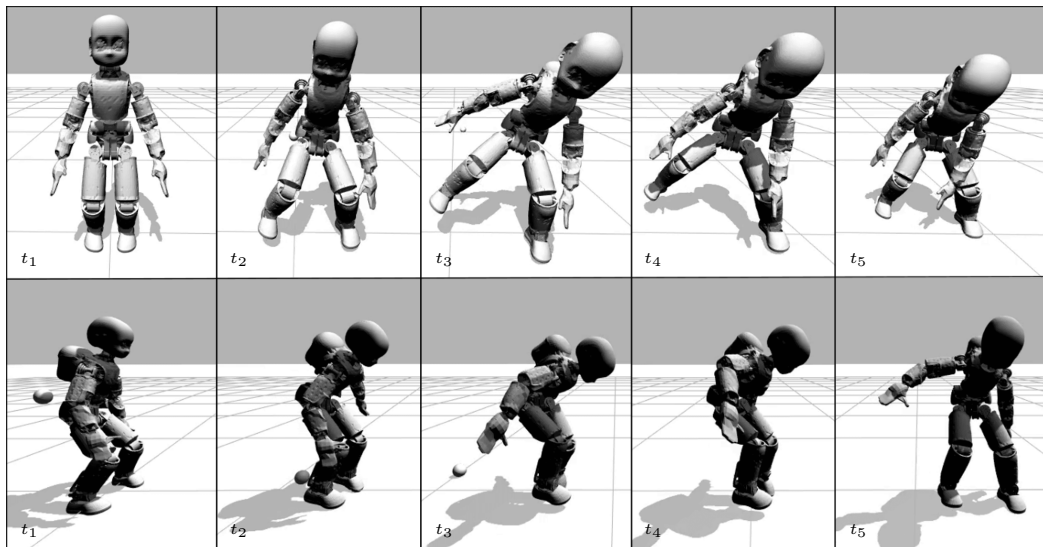


Figure 4.4. Two motion sequences of the simulated iCub v2.7 humanoid implementing a combination of ankle, step, and articulated momentum-based recovery strategies emerged from the learning process. In both the sequences, the robot is pushed by a 4 kg sphere thrown horizontally at 1.5 m/s. The impact takes place on the robot right side and on its back in the upper and lower sequence, respectively. For each sequence, five representative frames show the highly-dynamic motions through which the policy faces the push without falling.

4.3.2 Emerging behaviours

A qualitative analysis of the balancing and push-recovery strategies demonstrated by the policy after training highlights the emergence of rich whole-body recovery behaviours, no matter the robot dynamic randomization performed at training stage – see Section 4.1.3 – which could have caused the control policy to be overly-conservative [Xie et al., 2019]. The emerging behaviours include the well-known ankle, hip and stepping strategies [Stephens, 2007] combined with more articulated momentum-based strategies. Such complex behaviours that exploit the total momentum of the robot kinematic structure are indeed enabled by letting the policy control also the joints of the upper body, as opposed to previous works tackling RL-based humanoid push recovery by considering lower-body joints only [Yang et al., 2017, 2018]. Moreover, let us recall that all the emerging strategies are handled simultaneously by our control policy, with no need of any switching logic, as opposed to many classical control schemes for humanoid push-recovery.

Figure 4.4 shows two representative sequences in which the learned control policy faces remarkable pushes resulting from throwing at high speed a heavy object towards the robot, i.e., a 4 kg sphere shot horizontally at 1.5 m/s. This is an unseen scenario during training, used to trigger more realistic force profiles for testing purposes. At training time, constant forces for a fixed time interval are indeed applied to the robot base link only – see Section 4.1.3. Nevertheless, such constant forces applied during training succeed in triggering agent exploration enough to discover the complex behaviours demonstrated at testing time, and detailed in the following.

In the upper sequence of Figure 4.4, the 4 kg sphere hits the robot on the right side of its waist (right before t_2). The strong push makes the robot jump on its left side (t_3). After a few consecutive small jumps (t_4), the policy successfully achieves landing in double support (t_5). Throughout the motion, both the torso and the arms are exploited to vary the centroidal angular momentum, as you can see by comparing, for instance, t_3 and t_5 . The simulation from which these screenshots are taken is shown in a dedicated portion of the accompanying video available at <https://youtu.be/FaOMtfYZiGA?t=88>. In the video you can also appreciate how, once in double support, a straighter posture (closer to the reference steady-state postural shown at t_1) is finally retrieved.

In the lower sequence of Figure 4.4, instead, the 4 kg sphere hits the robot on the back of its torso (right after t_1). As a consequence, the robot performs a series of small steps forward (from t_2 to t_4) and a final rotational adjustment (t_5). Also here, it can be seen how the upper body is actively exploited to maintain balance. For instance, the torso pitch varies significantly if you compare t_3 with either t_2 or t_4 . At the same time, the configuration of the robot arms is remarkably different between t_2 and t_5 . Although the simulation from which these screenshots are taken is not included in the accompanying video, a larger variety of emerging push-recovery strategies is displayed in the final portion of the video available at <https://youtu.be/FaOMtfYZiGA?t=110>.

A few additional remarks on the emerging behaviours follow. First, commanding joint velocities rather than joint positions – see Section 4.2.1 – leads to fairly smooth behaviours, as one would expect. On the other hand, we cannot compare this choice against a policy that directly controls joint positions, since our training attempts for such a policy (with no additional filter, not to introduce further delays in the control pipeline) do not succeed. Then, let us recall that no self-collision avoidance is implemented in the proposed approach. For instance, as it can be seen in the accompanying video, the robot arms often collide with their corresponding legs. Despite this, interestingly, the policy learns to keep the legs distant from each other, and therefore no legs crossing nor feet touching is observed at testing time – although the robot kinematics does not prevent it. Finally, let us remark the need for the steady-state postural reward – see Section 4.1.2 – without which undesirable foot tipping and asymmetric steady-state configurations can be noticed – see the accompanying video at <https://youtu.be/FaOMtfYZiGA?t=63>.

The main drawback of the emerged behaviours is not to be as natural as human ones, and also probably unfeasible. Especially when the robot is subject to strong external pushes, the resulting recovery strategy may not look natural at all. The control policy tends to prefer small jumps to full steps, which we relate to three main factors: the stiffness given by the low-level PIDs, the difficulty of accurate contact modeling, and the absence of reference motion capture data.

As concerns low-level control, we already mentioned our choice of controlling the robot in position based on the current state-of-the-art controllers for iCub v2.7 – see Section 4.2.1. Moreover, despite introducing variable delay – see Section 4.1.3 – our simulations do not saturate joint torques. The joint torques minimization in the reward does not prevent occasional high torque spikes synthesized by the PIDs. Integrating more realistic actuator dynamics or adopting intrinsically-compliant torque-based controllers should (at least partially) address these issues.

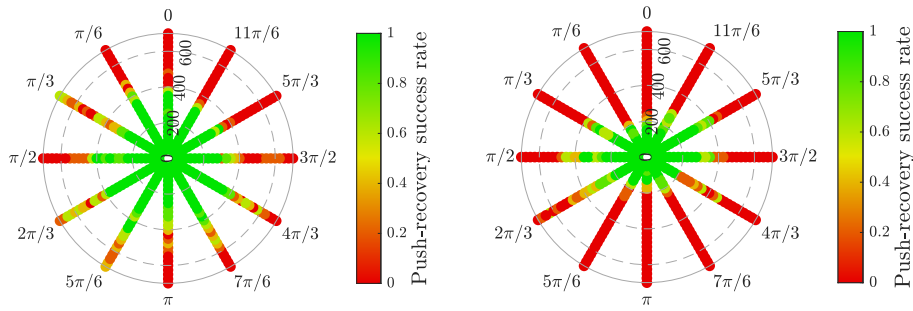


Figure 4.5. Push-recovery success rates for deterministic forces with magnitudes in [50, 700] N and directions spanning the horizontal plane (forward push: 0 rad) applied to the robot base link. The robot feet friction coefficient is $\mu_c = 1$ (left) or $\mu_c = 0.2$ (right).

Regarding contacts, using only DART (with its default collision detector and constraint solver) may have caused the policy to overfit to subtle implementation details. Modeling differences between physics engines³ notably make policies hardly transferable among them or the real world. Higher robustness could be achieved by randomizing the entire engine beyond the common physics parameters.

As for the absence of reference motion capture data in our reward (the postural \bar{s}_0 is fixed and only acts as a regularizer at steady-state, not during the transient dynamic recovery), our choice is motivated by two main reasons. First, our interest is to investigate the emergence of balancing and push-recovery strategies with no prior expert knowledge in the form of motion capture data exploited by other works [Peng et al., 2017, 2018] and understand whether interesting behaviours for such a challenging task can be learned from scratch. Second, the definition and collection of proper expert motions is not straightforward in our specific push-recovery scenario.

4.3.3 Push-recovery performances

We validate the robustness and generalization capabilities of the trained policy by performing an extensive quantitative analysis on the simulated iCub v2.7, including out-of-sample tasks of increasing complexity. The validation scenarios involving planar and spherical forces of different magnitude and duration, applied to the robot base link as well as to other links, are detailed in the following.

Deterministic planar forces

First, we evaluate the policy push-recovery performances in the event of *individual* deterministic *horizontal* forces. In this setting, forces of different horizontal direction and magnitude are applied to the robot base frame for the fixed duration used at training time, i.e., 200 ms, when the robot is stably standing still and front-facing, i.e., after 3 s from the beginning of the simulation. We define a test successful if the robot is still standing after 7 s. Figure 4.5 (left) reports the average success rates for forces spanning 12 directions in the horizontal plane, with magnitudes increasing from 50 N to 700 N at intervals of 25 N. For each combination of magnitude and direction, the average success rate over 5 tests is reported, starting at initial joint configurations randomized by adding zero-mean Gaussian noise ($\sigma = 2$ deg).

³Please refer to [Yoon et al., 2023] for a comparison of different physics engines for robotics, and to <https://leggedrobotics.github.io/SimBenchmark> for a more locomotion-oriented benchmarking.

The plot shows how forces with magnitudes within the training range of $[0, 200]$ N are always counteracted successfully. Remarkably, the policy is also robust to forces in all the directions with magnitudes in the out-of-sample range of $[200, 300]$ N, even up to 400 N in some directions. The very same validation illustrated in Figure 4.5 (left), where the Coulomb friction coefficient of the robot feet is set to $\mu_c = 1$, is repeated for an out-of-sample friction coefficient $\mu_c = 0.2$ and shown in Figure 4.5 (right). Also in such a challenging scenario, the policy is able to successfully recover from pushes with magnitudes in the training range of $[0, 200]$ N.

Random spherical forces on the base link

We further evaluate the robustness of the control policy in challenging scenarios involving *sequences* of random *spherical* forces with different combinations of magnitude and duration. In this setting, forces are applied to the robot base frame in a random spherical direction at a higher frequency with respect to training time, i.e., on average every 3 s rather than every 5 s. For each combination, 50 testing episodes are executed, with no domain randomization. Episodes terminate if the robot falls or after 60 s, with an average of 20 force applications for each successfully-completed episode. In this case, we select as evaluation metric the number of consecutive force applications that the policy manages to withstand.

Figure 4.6 (top) reports aggregate results for all the combinations of force magnitude and duration. In particular, forces of 5 different magnitudes in $[100, 300]$ N and 4 different durations in $[0.1, 0.4]$ s are considered. No matter their magnitude, forces lasting 0.1 s are properly balanced. As expected, performances decrease with growing magnitude and duration of the applied force. Nevertheless, the agent proves able to withstand repeated applications of out-of-sample forces. For instance, on average it withstands 9 consecutive applications of forces of 300 N lasting 0.2 s.

Random spherical forces on the chest and elbow links

Finally, we evaluate the robustness of the control policy to forces applied to robot links different than the base, i.e., in a previously-unseen scenario for the policy trained through force applications to the base link only. In particular, we repeat for the chest link and the elbow link the very same analysis illustrated in Figure 4.6 (top) for the robot base link, with the results averaged over 50 different trials for each considered testing scenario.

Figure 4.6 (middle) and (bottom) show the obtained results for the chest and elbow links, respectively. As expected, forces applied on links which are far from the robot CoM turn out to be more challenging to counterbalance. Nevertheless, the policy is able to generalize also in this case with good performances. For instance, it proves able to recover, on average, from 10 consecutive applications of forces of 200 N lasting 0.2 s on the elbow link, as opposed to an average of 17 for the base link. The average number of consecutive counterbalanced forces with the same magnitude and duration decreases to 5 for the chest link. Let us remark, however, that the randomness of the interval between two subsequent force applications considered for this analysis leads sometimes to extremely challenging scenarios in which multiple forces are applied in a very short time span.

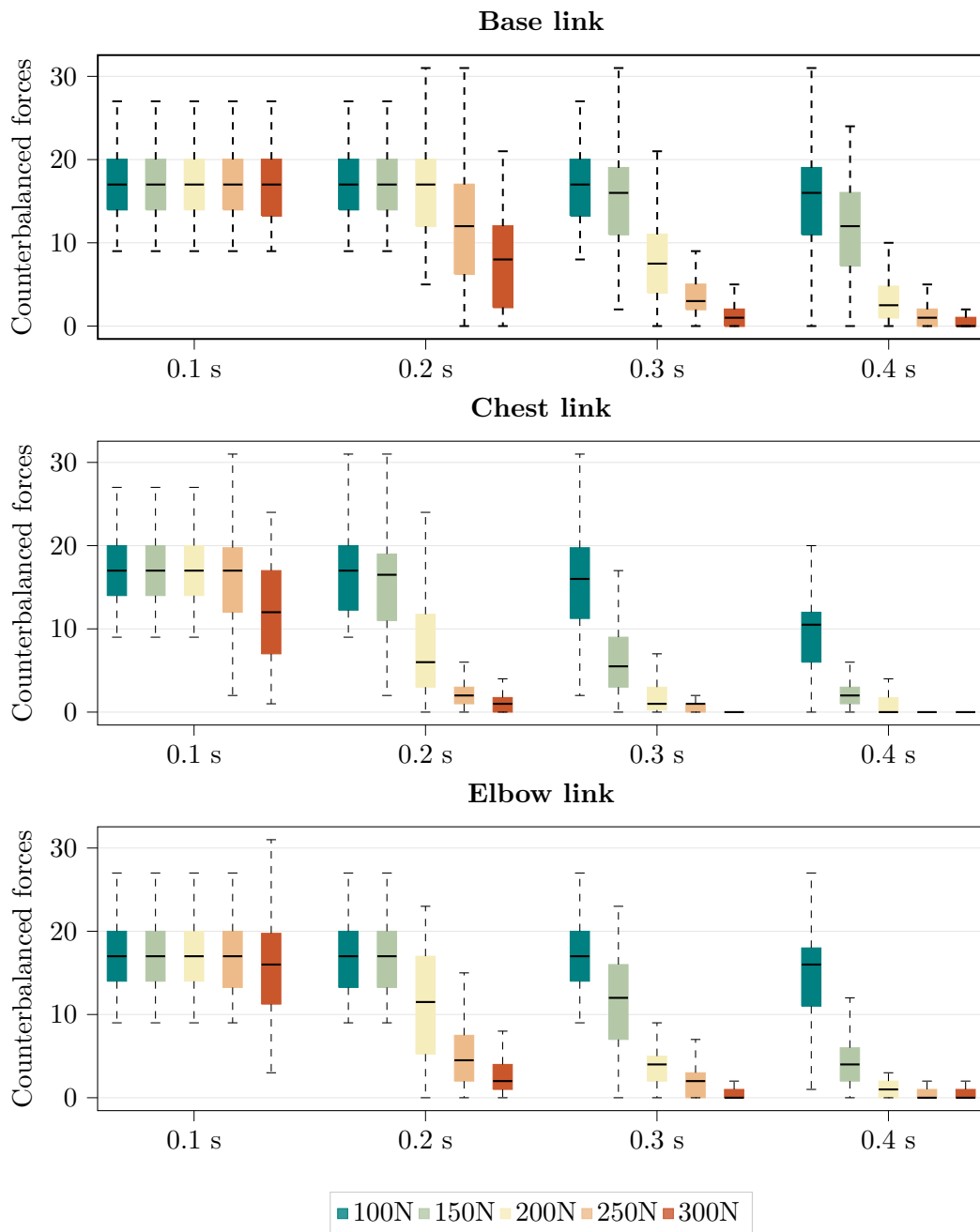


Figure 4.6. Consecutive counterbalanced forces in random spherical directions applied to the robot base (top), chest (middle) and elbow (bottom) links. Forces of 5 different magnitudes in $[100, 300]$ N and 4 different durations in $[0.1, 0.4]$ s are considered. For each combination of magnitude and duration, results are averaged over 50 trials.

4.4 Conclusions

In this chapter, we present a control architecture based on deep reinforcement learning to address whole-body balancing and push recovery for simulated humanoids. The proposed approach involves a state definition inspired by floating-base dynamics and an interpretable reward formulation shaped by domain knowledge in humanoid control. Compared to previous works, our agent controls a larger number of joints belonging to the robot legs, torso and arms, which enables it to extend the space of push-recovery motions to complex whole-body behaviours.

We validate the proposed control policy on the simulated iCub v2.7 humanoid, demonstrating the emergence of robust and highly-dynamic whole-body push-recovery strategies through which our policy is able to withstand repeated applications of strong external pushes.

The main limitation of the proposed approach is that the emerging recovery strategies, although successful in simulation, look often unnatural and unreliable. This is partially due to the stiffness introduced by the low-level position control and the modeling inaccuracies of the simulation, and could be mitigated in future works by considering alternative low-level controllers and more realistic actuator models, as well as by including in the domain randomization the entire physics engine, which would in turn enable an easier policy transfer to real robots.

The unnatural-looking behaviours learned by our control policy also arise from the absence of prior expert knowledge in the form of motion capture data driving the training process. While avoiding to shape the learned behaviours was a requirement for this work, focused on investigating the emergence of complex behaviours from scratch, in the next chapter we will examine the advantages of exploiting motion capture data with deep supervised learning to efficiently generate human-like walking motions for humanoid robots. Given the limitations of the approach proposed in this chapter, rather than directly extend it from the balancing to the walking task, we prefer indeed to address walking by deep supervised learning and take advantage of motion capture data.

Chapter 5

Learning Human-Like Whole-Body Trajectory Generators

In the previous chapter, we introduced a control architecture based on deep reinforcement learning capable of producing robust whole-body push-recovery strategies for simulated humanoids. Because of the inaccuracies of actuator modeling and simulated dynamics, along with the lack of regularizers driving the policy towards desired solutions, we did not achieve motions reliable enough to directly target sim-to-real transfer. Inspired by the performances demonstrated by character animation systems exploiting motion capture data to efficiently generate responsive natural-looking behaviours [Holden et al., 2017; Zhang et al., 2018; Starke et al., 2021], in this chapter we target the problem of general and *human-like* whole-body trajectory generation for humanoid robots via efficient learning-based techniques proved successful in computer graphics.

To this purpose, we collect a wide-ranging dataset of motion capture locomotion trajectories, properly retargeted from the human subject to the robot, and exploit deep supervised learning to train a whole-body trajectory generator. Our solution, kinematically validated on the iCub v2.7 humanoid model, efficiently produces diverse walking patterns and smooth transitions among them that state-of-the-art methods can only compute offline due to high computational cost. Moreover, the motions thus generated exhibit a certain degree of human-likeness both at the joint and footstep levels. Therefore, they represent a valid source of footstep plans and postural references for whole-body humanoid locomotion controllers to obtain human-like real-world robot motions, as detailed in the next chapter.

More in detail, the chapter is organized as follows. Section 5.1 introduces state-of-the-art techniques for motion retargeting and responsive motion synthesis. Details on the motion capture dataset collected for this work are provided in Section 5.2, while the methodology followed to retrieve from it kinematically-feasible robot motions is presented in Section 5.3. The learning-based architecture trained to interactively generate whole-body trajectories is illustrated in Section 5.4 and kinematically validated, also in terms of human-likeness of the generated motion, in Section 5.5. Finally, Section 5.6 concludes the chapter.

The content of this chapter partially appears in:

Viceconte, P. M., Camoriano, R., Romualdi, G., Ferigo, D., Dafarra, S., Traversaro, S., Oriolo, G., Rosasco, L., and Pucci, D. (2022). ADHERENT: Learning Human-like Trajectory Generators for Whole-body Control of Humanoid Robots. *IEEE Robotics and Automation Letters*, 7(2):2779–2786.

Video <https://www.youtube.com/watch?v=s7-pML0ojK8>

Github [ami-iit/paper_viceconte_2021_ral_adherent](https://github.com/ami-iit/paper_viceconte_2021_ral_adherent)

Zenodo https://zenodo.org/record/6201915#.Yh0T_Bso-8g

5.1 Background

Before illustrating the details of the learning-based trajectory generation pipeline proposed in this work, let us briefly introduce two state-of-the-art approaches for retargeting and interactive character animation on which we will build upon in the following sections, i.e., Whole-Body Geometric Retargeting and Mode-Adaptive Neural Networks, respectively.

5.1.1 Whole-body Geometric Retargeting

Given a robotic platform and a series of movements performed by a human subject, the goal of *human-robot motion retargeting* is to transfer the human motion onto the robot in such a way to obtain an anthropomorphic robot motion, i.e., a robot motion that mimics as much as possible the original motion performed by the human. The substantial dissimilarities in the mechanical structures characterizing the human and the robot prevent a direct mapping of the human motion onto the robot, making human-robot motion retargeting a non-trivial challenge.

Among the various approaches addressing human-robot motion retargeting in the literature, e.g., [Pollard et al., 2002; Ott et al., 2008; Miura et al., 2009; Penco et al., 2018]), Whole-Body Geometric Retargeting (WBGR) is a recently-proposed method relying on geometric mapping between the human and the robot links and inverse kinematics, which easily adapts to different robot models and human subjects [Darvish et al., 2019].

Assuming a certain degree of topological similarity between the human’s and robot’s mechanical structures, WBGR builds upon the definition of a set of correspondences between the frames associated with m human and robot links. Such correspondences for the iCub v2.7 humanoid – see Section 3.4.1 – are shown in Figure 5.1, where numbering highlights correspondent frames. This *geometric* mapping between human (\mathcal{H}) and robot (\mathcal{R}) link frames is complemented by the constant rotations ${}^{\mathcal{H}}R_{\mathcal{R}}^i$, with $i \in \{1, \dots, m\}$, accounting for possible human-robot link frame misalignments, manually identified by comparing the human and robot models in a reference configuration such as the *T-pose* shown in Figure 5.1.

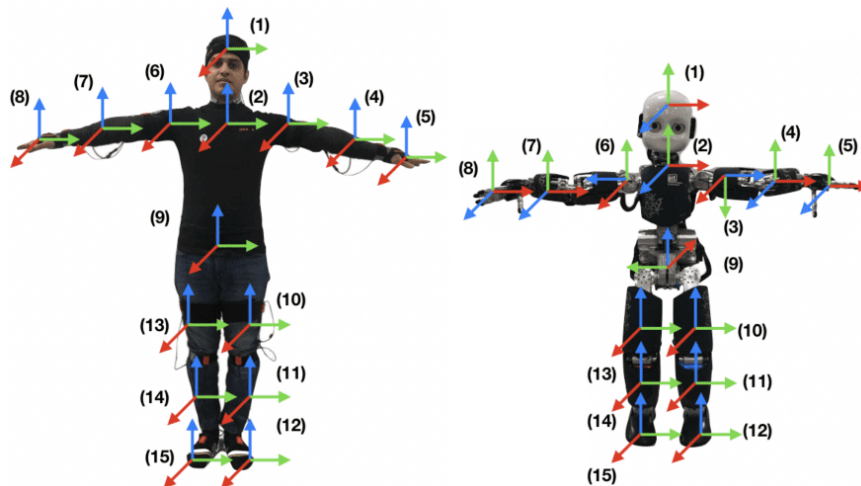


Figure 5.1. Human-robot frame correspondences exploited by WBGR. Frames with the same number are associated with each other, although they can have a different orientation when compared in a reference configuration such as the *T-pose* shown here.

Then, given the human link orientations ${}^{\mathcal{I}}R_{\mathcal{H}}^i$, with $i \in \{1, \dots, m\}$, defining at each time instant the pose of the human to be retargeted onto the robot, WBGR retrieves the robot joint angles realizing a pose which resembles the human’s one by solving an inverse kinematics optimization [Sciavicco and Siciliano, 1998; Latella et al., 2019] where the target robot link orientations are given by

$${}^{\mathcal{I}}R_{\mathcal{R}}^i = {}^{\mathcal{I}}R_{\mathcal{H}}^i {}^{\mathcal{H}}R_{\mathcal{R}}^i. \quad (5.1.1)$$

The inverse kinematics formulation, optionally benefiting from a dynamical optimization method that ensures the convergence of the frame orientation errors to a minimum [Rapetti et al., 2020], is then expressed as a quadratic programming¹ (QP) problem and solved using off-the-shelf solvers, e.g., [Stellato et al., 2020].

Notice that, thanks to the geometric approach which only requires the identification of suitable correspondences between the human and the robot links and the computation of fixed rotations accounting for their misalignments, WBGR allows for changes of human subject or robotic platform with minimal effort, proving increased scalability with respect to concurrent state-of-the-art retargeting methods.

5.1.2 Mode-Adaptive Neural Networks

Among the wide variety of methods for responsive and realistic character animation proposed in the computer graphics literature presented in Section 3.2, and in particular within the category of *kinematic* motion synthesis methods illustrated in Section 3.2.1, Mode-Adaptive Neural Networks (MANN) [Zhang et al., 2018] is a recently-proposed neural network architecture specifically designed for *multi-modal* and unlabeled data, built upon the Mixture of Experts (MoE) paradigm [Nowlan and Hinton, 1990; Jacobs et al., 1991; Jordan and Jacobs, 1994].

¹The definition of a quadratic programming (QP) problem and its close form solution are given in Eq. (6.1.10) and Eq. (6.1.11), respectively.

As other interactive kinematic motion synthesis approaches exploiting motion capture data, MANN frames the character animation problem as the *kinematic prediction* of the whole-body character configuration in the next animation step, given the character configuration in the current animation step and some *high-level target* specification, in the form of a future trajectory to be followed, provided by the user interactively commanding the motion synthesis. This problem, modeled as a nonlinear autoregressive model with exogenous inputs, is then solved by employing powerful learning-based predictive architectures able to capture the complexity of the character motion in high dimensions. In particular, MANN proposes a peculiar network architecture composed of two fully-connected feedforward neural networks:

- the *Motion Prediction Network* (MPN), in charge of predicting the next user-compatible character configuration;
- the *Gating Network* (GN), producing the *blending coefficients* used to compute the weights of the MPN dynamically.

Let us indicate with the subscript \cdot_i quantities related to the i -th animation step t_i . Assume $x_i \in \mathbb{R}^n$ to encode the current character configuration as well as the desired future motion specified by the user. The MPN, whose weights are denoted by α , predicts the vector $y_i \in \mathbb{R}^m$ encoding the next character configuration, compatible with the user-specified motion, along with other features. Proposed in the form of a three-layer neural network with ELU activation function – see Eq. (2.1.6) – for the hidden units and a linear output unit, it implements the operation

$$y_i = \Theta(x_i|\alpha) = W_2 \text{ELU}(W_1 \text{ELU}(W_0 x_i + b_0) + b_1) + b_2, \quad (5.1.2)$$

where the MPN weights α are defined as

$$\alpha = \{W_0 \in \mathbb{R}^{h \times n}, W_1 \in \mathbb{R}^{h \times h}, W_2 \in \mathbb{R}^{m \times h}, b_0 \in \mathbb{R}^h, b_1 \in \mathbb{R}^h, b_2 \in \mathbb{R}^m\}, \quad (5.1.3)$$

and h is the width of the MPN hidden layers.

Given $x_i \in \mathbb{R}^n$ or, more often, a subset $\hat{x}_i \in \mathbb{R}^{n'}$, with $n' < n$, the GN, whose weights are denoted by μ , produces the blending coefficients $\theta_i \in \mathbb{R}^K$ used to dynamically compute the MPN weights α_i as follows. Let us introduce the softmax activation function ρ defined component-wise on its input $z \in \mathbb{R}^K$ as

$$\rho(z)_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}, \quad k \in \{1, \dots, K\}. \quad (5.1.4)$$

Proposed as a three-layer neural network with ELU and softmax activation functions for the hidden and output units, respectively, the GN implements the operation

$$\theta_i = \Omega(\hat{x}_i|\mu) = \rho(W_2' \text{ELU}(W_1' \text{ELU}(W_0' \hat{x}_i + b_0') + b_1') + b_2'), \quad (5.1.5)$$

where the GN weights μ are defined as

$$\mu = \{W_0' \in \mathbb{R}^{h' \times n}, W_1' \in \mathbb{R}^{h' \times h'}, W_2' \in \mathbb{R}^{K \times h'}, b_0' \in \mathbb{R}^{h'}, b_1' \in \mathbb{R}^{h'}, b_2' \in \mathbb{R}^K\}, \quad (5.1.6)$$

and h' is the width of the GN hidden layers.

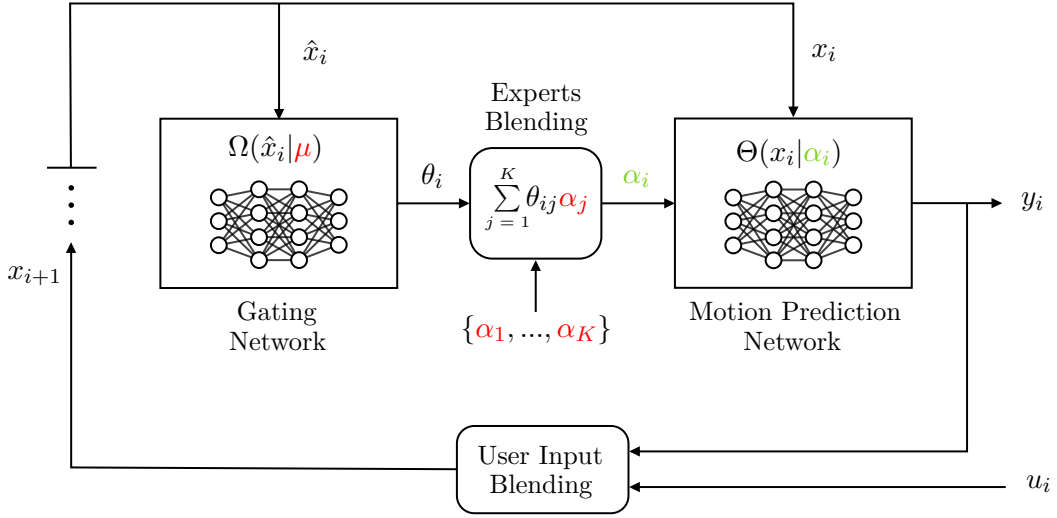


Figure 5.2. Sketch of the autoregressive MANN architecture, composed of the GN with parameters μ , the K expert weights $\{\alpha_1, \dots, \alpha_K\}$ and the MPN with parameters α_i . The parameters highlighted in red are learned during training and fixed at inference time, while those highlighted in green are dynamically recomputed at each inference step. The GN input \hat{x}_i either coincides with the MPN input x_i or is a subset of its components.

As already mentioned, the blending coefficients θ_i outputted by the GN are used to dynamically compute the weights α of the MPN. This is implemented by actually maintaining K copies of the MPN weights α , i.e., K expert weights $\{\alpha_1, \dots, \alpha_K\}$. At training time, both the GN weights μ and the K expert weights $\{\alpha_1, \dots, \alpha_K\}$ are simultaneously learned. At runtime, the MPN weights α_i are dynamically computed by linearly combining the K experts $\{\alpha_1, \dots, \alpha_K\}$ with the blending coefficients θ_i produced by the GN, that is,

$$\alpha_i = \sum_{j=1}^K \theta_{ij} \alpha_j, \quad (5.1.7)$$

where θ_{ij} is the j -th blending coefficient at time i , i.e., $\theta_i = [\theta_{i1}, \dots, \theta_{iK}]^\top$.

The whole MANN architecture is summarized in Figure 5.2. At each animation step, the GN and the MPN are fed with the subsampled input \hat{x}_i and the full input x_i , respectively. The GN outputs the blending coefficients θ_i from Eq. (5.1.5), used as prescribed by Eq. (5.1.7) to dynamically compute the MPN weights α_i . Notice how such a "mixture-of-weights" approach can be equivalently interpreted as a mixture-of-experts (MoE) whose blending occurs at the features level rather than at the final output layer (as in conventional MoE). Finally, the MPN weights α_i are exploited by the MPN to retrieve the output y_i from Eq. (5.1.2).

The next character configuration included in y_i is then used to update the character state and let the animation proceed. In an autoregressive fashion, the output y_i is also combined with the next high-level target u_i from the user to form the next MANN input x_{i+1} . Such a mechanism enables MANN, once trained for a typical regression task on a dataset of suitable I/O features $\{x_i, y_i\}$ extracted from motion capture data, to efficiently generate trajectories which accurately resemble the reference data while being highly responsive to the user-specified commands.

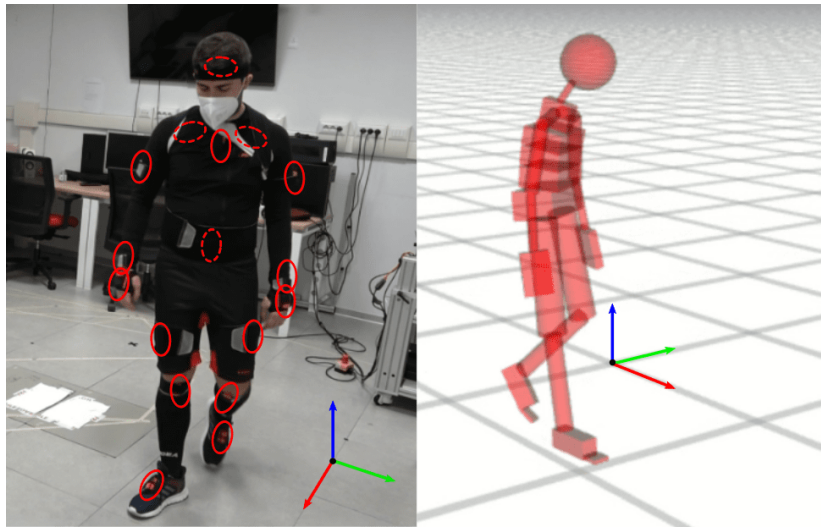


Figure 5.3. The human subject equipped with the XSens sensorized suit during the motion capture data collection, along with its 48-DoFs associated model composed by 23 simple geometrical shapes. The locations of the 17 IMUs spanning the human body (some of which are covered by elastic bands) are highlighted in red, with dashed lines indicating sensors placed on the back.

5.2 Dataset collection

To collect human locomotion reference trajectories in the form of motion capture data, we take advantage of the human wearable data processing framework from [Latella, 2019; Rapetti et al., 2020] that fuses data from a sensorized suit by XSens technologies [Roetenberg et al., 2009].

The whole-body motion tracking suit by XSens consists of a set of 17 Inertial Measurement Units (IMUs) distributed all over the human body, providing kinematic and dynamic quantities relevant for estimating the human motion at a rate of 60 Hz. Figure 5.3 (left) shows the human subject wearing the XSens suit during the acquisition of motion capture data.

Measurements from the XSens suit are exploited to reconstruct the human motion on a simplified model of the human [Latella, 2019; Tirupachuri, 2020], extended from the one that comes with the XSens suit [Roetenberg et al., 2009] and shown in Figure 5.3 (right). In our case, the human is modeled with a set of $n_l = 23$ rigid bodies as links, characterized by basic geometrical shapes such as cylinders, spheres and parallelepiped boxes with uniform and isotropic density. In this simplified human model, most of the links are connected with each other by three subsequent revolute joints ensuring 3 degrees of freedom (DoFs) per joint, which would lead to a total of $3(n_L - 1) = 66$ DoFs. However, according to the properties of human biological joints, a few joints in the simplified human model are connected by 1 or 2 revolute joints only, with the total number of DoFs actually dropping to 48 DoFs. Despite being far from capturing the real biomechanics of the human body, much more complex due to the combined action of muscles, tendons and sophisticated articulations, such a representation allows for sufficiently-accurate reference locomotion trajectories for the purpose of this work.

Table 5.1. Breakdown of the motion capture dataset collected at 60 Hz.

Walking motion	Duration [min]	Datapoints	Stops
Forward	8.2	29500	25
Backward	9	32450	25
Side	9.45	34000	27
Diagonal	4.2	15125	16
Mixed	30.48	109710	75

5.2.1 Motion capture dataset

Our motion capture dataset spans a wide range of walking motions (forward, backward, lateral, and diagonal) performed on a flat terrain while continuously changing the steering direction. The collected sequences, characterized by footsteps of variable length and also including several stops and restarts, are detailed in Table 5.1. We perform our dataset collection by first acquiring data for several minutes in a row for each individual walking pattern. This is to sufficiently cover each target motion and produce an overall balanced dataset. Then, a 30-minutes long mixed sequence involving all the walking patterns is collected. This is to include enough information about the transitions between the different motions, crucial for the trajectory generator to learn good blending capabilities.

Our final dataset comprises around 1 hour of unlabeled motion capture data at 60 Hz. In order to compensate for asymmetries we finally double our dataset by mirroring. In particular, for each datapoint the base orientation is mirrored with respect to the world X-Z plane, while the link orientations for the left and right limbs are switched and mirrored with respect to the model’s mid-sagittal plane. After mirroring, we obtain a motion capture dataset of 441570 datapoints, that is publicly available on Zenodo at https://zenodo.org/record/6201915#.Yh0T_Bso-8g.

5.3 Retargeting

For our purposes, the reference locomotion trajectories collected from the human need to be retargeted onto the robot. We deal with this issue by exploiting the Whole-Body Geometric Retargeting (WBGR) technique introduced in Section 5.1.1. WBGR, however, does not address retargeting for the *base motion*, and simply uses for the robot base the reference motions collected for the human base. Nevertheless, depending on the human and robot mechanical structures, such motions may not be compatible with the robot kinematics. As a result, a swaying effect arises when dynamic motions are retargeted via WBGR to robot models with remarkable structural differences with respect to the human subject [Darvish et al., 2019]. For instance, the robot moves faster than what its walking pace entails.

While in computer graphics generating models which perfectly fit the collected data is a viable workaround [Zhang et al., 2018; Bergamin et al., 2019], base motion retargeting for actual robots requires special attention. This is why we complement WBGR with a *kinematically-feasible* base motion retargeting procedure which renders the robot base motion compatible with the retargeted robot joint trajectories.

5.3.1 Kinematically-feasible base motion retargeting

In order to obtain kinematically-feasible robot base motions compatible with the motion of the other links retrieved via WBGR, we implement the following procedure. First, we assume that:

1. The robot makes at least one known contact with the environment at each retargeting step.
2. Each robot foot is modeled as a rectangular patch.

Notice that the first assumption prevents the application of our kinematically-feasible base motion retargeting procedure to motions which involve phases with no contact with the external environment, such as running or jumping. Despite being a strong assumption, it cannot be relaxed in the proposed approach. The second assumption, instead, is made for computational reasons. It allows indeed to evaluate contacts between the robot feet and the ground by simply considering a discrete set of points, i.e., the vertices of the rectangular patch. Without this assumption, contacts evaluation would be more tricky, especially for complex feet shapes such as the rounded shape of the iCub v2.7 feet – see Section 3.4.1.

Given the above assumptions, we retrieve kinematically-feasible base motions that are compatible with the retargeted motions of the other robot links by means of the following procedure:

1. The contact point ${}^{\mathcal{I}}p_c$ between the robot and the ground is identified as the *lowest vertex* among the eight vertices of the robot feet’s rectangular patches;
2. The retargeted base orientation ${}^{\mathcal{I}}R_B$ is directly maintained from the human motion capture data (as in WBGR);
3. The *kinematically-feasible* retargeted base position ${}^{\mathcal{I}}p_b$ is computed by forward kinematics from the selected contact point ${}^{\mathcal{I}}p_c$, constrained to remain *fixed* between two consecutive retargeting steps.

In particular, the kinematically-feasible retargeted base position ${}^{\mathcal{I}}p_b$ is given by

$${}^{\mathcal{I}}p_b = {}^{\mathcal{I}}p_c + {}^{\mathcal{I}}R_{\mathcal{C}[\mathcal{F}]}{}^{\mathcal{C}[\mathcal{F}]}p_b, \quad (5.3.1)$$

where $\mathcal{C}[\mathcal{F}]$ is the mixed reference frame – see Section 1.1.2 – having its origin in the lowest vertex selected as contact point and the same orientation as the frame \mathcal{F} attached to the foot the lowest vertex belongs to, i.e., the support foot. Accordingly, in Eq. (5.3.1) ${}^{\mathcal{C}[\mathcal{F}]}p_b$ denotes the robot base position, expressed in $\mathcal{C}[\mathcal{F}]$, computed by forward kinematics from the known contact point in the joint configuration returned by the latest WBGR iteration.

As a result, we obtain retargeted motions for the robot that resemble the human ones at the links level and are also kinematically-feasible at the base level, being therefore suitable for the extraction of the network I/O features described in the next section. A complete overview of the final dataset, retargeted onto the iCub v2.7 humanoid and including the forward, backward, right-side, left-side, diagonal and mixed motions described in Section 5.2.1, is provided in a dedicated portion of the accompanying video available at <https://youtu.be/s7-pML0ojK8?t=202>.

5.4 Trajectory generation

For the interactive generation of data-driven trajectories for the humanoid robot, we exploit the Mode-Adaptive Neural Network (MANN) architecture outlined in Section 5.1.2. Our MANN implementation, inspired by the original one [Zhang et al., 2018], consists of a Motion Prediction Network (MPN) and a Gating Network (GN) both composed of three hidden layers with $h = 512$ and $h' = 32$ units each, respectively. We employ the ELU activation function from Eq. (2.1.6) for both the networks. We use $K = 4$ sets of expert weights.

The exact definition of the input x_i and output y_i for our MANN implementation is given in Section 5.4.1. Motivated by empirical results, we use the full input x_i rather than a reduced input \hat{x}_i to feed the GN. With reference to the MANN architecture shown in Figure 5.2, the processing of the exogenous input u_i from the user and its blending with the MANN output y_i is illustrated in Section 5.4.2.

5.4.1 Features extraction

In order to generate trajectories for the robot, we would like our MANN implementation to output at each generation step the free-floating system configuration q , defined in Eq. (1.2.17), and the free-floating system velocity ν , defined for instance in mixed representation in Eq. (1.2.16). Moreover, to allow for an interactive generation, we would like the network input to include a description of the desired future motion that an external user can influence to drive the process.

Inspired by previous work [Zhang et al., 2018; Holden et al., 2017], we decide to let our network predict only a *reduced* configuration of the free-floating system. In particular, our network predicts the robot *joint state* $\{s, \dot{s}\}$ and the variation $\dot{\psi}$ of the robot *base yaw* angle ψ , ignoring instead the base linear motion and the base roll and pitch angles. The base roll and pitch are simply constrained to zero. While walking, these angles remain anyways close to zero and therefore this seems a reasonable simplification for the task to be learned. On the contrary, the base yaw needs to remain free to let the user command steering motions. As regards instead the base linear motion, we apply to the network output the very same procedure used to ensure kinematic feasibility at retargeting stage (see Section 5.3.1). This procedure computes indeed a base linear motion compatible with the joint motion predicted by the network but also kinematically-feasible, i.e. such that the contact point between the robot foot and the ground remains fixed and, consequently, no sliding nor similar artifacts affect the generated motion.

To make the generation interactive, again inspired by previous work [Zhang et al., 2018; Holden et al., 2017], we encode the information related to the desired robot motion via the motion of its base projected on the ground. The user input u_i is indeed relatively simple to express as a desired ground-projected trajectory of the robot base. By including in y_i a component representing the network prediction for the future ground-projected evolution of the robot base, we can then blend it with u_i and exploit it in an autoregressive fashion to build the next input x_{i+1} . Blending is essential to ensure smooth changes in x_i , which would instead be missing by directly including u_i in x_i . Further details and visualization of both the features (Figure 5.4) and the user input processing (Figure 5.5) are provided in the following.

Before illustrating the definitions of our MANN input x_i and output y_i resulting from the above considerations, let us introduce the *facing direction* as the bidimensional vector given by the normalized ground-projected mean between the *frontal* robot base and chest directions. Notice that the definition of the frontal robot base and chest directions depends, in turn, on the definition of the robot base \mathcal{B} and chest \mathcal{T} frames. For instance, in the case of the iCub v2.7 humanoid, \mathcal{B} and \mathcal{T} are the frames denoted by (9) and (2) in Figure 5.1, respectively. Therefore, the frontal base direction coincides with the negative x axis of \mathcal{B} while the frontal chest direction with the positive z axis of \mathcal{T} . Their mean, projected on the ground and normalized, defines the facing direction for such a robot.

Let us also introduce the bidimensional *local* reference frame \mathcal{L} in which we assume all the quantities related to the ground-projected base trajectory in x_i and y_i to be expressed. At each step t_i , \mathcal{L} is defined to have its origin in the current ground-projected robot base position and orientation defined by the current facing direction (along with its orthogonal vector). \mathcal{L} is shown, for instance, in the frame at $t = t_1 + 1.5s$ of Figure 5.4. Notice the importance of expressing the features in \mathcal{L} , i.e., locally. Consider for instance two forward walking motions in different global directions. Only by expressing the correspondent features locally, such motions would appear to the network in a similar form and could be recognized as instances of the same task, i.e., walking forward, which is independent from the global direction.

Input features

The input vector x_i includes the robot *joint state* at t_{i-1} and the *ground-projected base* trajectory data at t_i . We define the ground-projected base trajectory data, i.e., past and future data about the robot base trajectory projected on the ground, over a time window of 2 s centered at t_i . In particular, we subsample $k = 2r = 12$ datapoints equally-spaced over the considered window. As a result, x_i is defined as

$$x_i = \underbrace{\{\text{vec}(P_i), \text{vec}(D_i), \text{vec}(V_i), l_i\}}_{\text{Ground-projected base trajectory data}}, \underbrace{\{s_{i-1}, \dot{s}_{i-1}\}}_{\text{Joint state}} \in \mathbb{R}^{137}, \quad (5.4.1)$$

where the $\text{vec}(\cdot)$ operator vectorizes matrices by rows and

- $P_i = \{p_i^1, \dots, p_i^k\} \in \mathbb{R}^{2 \times k}$ are ground-projected base positions (among which p_i^r is the current ground-projected base position), expressed in \mathcal{L} ;
- $D_i = \{d_i^1, \dots, d_i^k\} \in \mathbb{R}^{2 \times k}$ are facing directions (among which d_i^r is the current facing direction), expressed in \mathcal{L} ;
- $V_i = \{v_i^1, \dots, v_i^k\} \in \mathbb{R}^{2 \times k}$ are ground-projected base velocities (among which v_i^r is the current ground-projected base velocity), obtained in the training data by differentiation of P_i , expressed in \mathcal{L} ;
- $l_i = \sum_{j=r+1}^k \|p_i^j - p_i^{j-1}\| \in \mathbb{R}$ is the length of the future ground trajectory,
- $s_{i-1} \in \mathbb{R}^{32}$ and $\dot{s}_{i-1} \in \mathbb{R}^{32}$ are the joint positions and velocities at t_{i-1} .

The ground-projected base positions P_i and facing directions D_i included in the network input x_i are shown, for different time instants during a forward walking motion with multiple turns, in Figure 5.4.

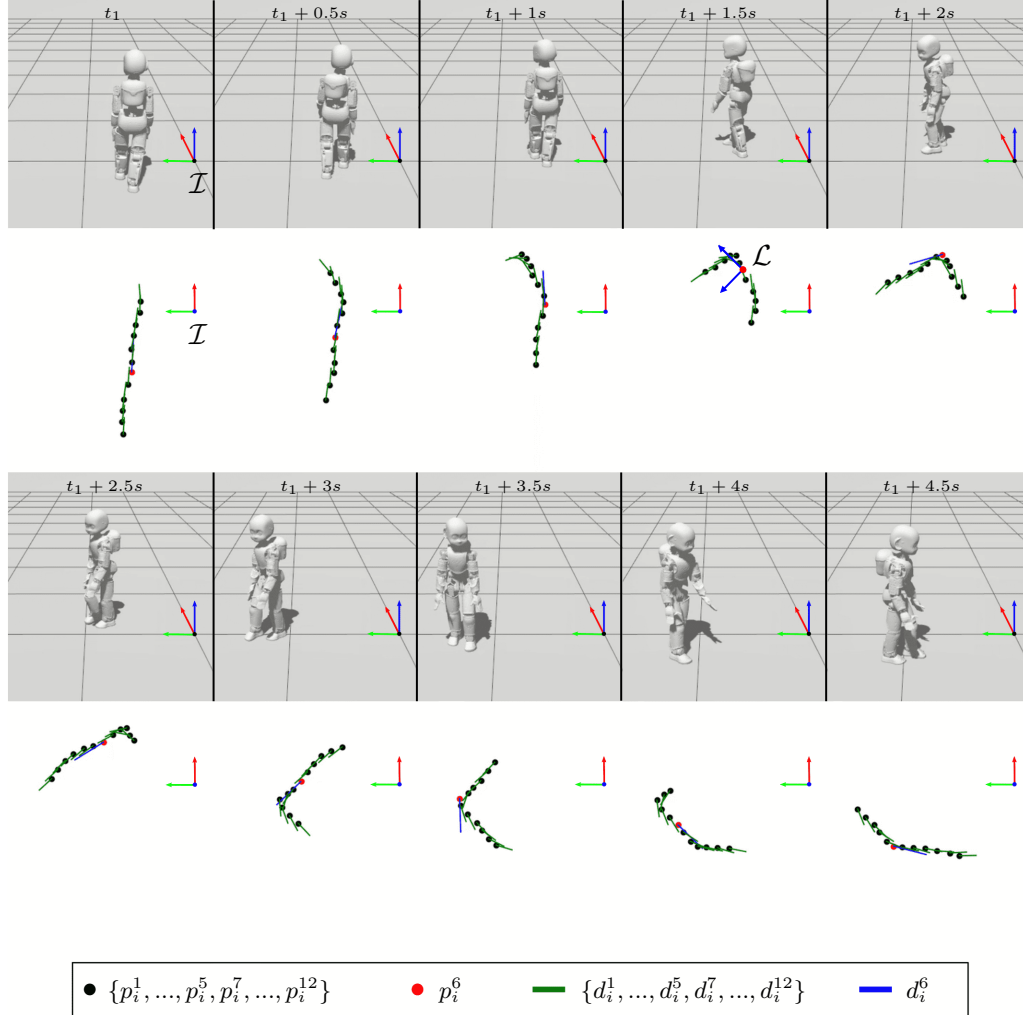


Figure 5.4. Ground-projected base positions $P_i = \{p_i^1, \dots, p_i^k\}$ and facing directions $D_i = \{d_i^1, \dots, d_i^k\}$, with $k = 2r = 12$, at sequential time instants t_i of a forward walking with multiple turns, distant 0.5 s from each other. At each t_i , the top view below the robot visualization shows the correspondent P_i (black dots, with the current ground-projected position p_i^6 in red) and D_i (green arrows, with the current facing direction d_i^6 in blue and longer) spanning the considered 2 s time window. For the sake of clarity, P_i and D_i are visualized here expressed in the global frame \mathcal{I} , but those actually included in x_i are expressed in the local frame \mathcal{L} , shown explicitly at $t_1 + 1.5s$. Notice how, at time t_i , P_i and D_i actually encode information about 1s in the past and 1s in the future, since they store the past ground-projected base trajectory up to $t_i - 1s$ and anticipate the future ground-projected base trajectory until $t_i + 1s$. For instance, at $t_1 + 1s$, the first black dot p_{1+1s}^1 is almost coincident with the central red dot p_1^6 at t_1 , while the last black dot p_{1+1s}^{12} is almost coincident with the central red dot p_{1+2s}^6 at $t_1 + 2s$.

Output features

The output vector y_i includes the *joint state* at t_i , the *base yaw* variation from t_{i-1} to t_i , and the *future ground-projected base trajectory* data at t_{i+1} (i.e., $r = k/2 = 6$ datapoints equally-spaced in a 1s window starting at t_i). As a result, y_i is defined as

$$y_i = \left\{ \underbrace{\text{vec}(P_{i+1}), \text{vec}(D_{i+1}), \text{vec}(V_{i+1})}_{\text{Future ground-projected base trajectory data}}, \underbrace{s_i, \dot{s}_i}_{\text{Joint state}}, \underbrace{\dot{b}_i^a}_{\text{Base yaw variation}} \right\} \in \mathbb{R}^{101}, \quad (5.4.2)$$

where the $\text{vec}(\cdot)$ operator vectorizes matrices by rows and

- $P_{i+1} = \{p_{i+1}^1, \dots, p_{i+1}^r\} \in \mathbb{R}^{2 \times r}$ are future ground-projected base positions;
- $D_{i+1} = \{d_{i+1}^1, \dots, d_{i+1}^r\} \in \mathbb{R}^{2 \times r}$ are future facing directions;
- $V_{i+1} = \{v_{i+1}^1, \dots, v_{i+1}^r\} \in \mathbb{R}^{2 \times r}$ are future ground-projected base velocities;
- $s_i \in \mathbb{R}^{32}$ and $\dot{s}_i \in \mathbb{R}^{32}$ are the joint positions and velocities at t_i ;
- $\dot{b}_i^a = \beta_i / \Delta t_i \in \mathbb{R}$ is the base yaw variation, with $\Delta t_i = t_i - t_{i-1}$, and β_i denoting the angle between the current facing directions d_i^r at t_i and d_{i-1}^r at t_{i-1} .

The network output y_i is used to update the robot configuration q . In particular, s_i becomes the new joint configuration and \dot{b}_i^a is exploited to update the base yaw, obtaining therefore the updated base orientation ${}^{\mathcal{I}}R_B$.

We observe that, when the user tries to stop the robot by releasing the analog for the motion direction (see Section 5.4.2), a small in-place rotation persists. Indeed, given an x_i corresponding to a desired stop, the network predicts a y_i whose \dot{b}_i^a component is slightly different from zero. To mitigate this issue, we impose $\dot{b}_i^a = 0$ once a desired stop for the robot is detected. Notice that we are referring here to desired stops at the network level, which can also occur several time instants after the user releases the joystick analog. We detect such stops by searching for almost-identical consecutive network outputs. In particular, a stop is detected if $\|y_i - y_{i-1}\| < \tau_{stop}$, with $\tau_{stop} = 0.05$ from empirical considerations.

Moreover, let us recall that y_i contains no information on the linear motion of the robot base. The updated base position ${}^{\mathcal{I}}p_b$ is indeed computed by applying the very same procedure ensuring kinematic feasibility at the retargeting stage (see Section 5.3.1). Finally, the future ground-projected base trajectory data $\{P_{i+1}, D_{i+1}, V_{i+1}\}$ included in y_i is blended with the user input u_i as described in the next section.

5.4.2 User input processing

At inference time, the user provides via the analogs of a joystick two continuous signals to interactively shape the robot trajectories:

- The desired *motion direction*, i.e., the direction in which the user wants the robot to move;
- The desired *facing direction*, i.e., the direction towards which the user wants the robot to align its ground-projected mean of base and torso frontal directions.

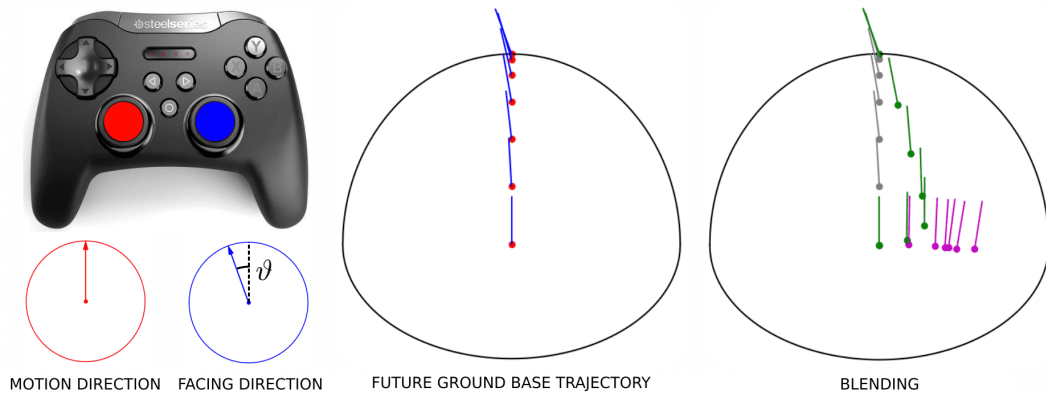


Figure 5.5. (Left) The desired motion and facing directions provided by the user through the joystick analogs highlighted with the correspondent colors. (Center) The desired future ground-projected base trajectory $u_i = \{P_{i+1}^*, D_{i+1}^*, V_{i+1}^*\}$ retrieved from the user input on the left. P_{i+1}^* are the red dots and D_{i+1}^* the blue vectors, while V_{i+1}^* are not visualized for the sake of clarity. (Right) The user-specified desired future trajectory u_i (grey) is blended with the future trajectory included in the previous network prediction y_i (magenta), leading to the future ground-projected base trajectory actually included in the next network input x_{i+1} (green).

At fixed desired facing direction, varying the desired motion direction allows to switch between frontal, sideways, and backward walking. At fixed desired motion direction, varying the desired facing direction enables steering. Moreover, releasing the analog for the motion direction leads the robot to a stop. The user inputs are visualized in Figure 5.5 (left), from the local viewpoint of a robot which is commanded to proceed forward while steering left.

Notice that the user-specified motion and facing directions may result in a desired robot motion which is absent or extremely rare in the training dataset. In this case, the network may generate *unexpected motions*. In particular, we observe such an issue arising if the user requests the robot to steer too abruptly. With reference to Figure 5.5 (left), we decide therefore to enforce a constraint on the angle ϑ between the nominal forward facing direction and the desired facing direction. Specifically, ϑ is constrained to remain in $[-\pi/4, \pi/4]$ rad during forward walking and in $[-\pi/9, \pi/9]$ rad during backward and sideways walking, where the acceptable intervals are found empirically. Preventing too abrupt motions by constraining ϑ , despite representing a limitation on the trajectories that can be commanded by the user, is essential to keep the motion generation consistent.

Moreover, an accurate processing of the user-specified inputs is critical for the predictive performances of MANN. Given the desired facing and motion directions, such a processing involves two main steps:

1. The definition of a desired future ground-projected base trajectory u_i from the user-specified inputs;
2. The blending of u_i with the future ground-projected base trajectory included in the network prediction y_i .

As regards the first step, we smoothly interpolate the two signals from the user to generate a desired future ground-projected base trajectory $u_i = \{P_{i+1}^*, D_{i+1}^*, V_{i+1}^*\}$, whose components are defined as in Eq. (5.4.2).

In particular, the desired motion direction is used to define the last point of a quadratic Bézier curve starting from the current ground-projected base position p_i^r and constrained to end on the asymmetric shape shown in black in Figure 5.5, composed of two experimentally-found semi-ellipses with same horizontal axis of 0.35 m, and vertical axes of 0.4 m and 0.25 m for the upper and lower semi-ellipse, respectively. We obtain P_{i+1}^* by subsampling $r = k/2 = 6$ datapoints from this Bézier curve. As a result of the asymmetric constraint, P_{i+1}^* is longer for forward rather than sideways or backward walking. The user-specified facing direction is instead mapped into a series of facing directions D_{i+1}^* progressively driving the current facing direction (always forward in the local robot perspective) to the desired one. Finally, V_{i+1}^* is obtained by differentiating P_{i+1}^* . Figure 5.5 (center) provides a visualization of P_{i+1}^* (red dots) and D_{i+1}^* (blue vectors) generated from the user-specified inputs shown in the left part of the same figure.

Concerning the second step, we blend $u_i = \{P_{i+1}^*, D_{i+1}^*, V_{i+1}^*\}$ (retrieved as described above) with $\{P_{i+1}, D_{i+1}, V_{i+1}\}$ included in the previous network output y_i by following the method proposed in [Holden et al., 2017]. In particular, we blend correspondent elements, e.g., P_{i+1}^* with P_{i+1} or D_{i+1}^* with D_{i+1} , that we generically denote by $a_{i+1}^*, a_{i+1} \in \mathbb{R}^{2 \times r}$, through the blending function

$$\begin{aligned} \Psi : \mathbb{R}^{2 \times r} \times \mathbb{R}^{2 \times r} \times \mathbb{R}^r \times \mathbb{R} &\mapsto \mathbb{R}^{2 \times r}, \\ \Psi(a_{i+1}, a_{i+1}^*, t, \tau_a) &= \text{diag}(1_r - t^{\tau_a}) a_{i+1} + \text{diag}(t^{\tau_a}) a_{i+1}^*, \end{aligned} \quad (5.4.3)$$

where

- the operator $\text{diag}(\cdot) : \mathbb{R}^n \mapsto \mathbb{R}^{n \times n}$ casts its vector argument into a diagonal matrix,
- 1_r is a column vector of ones of dimension r ,
- $t \in \mathbb{R}^r$ is a column vector of dimension r of elements equally-spaced in $[0, 1]$,
- $\tau_a \in \mathbb{R}$ is a parameter, associated to the generic element a to be blended, that controls the responsiveness of the blending.

In particular, we set the responsiveness parameters τ_a for ground-projected base positions, facing directions and ground-projected base velocities to $\tau_P = 1.5$, $\tau_D = 1.3$ and $\tau_V = 1.3$, respectively. An example of the blending of P_{i+1}^* with P_{i+1} and D_{i+1}^* with D_{i+1} according to the selected τ_P and τ_D is shown in Figure 5.5 (right). Especially for the positions P_{i+1} , it can be seen how the blended trajectory (green) remains close to the network prediction (magenta) at the beginning, and smoothly gets closer and closer to the user-specified trajectory (gray) as it goes further in the future, until coinciding with it at the very end. Responsiveness parameters τ_a different from those that we experimentally selected would result in a different shaping of the blended trajectory.

5.5 Results

In this section, we illustrate the results obtained after training the architecture detailed in Section 5.4 on the dataset described in Section 5.3. Notice that the following validation is conducted in a purely kinematic setting, as if we were targeting a computer graphics application. Our first step towards the exploitation of the proposed data-driven trajectory generator for real-world humanoids is indeed to obtain efficiently-generated whole-body trajectories adaptable to diverse walking pattern which also exhibit a certain degree of human-likeness, and all these properties do not need dynamic simulations to be evaluated. For the kinematic validation, we use the iCub v2.7 humanoid robot model.

First, let us detail the frameworks and tools we exploit to implement our learning-based trajectory generator. For calculating rigid-body quantities, both at retargeting and trajectory generation stage, we use the iDynTree library [Nori et al., 2015]. For visualization purposes, i.e., with no physics engine enabled, we exploit the Gazebo Sim simulator. The MANN architecture is implemented in TensorFlow [Abadi et al., 2016] (although we also tested a PyTorch [Paszke et al., 2019] implementation with comparable performances) and trained as follows.

We obtain the input and output matrices $X \in \mathbb{R}^{441570 \times 137}$ and $Y \in \mathbb{R}^{441570 \times 101}$ by stacking the network features computed on the overall retargeted dataset. Normalized to have zero mean and unit variance, X and Y constitute our training dataset. We perform 150 training epochs on such a dataset, using mini-batches of 32 randomly-selected samples, for a total training time of around 25 hours on an NVIDIA GeForce GTX 1650 GPU. The learning task is a classical regression task, aiming at minimizing the mean squared error (MSE) between the ground truth and the network prediction – see Eq. (2.1.18). We use Adam optimizer with warm restart (AdamWR) [Loshchilov and Hutter, 2017], configured with learning rate $\eta = 1.0 \cdot 10^{-4}$, weight decay rate $\lambda = 2.5 \cdot 10^{-3}$, and η -restart control parameters $T_i = 10$ and $T_{mul} = 2$. We apply dropout regularization with a keep-probability set to 0.7.

5.5.1 Learned walking patterns

After training, our MANN implementation proves capable of responsively generating whole-body trajectories for different walking patterns, with each network prediction requiring around 3 ms on a 9-th generation Intel[®] Core i7 CPU @ 2.60 GHz. By simply varying motion and facing directions, the user can generate forward, backward, and sideways walking motions. Changes in the input signals promptly translate into smooth transitions between different walking patterns. By releasing the analog sticks, the user can stop and then restart the robot motion at will.

Figures 5.6, 5.7, 5.8 and 5.9 show the kinematic visualization of a 10-s forward, backward, right-side and left-side motion thus obtained, respectively. The figures include a top view of the generated footsteps, highlighting how each walking pattern is learned with different speed and accuracy. Also depending on the user input processing described in Section 5.4.2, forward walking is generated at higher speed and straighter with respect to backward walking. An intermediate speed and a slight curvature characterize sideways walking. For each generated motion, the mean footstep displacement from the desired motion direction is reported.

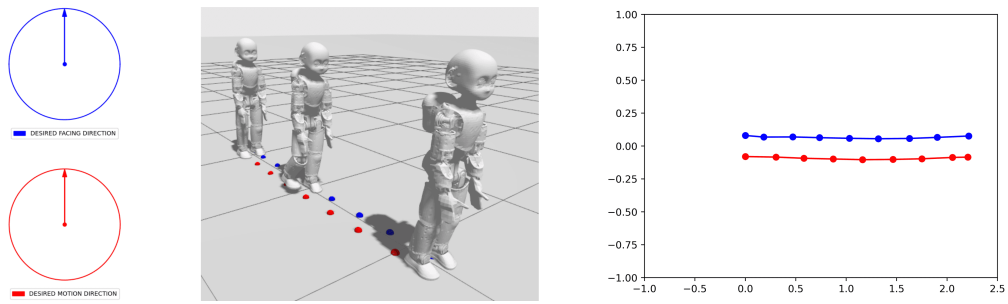


Figure 5.6. 10-s MANN-generated forward walking. Mean footstep displacement: 0.05 cm.

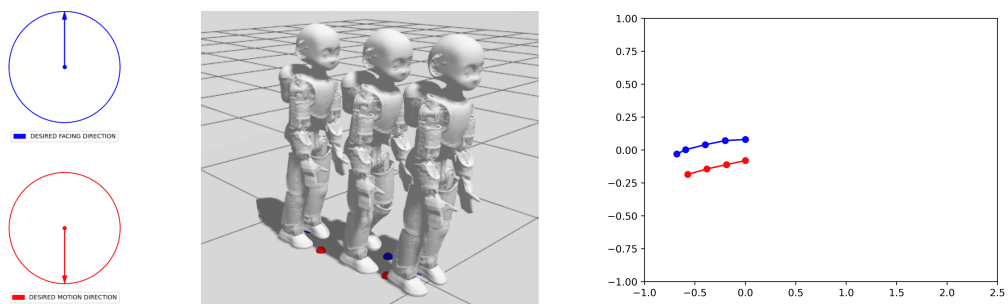


Figure 5.7. 10-s MANN-generated backward walking. Mean footstep displacement: 3.12 cm.

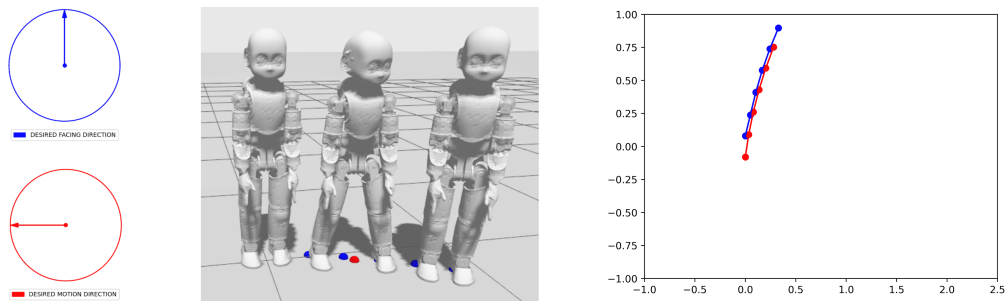


Figure 5.8. 10-s MANN-generated left-side walking. Mean footstep displacement: 6.02 cm.

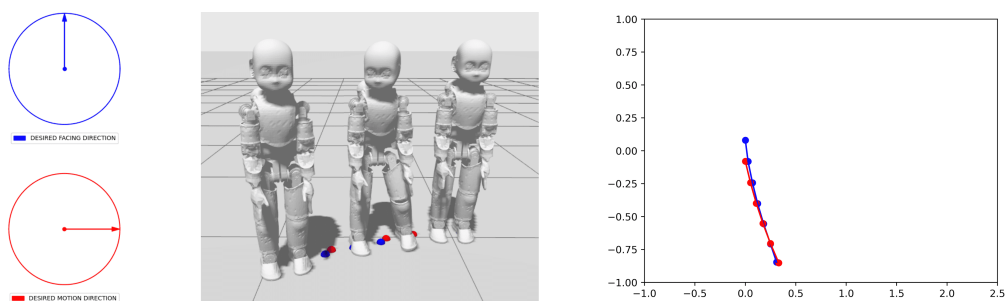


Figure 5.9. 10-s MANN-generated right-side walking. Mean footstep displacement: 5.86 cm.

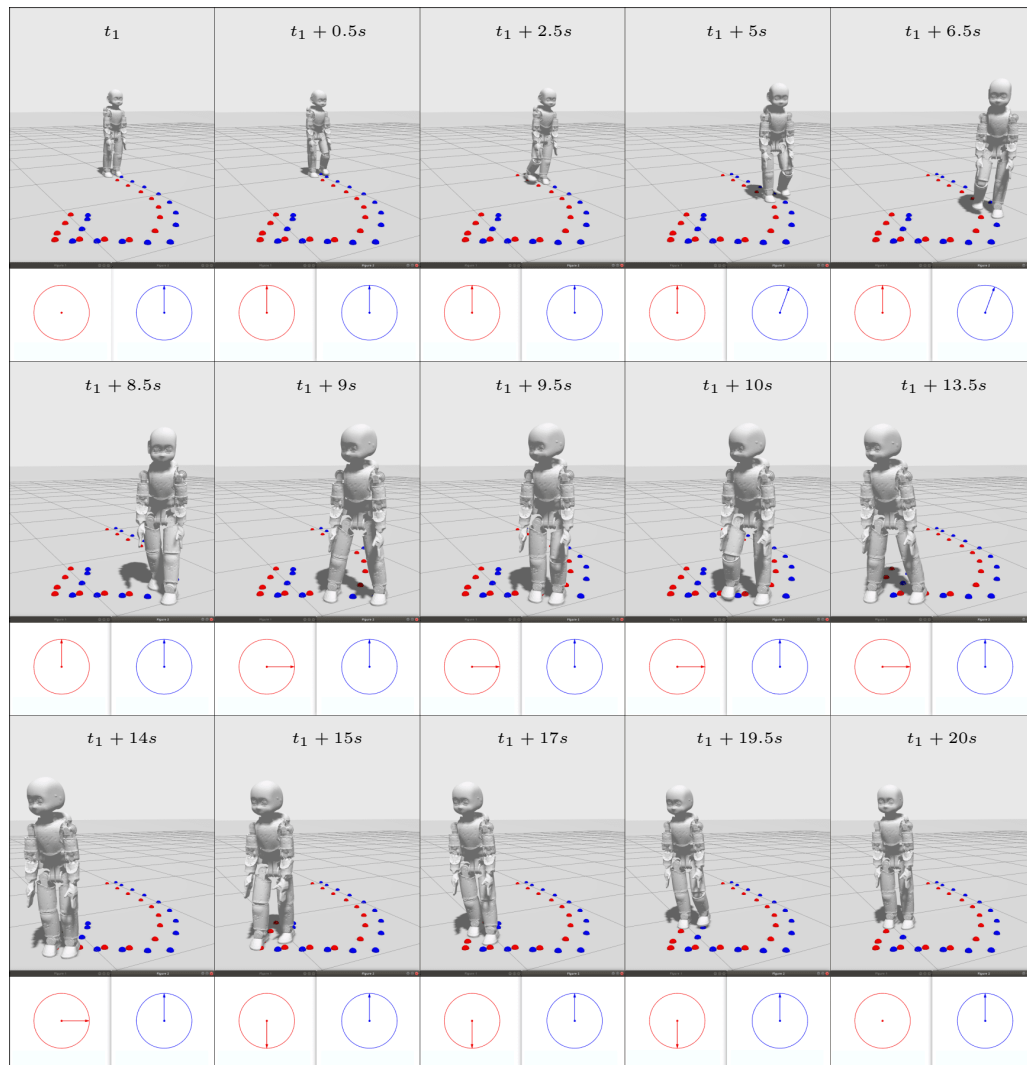


Figure 5.10. Kinematic visualization of a complex trajectory interactively generated using the proposed learning-based architecture. The trajectory spans 20 s. Frames at representative time instants t_i are shown. For each frame, the user-specified motion (red) and facing (blue) directions are displayed below the robot visualization. The trajectory starts and ends with the user asking the robot to stay still. All along the trajectory, a variety of walking patterns are commanded: straight forward walking (e.g., at $t_1 + 0.5s$), right-oriented forward walking (e.g., at $t_1 + 5s$), right-side walking (e.g., at $t_1 + 9s$), and backward walking (e.g., at $t_1 + 15s$). Our MANN implementation successfully generates the requested motions and smoothly handles transitions from one to the other.

Figure 5.10 shows the kinematic visualization of a complex trajectory, including different walking patterns and smooth transitions between them, interactively generated from the user inputs shown below each frame. For the sake of clarity, the contacts between the feet and the ground for the entire trajectory are visualized, in red and blue for the right and left foot, respectively. Further examples of interactive trajectory generations are displayed in a dedicated portion of the accompanying video available at <https://youtu.be/s7-pML0ojK8?t=212>.

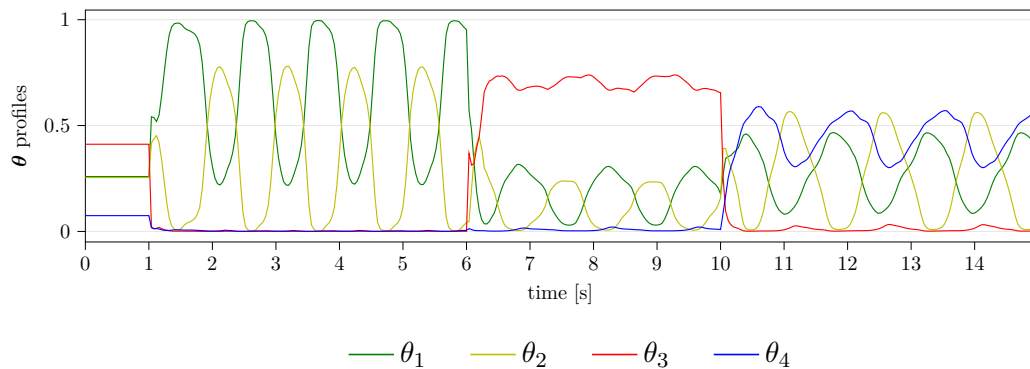


Figure 5.11. Blending coefficients θ profiles for an articulated trajectory of 15 s including standing (0-1 s), straight forward walking (1-4 s), steered forward walking (4-6 s), right-side walking (6-10 s), and left-side walking (10-15 s).

Blending coefficients activation

Besides visually inspecting the quality of the generated trajectories, we analyze how the experts $\{\alpha_1, \dots, \alpha_4\}$ trained in our MANN implementation specialize in different motions. For this purpose, we plot the profiles of the blending coefficients $\theta = \{\theta_1, \dots, \theta_4\}$ weighting the experts to dynamically compute the weights α_i of the Motion Prediction Network – see Eq. (5.1.7).

Figure 5.11 shows the blending coefficient profiles for different walking patterns which follow one another within a trajectory of 15 s. Note how such profiles show distinctive *periodic* patterns characterizing each motion type. For instance, in both the straight and steered forward walking phases, only θ_1 and θ_2 are active, and specialize in left and right swing motions, respectively. On the contrary, θ_3 and θ_4 become more active during right-side and left-side walking, respectively. The specialization of the experts in different motions and their periodic activation represents a qualitative confirmation of the appropriateness of the Mixture of Experts (MoE) approach to the task of generating locomotion trajectories. The real-time evolution of the expert profiles is shown in a dedicated portion of the accompanying video available at <https://youtu.be/s7-pML0ojK8?t=212>.

5.5.2 Human-likeness

We evaluate here the *human-likeness* of the trajectories generated with the proposed approach. Let us clarify first that a quantitative formulation of the concept of human-likeness is still an open point, as well as the definition of an objective and widely-accepted metric to measure it. Nevertheless, in order to understand which degree of human-likeness characterizes the trajectories generated by our MANN implementation, we compare them with trajectories directly retargeted from the human to the robot. Despite the fact that the retargeting process could modify the original motions to adapt them to the robot structure, which is not as flexible as the human one, this is the closest example of human-like motions (by definition) we can think of for our comparison. Therefore, we select trajectories retargeted from the human to the robot as our ground-truth in terms of human-likeness.

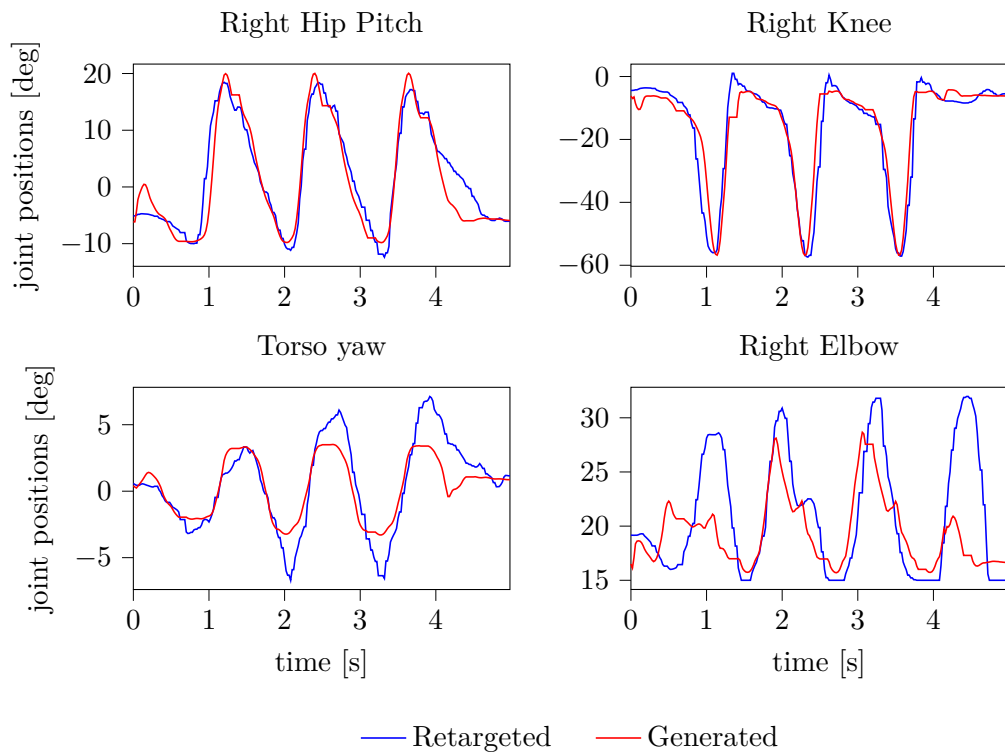


Figure 5.12. Generated vs. human-retargeted joint trajectories for a set of representative joints during forward walking.

In particular, we compare a forward walking generated after training with a human-retargeted forward walking. Figure 5.12 illustrates the comparison for four representative joints spread over the robot body, i.e., the elbow, torso yaw, hip pitch and knee. The generated trajectories do not follow in every detail the retargeted motion they are compared with. This is however expected, since we compare the output of the trajectory generator, learned from a large dataset including several instances of forward walking different from one another, with a specific individual example of retargeted forward walking. Still, the similarity of the patterns emerging in Figure 5.12 demonstrates that the trained model is indeed able to generate trajectories which resemble the human motions retargeted onto the robot, and can therefore be said to have at least a certain degree of *human-likeness*.

An additional indicator of the performances of the trained model in producing motions resembling those it has been trained on is given by the root mean squared error (RMSE) of the network-predicted joint positions $s_i \in y_i$. Notice that we do not consider the RMSE of the entire network output y_i since it includes a lot of other components which are irrelevant in terms of human-likeness of the generated motion. We compute the RMSE of the network-predicted joint positions on a previously unseen test set of around 10000 datapoints including all the considered motions (forward, sideways, and backward) as well as transitions between them and several stops and restarts. An average RMSE of 3.09 deg is obtained, confirming that the network has effectively learned the regression task on this output component.

5.6 Conclusions

In this chapter, we present a learning-based whole-body trajectory generator for humanoid robots, trained by deep supervised learning on a dataset of locomotion trajectories in the form of motion capture data. We collect our dataset to span a wide range of walking motions and retarget it from the human onto the robot while guaranteeing kinematic feasibility. After extracting suitable features from the retargeted dataset, we train our MANN implementation which allows an external user to interactively shape the generated trajectories by simply specifying desired motion and facing directions for the robot.

We validate the proposed approach, in a kinematic setting, on the iCub v2.7 humanoid robot model. Our trajectory generator proves efficient (3 ms per prediction step) at providing responsive whole-body trajectories for different walking patterns while smoothly handling transitions among them. Moreover, we show that the generated trajectories exhibit a certain degree of human-likeness, inherited from the human-retargeted training data.

Our approach, assuming at least one known contact with the environment at any time step, cannot be applied to contact-free motions such as running or jumping. Its extension to motion skills other than simple locomotion, given the task-oriented choice of training features and processing of the user inputs, requires careful investigation. Moreover, the proposed architecture must be retrained from scratch whenever new motion skills are added. Besides architectural changes to increase modularity, future work could pursue navigation of uneven ground by including perceptual terrain features in the network input.

However, in the context of walking pattern generation on flat terrains, our learning-based whole-body trajectory generator demonstrates efficient and general enough to motivate its integration with state-of-the-art controllers for humanoid locomotion, that is the topic addressed in the next chapter.

Chapter 6

Human-Like Whole-Body Control of Humanoid Robots

In the previous chapter, we introduced a learning-based trajectory generator for humanoid robots capable of efficiently producing whole-body trajectories that reactively follow the desired motion specified by an external user while mimicking the motion capture data used for training. This chapter deals with the integration of the proposed trajectory generator with a state-of-the-art whole-body controller for humanoid robots, with the aim of deploying efficient human-like locomotion on real-world humanoids.

We name the end-to-end system architecture resulting from such an integration ADHERENT (humAn-Driven wHolE-body REference geNerator and conTroller). With reference to the block diagram of ADHERENT depicted in Figure 6.1, the pipeline that leads to the online *MANN Inference* block leverages the work presented in the previous chapter. The exploitation of the MANN output within the *Trajectory Control* component constitutes instead the topic of this chapter, along with some extensions of the ADHERENT framework towards additional motions and different robotic platforms. We validate ADHERENT with extensive simulations and real-world experiments on both the iCub v2.7 and the iCub v3 humanoid robots (see Sections 3.4.1 and 3.4.2, respectively), thus demonstrating the robustness and transferability of the proposed approach.

The detailed organization of the chapter follows. Section 6.1 illustrates the different layers composing a state-of-the-art hierarchical control architecture for humanoid locomotion. Section 6.2 describes how to integrate our learning-based whole-body trajectory generator with the aforementioned hierarchical control architecture to form the end-to-end ADHERENT framework. An extension of ADHERENT covering crouching motions is illustrated in Section 6.3. Section 6.4 presents the experimental validation of ADHERENT, also in terms of portability across different robotic platforms. Finally, Section 6.5 concludes the chapter.

The content of this chapter partially appears in:

Viceconte, P. M., Camoriano, R., Romualdi, G., Ferigo, D., Dafarra, S., Traversaro, S., Oriolo, G., Rosasco, L., and Pucci, D. (2022). ADHERENT: Learning Human-like Trajectory Generators for Whole-body Control of

Humanoid Robots. *IEEE Robotics and Automation Letters*, 7(2):2779–2786.

Video <https://www.youtube.com/watch?v=s7-pML0ojK8>

Github [ami-iit/paper_viceconte_2021_ral_adherent](https://github.com/ami-iit/paper_viceconte_2021_ral_adherent)

Zenodo https://zenodo.org/record/6201915#.Yh0T_Bso-8g

The content of Sections 6.3 and 6.4.5 will be submitted for a publication tentatively titled as:

D’Elia, E., Viceconte, P. M., Rapetti, L., and Pucci, D. Learning Human-like Trajectory Generators for Humanoid Robot Locomotion with Crouching Abilities. (To be submitted).

Video <https://www.youtube.com/watch?v=Dor1hMqAAmo>

6.1 Background

We introduce here the state-of-the-art control architecture for humanoid locomotion that we selected for the integration with our learning-based trajectory generator. In Section 3.1, we already presented a commonly-adopted approach to address bipedal locomotion which employs the three-layer architecture depicted in Figure 3.1. Among the alternative architectures proposed in [Romualdi et al., 2018, 2020; Romualdi, 2022] by following this approach, we select and present here the one that defines:

- A *trajectory optimization* layer that starts from a unicycle-based planner to generate the desired DCM (see Section 1.3.3) and foot trajectories;
- A *simplified model control* layer that implements an instantaneous control law for the tracking of the DCM;
- A *whole-body QP control* layer that implements a kinematics-based controller ensuring the tracking of the desired CoM and foot trajectories by considering the complete robot kinematics.

The three aforementioned layers are described in the following.

6.1.1 Trajectory optimization layer

Assuming flat terrain, the trajectory optimization layer proposed in [Romualdi et al., 2018, 2020] relies on a simple unicycle model [Siciliano et al., 2008, Chapter 11] to generate the desired footstep locations and timings, which represents a viable solution either with constant [Faragasso et al., 2013; Cognetti et al., 2016] or variable [Dafarra et al., 2018] length and velocity of the footsteps. Since in our case the footstep plan provided by the learning-based trajectory generator will remove the need for the unicycle-based footstep generator, we skip the details related to the unicycle planner in this overview.

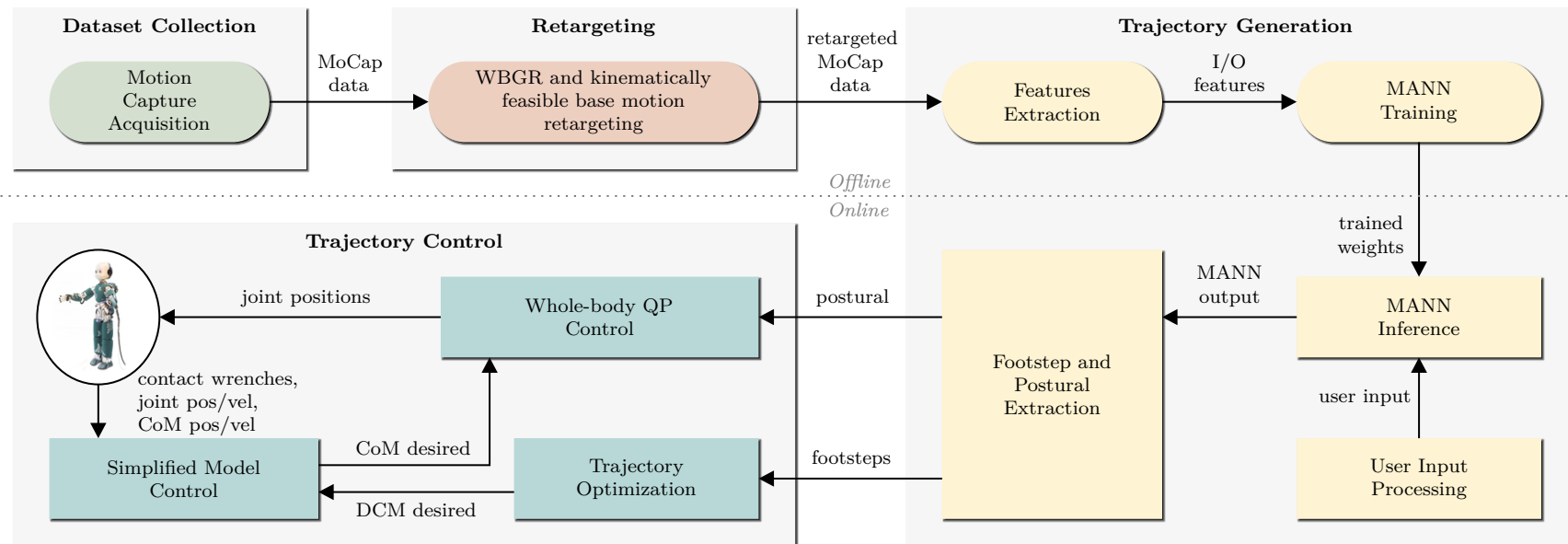


Figure 6.1. Block diagram of the end-to-end ADHERENT architecture proposed in this work, a comprehensive learning-based architecture for efficient human-like whole-body trajectory generation and control of humanoid robots. ADHERENT consists of four main components: Dataset Collection, Retargeting, Trajectory Generation, and Trajectory Control. In light of the modularity characterizing such an architecture, specific methods implementing each component in the currently-proposed implementation can be easily replaced by more efficient and effective ones in future instances of the architecture.

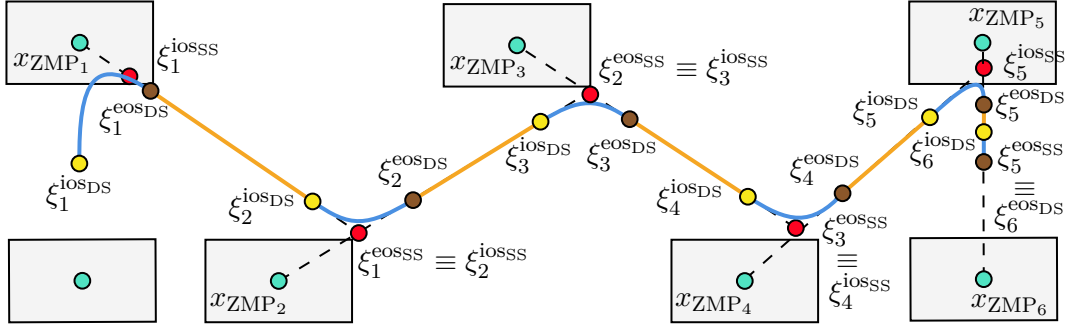


Figure 6.2. DCM trajectory planning for five straight footsteps on a flat terrain. The orange segments represent the portions of DCM trajectory planned in SS by applying the exponential interpolation technique in Eq. (6.1.1) between pairs of $\{\xi_i^{iosss}, \xi_i^{eosss}\}$ (red circles). The blue curves represent the portions of the DCM trajectory planned in DS by applying the polynomial interpolation method in Eq. (6.1.3) between pairs of $\{\xi_i^{iosds}, \xi_i^{eosds}\}$ (yellow and brown circles, respectively).

DCM trajectory planner

Given a set of footstep locations and timings from an external footstep planner, the DCM trajectory generator proposed in [Romualdi et al., 2018, 2020] extends [Englsberger et al., 2014] to plan a desired DCM trajectory which takes into account both single support (SS) and double support (DS) phases. An example of the desired DCM trajectory generated in the case of five straight footsteps on a flat terrain is illustrated in Figure 6.2.

First, the reference DCM trajectory during SS phases is planned. Assuming the LIP model hypothesis (see Section 1.3.1), the reference DCM trajectory in SS can be retrieved from the solution of Eq. (1.3.25) for a constant ZMP [Romualdi, 2022, Section 10.1.3]. In particular, the desired DCM ξ_i^{SS} for the i -th SS step is given by

$$\xi_i^{SS}(t) = xZMP_i + e^{\omega(t-t_i^{\text{step}})}(\xi_i^{eosss} - xZMP_i), \quad (6.1.1)$$

where $xZMP_i$ is the desired ZMP location during the i -th step (fixed and coincident with the step location), t_i^{step} is the step duration, ξ_i^{eosss} is the end-of-step desired DCM position, ω is the time constant of the LIP model, and the time t belongs to the step domain $t \in [0, t_i^{\text{step}}]$. Moreover, assume that the position of the reference DCM at the end of the last SS phase belongs to the affine combination of the ZMPs at the two last steps, i.e.,

$$\xi_{N-1}^{eosss} = \alpha_{LS} xZMP_N + (1 - \alpha_{LS}) xZMP_{N-1}, \quad (6.1.2)$$

where the parameter α_{LS} weighs the last two ZMP locations in the computation of ξ_{N-1}^{eosss} (e.g., ξ_5^{eosss} in Figure 6.2). Since instantaneous transitions between consecutive SS phases are assumed when planning the DCM in SS, consecutive steps are related by $\xi_{i-1}^{eosss} = \xi_i^{iosss}$, where ξ_i^{iosss} is the initial desired DCM position for the i -th step. A recursive application of the exponential interpolation technique in Eq. (6.1.1) from the last to the first step of the given footstep plan allows therefore to retrieve the desired DCM trajectory for the SS steps, depicted in orange in Figure 6.2.

Then, the reference DCM trajectory during DS phases is planned. Since a DCM trajectory with continuous derivative guarantees a continuous ZMP trajectory, third-order polynomials are exploited to smooth the edges of the reference DCM trajectory in DS. The desired DCM ξ^{DS} in DS is given by the polynomial interpolation

$$\xi^{\text{DS}}(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0, \quad (6.1.3)$$

where the coefficients a_j are chosen to satisfy the DCM velocity and position boundary conditions at the beginning and at the end of each DS phase – please refer to [Romualdi, 2022, Section 10.1.3] for further details. Figure 6.2 depicts in blue the desired DCM trajectory in DS, which smooths the overall DCM trajectory.

Finally, further boundary conditions to the DCM generation problem are applied in [Romualdi, 2022, Section 10.2.1] to ensure that the initial desired DCM ξ_1^{iOSDS} coincides with the measured ground-projected CoM and the final desired DCM ξ_N^{eOSDS} with the desired ground-projected CoM, i.e., the middle point between the feet.

Swing foot trajectory planner

Given the set of footstep locations and timings from an external footstep planner, the swing foot trajectory planner proposed in [Romualdi et al., 2018, 2020] aims to find minimum acceleration trajectories between pairs of consecutive footstep poses. The positional swing foot trajectory $p_F(t) \in \mathbb{R}^3$ and its rotational trajectory ${}^{\mathcal{I}}R_F(t) \in \text{SO}(3)$ are evaluated separately as follows.

Throughout a step, the swing foot moves from the initial position $p_F(t_0) = p_{F_0}$ to the final position $p_F(t_N) = p_{F_N}$, reaching its maximum height $p_F(t_{\text{apex}}) = p_{F_{\text{apex}}}$ at the apex time t_{apex} . The minimum acceleration trajectory $p_F(t) \in \mathbb{R}^3$ from p_{F_0} to p_{F_N} which passes by $p_{F_{\text{apex}}}$ at t_{apex} is obtained by concatenating 3rd-order polynomial functions [Romualdi, 2022, Section 10.2.2]

$$p_F(t) = \begin{cases} a_{0,3}t^3 + a_{0,2}t^2 + a_{0,1}t + a_{0,0} & \text{if } t_0 \leq t \leq t_{\text{apex}}, \\ a_{1,3}t^3 + a_{1,2}t^2 + a_{1,1}t + a_{1,0} & \text{if } t_{\text{apex}} < t \leq t_N, \end{cases} \quad (6.1.4)$$

with the coefficients $a_{i,j}$ chosen to meet the position and velocity boundary conditions for each portion of the trajectory.

Moreover, during the step, the swing foot rotates from the initial orientation ${}^{\mathcal{I}}R_F(t_0) = {}^{\mathcal{I}}R_{F_0}$ to the final orientation ${}^{\mathcal{I}}R_F(t_N) = {}^{\mathcal{I}}R_{F_N}$. The minimum acceleration trajectory ${}^{\mathcal{I}}R_F(t) \in \text{SO}(3)$ from ${}^{\mathcal{I}}R_{F_0}$ to ${}^{\mathcal{I}}R_{F_N}$, with zero initial and final angular velocity, is given by [Romualdi, 2022, Section 10.2.2]

$${}^{\mathcal{I}}R_F(t) = e^{\left(s(t-t_0) \log \left({}^{\mathcal{I}}R_{F_N} {}^{\mathcal{I}}R_{F_0}^\top \right) \right)} {}^{\mathcal{I}}R_{F_0}, \quad (6.1.5)$$

where $s(\tau)$ is defined as

$$s(\tau) = \frac{3}{(t_N - t_0)^2} \tau^2 - \frac{3}{(t_N - t_0)^3} \tau^3. \quad (6.1.6)$$

6.1.2 Simplified model control layer

Given a desired DCM trajectory, the simplified model control layer aims at stabilizing it by assuming the ZMP position x_{ZMP} as control input, and consists of a DCM instantaneous controller followed by a ZMP-CoM controller.

DCM instantaneous controller

The DCM *instantaneous* controller implements the control law

$$x_{\text{ZMP}}^* = \xi^{\text{ref}} - \frac{1}{\omega} \dot{\xi}^{\text{ref}} + K_p^\xi (\xi - \xi^{\text{ref}}) + K_i^\xi \int \xi - \xi^{\text{ref}} dt, \quad (6.1.7)$$

where $K_p^\xi > I_2$ and $K_i^\xi > 0_{2 \times 2}$. Applied to Eq. (1.3.25), the proposed control law leads to the closed-loop dynamics

$$\dot{\xi} - \dot{\xi}^{\text{ref}} = \omega \left(I_2 - K_p^\xi \right) \left(\xi - \xi^{\text{ref}} \right) - \omega K_i^\xi \int \xi - \xi^{\text{ref}} dt, \quad (6.1.8)$$

and therefore guarantees the tracking of the DCM references, with the DCM error and its integral asymptotically converging to zero. However, it does not guarantee the ZMP stability criterion (see Section 1.3.2), since it does not constrain the ZMP strictly inside the support polygon, and may therefore lead to unfeasible motions.

ZMP-CoM controller

The desired ZMP x_{ZMP}^* returned by the DCM instantaneous controller according to Eq. (6.1.7) is then stabilized, along with the planar CoM dynamics from Eq. (1.3.4), by the control law [Choi et al., 2006]

$$\dot{x}_{\text{LIP}}^* = \dot{x}_{\text{LIP}}^{\text{ref}} - K_{\text{ZMP}} \left(x_{\text{ZMP}}^{\text{ref}} - x_{\text{ZMP}} \right) + K_{\text{LIP}} \left(x_{\text{LIP}}^{\text{ref}} - x_{\text{LIP}} \right), \quad (6.1.9)$$

where $K_{\text{LIP}} > \omega I_2$ and $0_2 < K_{\text{ZMP}} < \omega I_2$.

6.1.3 Whole-body QP control layer

Given the desired CoM and foot trajectories, along with other kinematic references, the *kinematics-based* whole-body QP control layer ensures their tracking by considering the complete robot kinematics. In particular, it computes the mixed system velocity ${}^{B[Z]}\nu$, defined in Eq. (1.2.16) and simply denoted as ν here, by following the *stack-of-tasks* approach briefly introduced in Section 3.1.3. The constrained optimization problem embedding low-priority tasks in the cost function and dealing with high-priority tasks as constraints is expressed as a *quadratic programming*¹ (QP) problem and solved using off-the-shelf solvers, e.g., [Stellato et al., 2020].

¹An optimization problem with n variables and m constraints is defined a quadratic programming (QP) problem if the objective is convex quadratic and the constraints are affine functions. A QP problem can be expressed in the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2} x^\top P x + q^\top x + r \\ \text{s.t.} \quad & G x \preceq h, \end{aligned} \quad (6.1.10)$$

where $x \in \mathbb{R}^n$, $P \succ 0$ is a positive definite matrix, $q \in \mathbb{R}^n$, $G \in \mathbb{R}^{m \times n}$, $h \in \mathbb{R}^m$, and $r \in \mathbb{R}$. If the set $S = \{x | Gx \preceq h\}$ is not empty, the solution to the QP problem exists and is unique. In particular, being the cost function quadratic and therefore convex, such a solution is retrieved by setting the gradient of the cost function to zero, i.e.,

$$\nabla_x \left[\frac{1}{2} x^\top P x + q^\top x + r \right] = 0, \quad (6.1.11)$$

which leads to the closed-form solution $x^* = -P^{-1}q$.

Low and high priority tasks

To evaluate the desired system velocity ν , the kinematics-based whole-body QP controller exploits the following tasks, either with low and high priority.

The *centroidal momentum task* Ψ_h specifies a desired trajectory for the centroidal momentum $\bar{G}h$ (see Section 1.2.3) in the form

$$\Psi_h = \bar{G}h^* - J_{\text{CMM}}\nu = \begin{bmatrix} \bar{G}h^{p^*} \\ \bar{G}h^{\omega^*} \end{bmatrix} - \begin{bmatrix} I_3 & 0_{3 \times 3} \\ 0_{3 \times 3} & I_3 \end{bmatrix} \begin{bmatrix} J_{\text{CMM}_p} \\ J_{\text{CMM}_\omega} \end{bmatrix} \nu \quad (6.1.12a)$$

$$= \begin{bmatrix} \bar{G}h^{p^*} - J_{\text{CMM}_p}\nu \\ \bar{G}h^{\omega^*} - J_{\text{CMM}_\omega}\nu \end{bmatrix} = \begin{bmatrix} \Psi_{\text{CoM}} \\ \Psi_{h^\omega} \end{bmatrix}, \quad (6.1.12b)$$

where J_{CMM} is the centroidal momentum matrix (see Section 1.2.3) and Eq. (6.1.12b) makes explicit the task decomposition into its linear Ψ_{CoM} and angular Ψ_{h^ω} components. The linear target $\bar{G}h^{p^*}$ is often selected to guarantee the tracking of the reference CoM, while the angular target $\bar{G}h^{\omega^*}$ is often set equal to zero, i.e.,

$$\bar{G}h^{p^*} = m \left[\dot{x}_{\text{CoM}}^{\text{ref}} + K_{\text{CoM}} \left(x_{\text{CoM}}^{\text{ref}} - x_{\text{CoM}} \right) \right], \quad (6.1.13)$$

$$\bar{G}h^{\omega^*} = 0_{3 \times 1}, \quad (6.1.14)$$

where K_{CoM} is a positive matrix.

The *cartesian task* $\Psi_{L_{\text{SE}(3)}}$ specifies a desired cartesian trajectory ${}^{\mathcal{I}}H_L^{\text{ref}} = (p_L^{\text{ref}}, {}^{\mathcal{I}}R_L^{\text{ref}})$ for the link L via its mixed velocity ${}^{L[\mathcal{I}]}v_{\mathcal{I},L}$ (see Eq. (1.2.24)) as

$$\Psi_{L_{\text{SE}(3)}} = {}^{L[\mathcal{I}]}v_{\mathcal{I},L}^* - J_L\nu = \begin{bmatrix} \dot{p}_L^* \\ {}^{\mathcal{I}}\omega_{\mathcal{I},L}^* \end{bmatrix} - \begin{bmatrix} I_3 & 0_{3 \times 3} \\ 0_{3 \times 3} & I_3 \end{bmatrix} \begin{bmatrix} J_{L_p} \\ J_{L_\omega} \end{bmatrix} \nu \quad (6.1.15a)$$

$$= \begin{bmatrix} \dot{p}_L^* - J_{L_p}\nu \\ {}^{\mathcal{I}}\omega_{\mathcal{I},L}^* - J_{L_\omega}\nu \end{bmatrix} = \begin{bmatrix} \Psi_{L_{\mathbb{R}^3}} \\ \Psi_{L_{\text{SO}(3)}} \end{bmatrix}, \quad (6.1.15b)$$

where J_L is the mixed Jacobian of the link L (see Eq. (1.2.24)) and Eq. (6.1.15b) makes explicit the task decomposition into its linear $\Psi_{L_{\mathbb{R}^3}}$ and angular $\Psi_{L_{\text{SO}(3)}}$ components. The linear \dot{p}_L^* and angular ${}^{\mathcal{I}}\omega_{\mathcal{I},L}^*$ targets are selected to guarantee the tracking of the reference position p_L^{ref} and orientation ${}^{\mathcal{I}}R_L^{\text{ref}}$, respectively, i.e.,

$$\dot{p}_L^* = \dot{p}_L^{\text{ref}} + K_{L_p} \left(p_L^{\text{ref}} - p_L \right), \quad (6.1.16)$$

$${}^{\mathcal{I}}\omega_{\mathcal{I},L}^* = {}^{\mathcal{I}}\omega_{\mathcal{I},L}^{\text{ref}} + K_{L_\omega} \log \left({}^{\mathcal{I}}R_L^{\text{ref}} {}^{\mathcal{I}}R_L^\top \right), \quad (6.1.17)$$

where K_{L_p} and K_{L_ω} are positive matrices.

Finally, the *joint regularization task* Ψ_s specifies desired values s^{ref} for the joint positions through the joint velocities \dot{s} in the form

$$\Psi_s = \dot{s}^* - \begin{bmatrix} 0_{n \times 6} & I_n \end{bmatrix} \nu, \quad (6.1.18)$$

where n are the robot actuated degrees of freedom and the target \dot{s}^* is chosen as

$$\dot{s}^* = \dot{s}^{\text{ref}} + K_s \left(s^{\text{ref}} - s \right), \quad (6.1.19)$$

where K_s is a positive definite diagonal matrix.

6.2 Trajectory control

We illustrate here how we integrate the learning-based whole-body trajectory generator presented in Chapter 5 with the hierarchical state-of-the-art control architecture for humanoid locomotion introduced in Section 6.1 to form the end-to-end ADHERENT framework shown in Figure 6.1. In the following, we detail our implementation of the three layers composing the selected locomotion controller, with a particular focus on how each individual layer exploits the output of the learning-based whole-body trajectory generator.

Trajectory optimization layer

We provide as inputs to both the DCM planner (see Section 6.1.1) and the swing foot planner (see Section 6.1.1) the desired feet positions and orientations composing the footstep plan directly retrieved from the MANN-generated trajectories. While the interactive trajectory generation proceeds (in open-loop with respect to the actual tracking on the robot), our *Footstep Extractor* module adds a new footstep location to the plan every time a change of support foot is detected. Concerning orientations, since we assume a flat terrain, only the network-predicted yaw angle of the support foot is considered in the plan.

Notice that the direct exploitation of the MANN-generated trajectories to build a footstep plan in the proposed architecture removes the need for commonly-employed external footstep planners, such as the unicycle-based ones [Faragasso et al., 2013; Cognetti et al., 2016; Dafarra et al., 2018], which employ simplified models limiting their generalization capabilities when generating footsteps.

Finally, let us remark that also the swing foot trajectories may have been directly extracted from the MANN-generated robot motion, removing therefore the need for the swing foot planner as well. We observe, however, that minimum acceleration swing trajectories with a fixed maximum height $p_{F_{\text{apex}}}$ (see Section 6.1.1) increase the overall feasibility of the walking motions (keep in mind that the employed DCM instantaneous controller from Section 6.1.2 does not guarantee feasible motions). Therefore, we maintain the swing foot planner in our current implementation.

Simplified model control layer

We compute the desired CoM velocity \dot{x}_{LIP}^* by concatenating the DCM instantaneous controller from Eq. (6.1.7) and the ZMP-CoM controller from Eq. (6.1.9). Then, we retrieve the desired CoM position x_{LIP}^* by simple Euler integration.

Whole-body QP control layer

We consider the following high and low priority tasks for the kinematics-based whole-body QP controller:

- The right and left feet poses returned by the swing foot planner are considered as high-priority cartesian tasks (see Eq. (6.1.15)), denoted by $\Psi_{R_{\text{SE}(3)}}$ and $\Psi_{L_{\text{SE}(3)}}$, respectively.

- The CoM trajectory returned by the simplified model control layer is considered as a high-priority linear momentum task (see Eq. (6.1.12b)), denoted by Ψ_{CoM} .
- The data-driven whole-body joint postural $\{s, \dot{s}\}$ included in the MANN-generated output y (see Eq. (5.4.2)) is considered as low-priority joint regularization task (see Eq. (6.1.18)), denoted by Ψ_s . Notice that while our learning-based trajectory generator provides references compatible with the frequency of the data on which it has been trained, i.e., 60 Hz, the whole-body QP controller expects postural references at its own control frequency. Therefore, we implement a *Postural Extractor* module which smoothly interpolates network-predicted posturals to obtain references at the required frequency.
- An additional task with the purpose of zeroing the torso roll and pitch angles is considered as low-priority SO(3) task (see Eq. (6.1.15b)), denoted by $\Psi_{T_{\text{SO}(3)}}$.

Notice that, as opposed to classical humanoid robot locomotion controllers which usually adopt a constant postural term for the joints to remain as close as possible to a fixed predefined configuration [Righetti et al., 2011; Romualdi et al., 2018, 2020; Nava et al., 2016], our data-driven postural changes in time and acts as a joint regularizer towards human-like walking motions.

As regards the SO(3) task related to the roll and pitch angles for the torso, we observe that it increases the overall feasibility of the walking motions (that is not guaranteed by the employed DCM instantaneous controller described in Section 6.1.2). Therefore, we decide to add it despite the fact that it competes with the motion of the torso promoted by the network-generated whole-body postural. The torso yaw angle remains, however, free of following as much as possible the data-driven postural reference.

The whole-body optimization problem resulting from the definition of the aforementioned high and low priority tasks takes the form:

$$\underset{\nu}{\text{minimize}} \quad \Psi_{T_{\text{SO}(3)}}^\top \Lambda_T \Psi_{T_{\text{SO}(3)}} + \Psi_s^\top \Lambda_s \Psi_s \quad (6.2.1a)$$

$$\text{s.t.} \quad \Psi_{R_{\text{SE}(3)}} = 0 \quad (6.2.1b)$$

$$\Psi_{L_{\text{SE}(3)}} = 0 \quad (6.2.1c)$$

$$\Psi_{\text{CoM}}. \quad (6.2.1d)$$

Being the tasks linearly dependent on the robot velocity ν , the optimization problem in Eq. (6.2.1) can be casted into a QP problem which takes the form:

$$\underset{\nu}{\text{minimize}} \quad \nu^\top H \nu + 2g^\top \nu \quad (6.2.2)$$

$$\text{s.t.} \quad A\nu \preceq b,$$

where the Hessian matrix H and the gradient vector g are evaluated from the cost function in Eq. (6.2.1a), while the constraint matrix A and vector b are evaluated from the constraints in Eqs. (6.2.1b), (6.2.1c), and (6.2.1d).

The QP problem in Eq. (6.2.2) is solved using off-the-shelf solvers, e.g., [Stellato et al., 2020], to retrieve the optimal ν^* . Finally, the desired joint velocity \dot{s}^* included in the QP problem solution ν^* is integrated to obtain the desired joint position s^* actually set as references for the low-level position controller.

6.3 Crouching ability

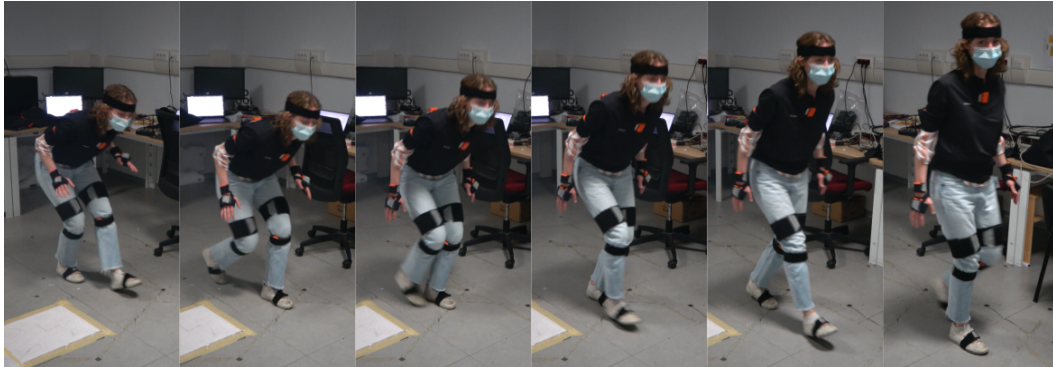


Figure 6.3. An example of crouching-to-upright walking transition included in the motion capture dataset collected for the extension of ADHERENT to crouching skills. Notice how the subject makes the transition while performing a step, i.e., without stopping.

Table 6.1. Breakdown of the crouching motion capture dataset. For each walking motion, also the number of transitions (both crouching-to-upright and viceversa) is reported.

Walking motion	Duration [min]	Datapoints	Stops	Transitions
Forward	3.8	13775	13	40
Backward	3.5	12429	7	32
Side	3.8	13821	7	30
Diagonal	1.6	5628	5	13
Mixed	11.4	41125	31	64

We describe here our investigation about the flexibility of the proposed ADHERENT architecture with respect to the addition of new motion skills. For this purpose, we extend ADHERENT to handle crouching capabilities while walking, including smooth transitions from upright to crouching walking and viceversa. Crouching is per se a desirable motion capability, because it can enhance walking stability [Lee et al., 2015] and help navigating in scenarios with low ceiling. More importantly, it allows us to assess the modularity of the proposed framework. In the following, we present the modifications to the whole ADHERENT pipeline (from dataset collection to trajectory control) required to extend it towards the new crouching skill.

6.3.1 Dataset collection and retargeting

To consider crouching motions, we extend our motion capture dataset presented in Section 5.2. The new dataset contains the same walking patterns considered in Section 5.2 but performed while crouching, including starts and stops as well as transitions from crouching to upright walking and viceversa. Details about the collected motions are reported in Table 6.1, while Figure 6.3 shows an example of crouching-to-upright walking transition. Before mirroring, our dataset comprises around 25 minutes of unlabeled motion capture data, that we retarget using the very same procedure guaranteeing kinematic feasibility described in Section 5.3.

6.3.2 Trajectory generation and control

For the network to distinguish and learn crouching, and for the user to be able to interactively command such an additional motion, we implement the following modifications to the trajectory generation and control ADHERENT components.

Crouching parameterization

To parameterize crouching, we examine different metrics (including the evolution of the CoM, the robot base frame \mathcal{B} and its head frame \mathcal{H}) looking for those demonstrating a more distinctive variation between crouching and upright walking. Due to the forward leaning characterizing crouching, the x coordinates of the CoM ${}^{\mathcal{B}}x_{\text{CoM}}$ and head ${}^{\mathcal{B}}x_h^x$, expressed locally in the robot base frame \mathcal{B} , as well as the global head height ${}^{\mathcal{I}}x_h^z$, show a remarkable variation between crouching and upright walking. Since the CoM is also influenced by other types of actions besides crouching, e.g., reaching the arms out to hold an object, we decide to parameterize crouching using information related to the head frame only, i.e., ${}^{\mathcal{B}}x_h^x$ and ${}^{\mathcal{I}}x_h^z$. The head motion is quite a natural parameterization for crouching, whose main purpose is lowering the head to pass under low ceilings. Table 6.4 (left) reports the average variation of the selected features between crouching and upright walking, for a sample forward walking retargeted from the human onto the robot.

Features extraction

The selected features, extracted over the 2 s time window including $k = 2r = 12$ datapoints considered in Section 5.4.1, are added to the MANN input which becomes

$$x_i = \left\{ \underbrace{\text{vec}(H_i)}_{\text{Crouching parameterization}}, \underbrace{\text{vec}(P_i), \text{vec}(D_i), \text{vec}(V_i), l_i}_{\text{Ground-projected base trajectory data}}, \underbrace{s_{i-1}, \dot{s}_{i-1}}_{\text{Joint state}} \right\} \in \mathbb{R}^{161}, \quad (6.3.1)$$

where the $\text{vec}(\cdot)$ operator vectorizes matrices by rows and $H_i = \{h_i^1, \dots, h_i^k\} \in \mathbb{R}^{2 \times k}$, with $h = [{}^{\mathcal{B}}x_h^x \quad {}^{\mathcal{I}}x_h^z]^\top$, are the head local x and global z coordinates, while the other terms are defined as in Eq. (5.4.1).

Similarly, by adding the crouching parameterization, the MANN output becomes

$$y_i = \left\{ \underbrace{\text{vec}(H_{i+1})}_{\text{Future crouching parameterization}}, \underbrace{\text{vec}(P_{i+1}), \text{vec}(D_{i+1}), \text{vec}(V_{i+1})}_{\text{Future ground-projected base trajectory data}}, \underbrace{s_i, \dot{s}_i}_{\text{Joint state}}, \underbrace{\dot{b}_i^a}_{\text{Base yaw variation}} \right\} \in \mathbb{R}^{113}, \quad (6.3.2)$$

where the $\text{vec}(\cdot)$ operator vectorizes matrices by rows, $H_{i+1} = \{h_{i+1}^1, \dots, h_{i+1}^r\} \in \mathbb{R}^{2 \times r}$ are the future head local x and global z coordinates, and the remaining terms are defined as in Eq. (5.4.2).

Let us recall that the extension of the network input and output vectors to include the additional features which parameterize crouching is required to have a future head trajectory in the network prediction blended with the user-specified input. Blending the network-predicted head trajectory with the desired one specified by the user (as opposed to feeding the network directly with the user-specified input) ensures indeed gradual changes in the network input when crouching is enabled or disabled, that is coherent with the training data in which transitions between upright and crouching walking take at least 0.5 s to complete.

User input processing

The crouching input joins the two other existing input signals, i.e., the motion and facing directions introduced in Section 5.4.2, as a third user-specified input for trajectory generation. In particular, the user is allowed to enable or disable crouching by pressing a dedicated button.

Depending on the boolean crouching status, i.e., either crouching enabled or disabled, a future desired trajectory H_{i+1}^* for the head is planned, whose components are defined in Eq. (6.3.2). The processing of the crouching input is simpler than the processing of the other user-specified inputs performed to retrieve the desired future ground-projected base trajectory $u_i = \{P_{i+1}^*, D_{i+1}^*, V_{i+1}^*\}$ described in Section 5.4.2. Specifically, the future head trajectory retrieved from the crouching input is a constant trajectory over the 1-s future time window. When crouching is disabled, it prescribes the features ${}^{\mathcal{B}}x_h^x$ and ${}^{\mathcal{I}}x_h^z$ to remain at their nominal values for upright walking for the entire window. When crouching is enabled, both features are shifted by their average variation between crouching and upright walking measured from the retargeted dataset (see Table 6.4, left).

Finally, the user-specified H_{i+1}^* is blended with the network-predicted H_{i+1} using the very same function adopted to blend corresponding elements in u_i , i.e., the blending function defined in Eq. (5.4.3). For the head trajectories, we use a responsiveness parameter $\tau_H = 1.5$.

Trajectory control

To stabilize trajectories that also include crouching motions, we exploit our implementation of the trajectory optimization, simplified model control and whole-body QP control layers detailed in Section 6.2, with no modifications.

Since the crouching objective is hindered by the $\text{SO}(3)$ task $\Psi_{T_{\text{SO}(3)}}$ aiming at zeroing the roll and pitch angles for the torso (see Section 6.2), we deactivate such a task when crouching is enabled.

6.4 Results

In this section, we address the experimental validation of the proposed ADHERENT framework. After showing the trajectory control performances of ADHERENT on the iCub v2.7 humanoid in Section 6.4.1, we investigate the transferability of the proposed approach on different robotic platforms by porting the entire ADHERENT architecture on the iCub v3 humanoid. Given the successful results of such a porting illustrated in Section 6.4.2, we further validate the trajectory control performances of the proposed framework on both robots.

In particular, the performances of ADHERENT in terms of robustness and human-likeness are evaluated in Section 6.4.3 and Section 6.4.4, respectively, with experiments carried out on both the iCub v2.7 humanoid and the iCub v3 humanoid in each section. Finally, Section 6.4.5 shows the results of the extension of ADHERENT to crouching motions, obtained using the simulated iCub v2.7 humanoid in the Gazebo simulator [Koenig and Howard, 2004].

6.4.1 Controlled walking patterns

By combining the learning-based whole-body trajectory generator from Chapter 5 with the hierarchical humanoid locomotion controller from Section 6.2, the proposed ADHERENT architecture allows to execute on the iCub v2.7 humanoid – see Section 3.4.1 – the learning-based walking trajectories shown in a kinematic setting in Section 5.5.1. These include forward, backward, and sideways walking, with start and stops as well as smooth transitions from one to another. Depending on the walking motion, the data-driven trajectories need to be scaled in terms of walking speed and/or footstep length for a successful execution on the real robot – see Section 6.4.3 for a deeper analysis. Once the DCM plan has been computed, the simplified model and whole-body QP controllers stabilize the trajectories at 100 Hz on a 4-th generation Intel® Core i7 @ 1.7 GHz.

Figure 6.4 (bottom) illustrates the successful execution on the iCub v2.7 humanoid of the complex data-driven trajectory visualized in a kinematic setting in the upper part of the same figure. Such a trajectory starts with a forward walking, which is then oriented towards right, and is followed by a right-side walking. After a few steps on the right, the robot smoothly transitions to backward walking before finally stopping. The complete footstep sequence performed by the robot is drawn on the screenshots for the sake of clarity. Further successful trajectory executions on the iCub v2.7 humanoid are shown in a dedicated portion of the accompanying video available at <https://youtu.be/s7-pML0ojK8?t=237>.

Let us remark here the key role of the hierarchical locomotion controller from Section 6.2 for the ADHERENT framework to succeed on physics-based simulations and real-robot experiments. Despite the MANN-generated trajectory being human-like – see Section 5.5.2 – and therefore likely "close" to feasible, directly tracking the joint references from the MANN-based trajectory generator in a physics simulation leads the robot to immediately fall. This is because, especially with our position-controlled robot, joint references which correspond to even slightly-unfeasible motions have a significant impact on the overall walking performances.

For instance, it is highly likely for the MANN-based trajectory generator to learn slightly non-zero roll and pitch angles for the support foot, leading to problematic impacts with the ground or imbalance during single support – these issues are instead mitigated by replanning the swing foot trajectories as described in Section 6.1.1. Similarly, the oscillatory references for the torso roll and pitch angles learned by the MANN-based trajectory generator represent a source of imbalance for the robot, increasing therefore the probabilities of a fall – this issue is instead mitigated by the torso-related $SO(3)$ task described in Section 6.2.

In summary, we observe that in our experimental setting directly tracking the joint references from the MANN-based trajectory generator does not lead to successful walking motions, and further trajectory optimization is required to increase the overall motion feasibility (that, however, is not guaranteed by the employed DCM instantaneous controller described in Section 6.1.2). The locomotion controller from Section 6.2 plays here a role which is equivalent to that of the RL-based policy allowing for tracking data-driven posturals as best as physically possible in concurrent works, e.g., [Bergamin et al., 2019].

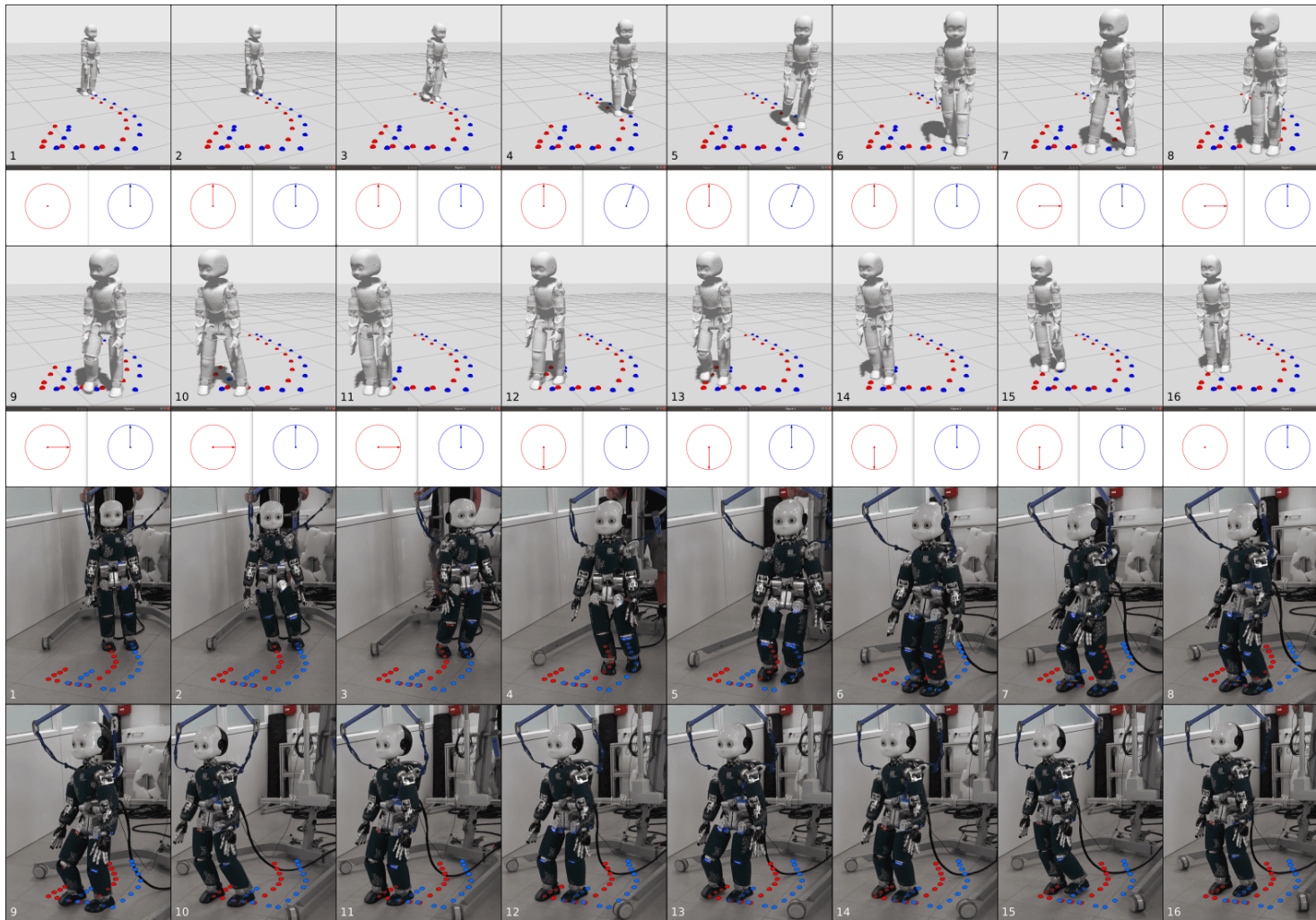


Figure 6.4. (Top) Kinematic visualization of a mixed trajectory including forward (1-3), oriented-forward (4-6), side (7-11), and backward (12-15) walking, with smooth transitions between them and a final stop (16). Below each frame, the user inputs interactively shaping the trajectory are plotted from the robot's local viewpoint (red: desired motion direction; blue: desired facing direction). (Bottom) The same trajectory performed on the iCub v2.7 humanoid. Numbering highlights frame correspondences.

Despite being successful in that the robot is able to walk without falling, the trajectories in the accompanying video at <https://youtu.be/s7-pML0ojK8?t=237> cannot be said to exhibit a high degree of agility. Both at the mechanical and at the control level, this is related to our specific validation platform.

The joints of the iCub v2.7 humanoid exploited for locomotion purposes are indeed all electrically actuated by brushless motors with harmonic drive transmissions – see Section 3.4.1. Compared to other actuation technologies such as the hydraulic actuation of Atlas² or the series elastic actuation of Valkyrie [Paine et al., 2015], such an actuation system does not provide high impact robustness, resulting therefore less adequate for dynamic locomotion [Hutter et al., 2016]. Improved agility could also be achieved by using leaf springs as Digit³ does.

Moreover, the stiffness imposed by controlling the robot in position penalizes agile motions. However, current state-of-the-art controllers for iCub v2.7 are not suitable yet for torque control of highly dynamic motions [Romualdi et al., 2020], especially due to the lack of joint torque sensors and the subsequent noisy estimation of joint torques from the force-torque sensors depicted in Figure 3.6a.

6.4.2 Transferability on a different platform

We analyze the transferability of the proposed ADHERENT framework by porting the entire pipeline onto the iCub v3 humanoid (see Section 3.4.2). With reference to the four main components of the ADHERENT architecture shown in Figure 6.1, the minimal changes required for the porting are summarized in the following:

- Dataset collection: no changes are required at this stage, since the motion capture dataset (see Section 5.2.1) does not depend on the robot.
- Retargeting: once proper human-robot frame correspondences are defined (see Figure 5.1), both WBGR (see Section 5.1.1) and its extension guaranteeing kinematic-feasibility of the base motion (see Section 5.3.1) seamlessly apply to different robots. Retargeting simply needs to be repeated on the new model.
- Trajectory generation: apart from little details such as the proper definition of the frontal robot base and chest direction, which depend on the robot base \mathcal{B} and chest \mathcal{T} frames definition (see Section 5.4.1), also the MANN-based trajectory generator does not depend on the adopted robot. However, the input and output features need to be recomputed on the new retargeted dataset and the MANN architecture needs to be trained from scratch on the new features. The user input processing and blending remains unchanged.
- Trajectory control: apart from gain tuning, also the trajectory control architecture is robot-independent. Even the tasks and priorities (see Section 6.2) are general enough to be maintained when switching platform.

In summary, the proposed ADHERENT framework is particularly suitable for being ported across different humanoids (under the basic assumption that reasonable human-robot link correspondences can be defined), although it is not optimized for this process (retraining from scratch can require a significant amount of time).

²<https://www.bostondynamics.com/atlas>

³<https://agilityrobotics.com/robots>

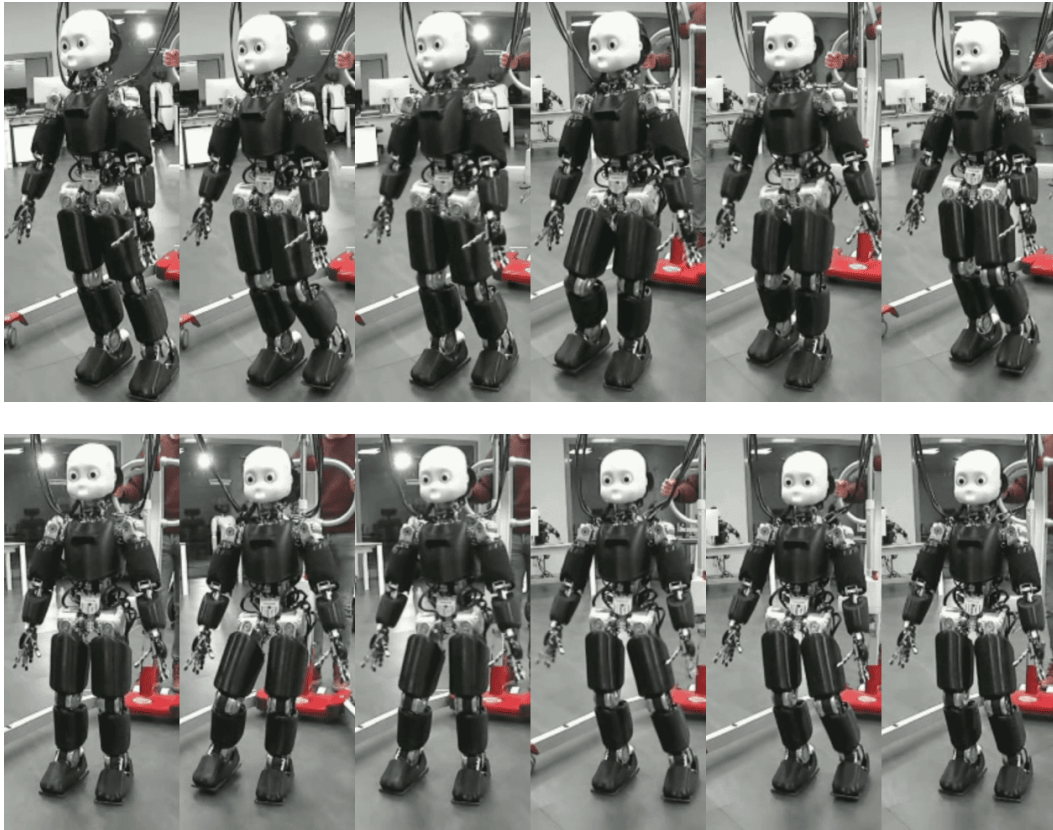


Figure 6.5. Backward (top) and right-side (bottom) walking performed on iCub v3.

As regards the specific porting of the ADHERENT framework on the iCub v3 humanoid, both at the trajectory generation level and at the trajectory control level we obtain comparable results with those illustrated for the iCub v2.7 humanoid in Section 5.5.1 and Section 6.4.1, respectively.

In terms of trajectory generation, after training with the very same parameters reported in Section 5.5 for iCub v2.7, also with iCub v3 our implementation allows an external user to interactively shape the robot trajectories for different walking patterns including forward, backward, and sideways motions, as well as starts and stops, with transitions among them smoothly handled by the network. The network prediction time remains around 3 ms on a 9-th generation Intel[®] Core i7 CPU @ 2.60 GHz, since the architecture is unchanged.

In terms of trajectory control, the proposed ADHERENT architecture allows to successfully execute on the iCub v3 humanoid all the aforementioned walking motions, including starts and stops and smooth transitions among the different motions. Also in this case, depending on the motion, a scaling in terms of walking speed and/or step length can be needed for successful executions on the real robot (see Section 6.4.3 for a deeper analysis). For the iCub v3 humanoid, the control loop runs at 100 Hz on a 11-th generation Intel[®] Core i7 @ 1.8 GHz.

Figure 6.5 illustrates the successful execution on the iCub v3 humanoid of two sample data-driven trajectories, i.e., backward (top) and right-side (bottom) walking.

6.4.3 Robustness analysis

Given the retargeted motion capture data used for training, the walking patterns learned through ADHERENT are characterized by a *nominal walking speed*⁴ v and a *nominal average step size* f . Depending on the motion style of the human subject who carried out the data collection and on the robot kinematic structure taken into account at retargeting stage, v and f may vary considerably. However, the trajectories generated at their nominal v and f , despite being kinematically-feasible, are not always successfully controlled (for several reasons, among which especially the performances of the employed high-level and low-level controllers). Particularly on real robots, they often need to be scaled both in terms of walking speed and footstep length for a successful tracking.

Therefore, we evaluate the robustness of the proposed framework on a challenging range of step sizes and walking speeds, both in simulation and on real robots. In particular, we scale the ADHERENT-generated trajectories in terms of walking speed by a factor $c_v \in \{0.25, 0.33, 0.5, 1\}$ (where 1 denotes no scaling, i.e., nominal walking speed) and in terms of footstep length by a factor $c_f \in \{0.2, 0.4, 0.6, 0.8, 1\}$ (where 1 denotes no scaling, i.e., nominal step size). A trajectory scaled by c_v and c_f is characterized by a scaled footstep size

$$\bar{f} = c_f \cdot f, \quad (6.4.1)$$

and a scaled walking speed

$$\bar{v} = c_v \cdot c_f \cdot v. \quad (6.4.2)$$

In the following, we report the results of this analysis on the simulated and real iCub v2.7 and iCub v3 humanoids. All the combinations of velocity and footstep scaling $\{c_v, c_f\}$ within the aforementioned ranges are tested on the simulated humanoids. As regards the real platforms, we restrict the analysis to a limited number of experiments, for which we also discuss the ZMP, DCM, and CoM tracking performances.

iCub v2.7 humanoid

The walking patterns learned through ADHERENT on the iCub v2.7 humanoid are characterized by:

- A nominal walking speed v of 0.525 ms^{-1} , 0.175 ms^{-1} , and 0.24 ms^{-1} for forward, backward, and side walking, respectively.
- A nominal average step size f of 28.0 cm , 19.5 cm , and 17.5 cm for forward, backward, and side walking, respectively.

Figure 6.6 summarizes the results of the robustness analysis of ADHERENT on the simulated and real iCub v2.7 humanoid. Forward, backward and sideways walking motions are considered. Concerning simulations, the green and red areas in Figure 6.6 highlight successes and failures, respectively. For each considered motion, most of the scaling combinations are successful. Regardless of c_v , the maximum admissible c_f for a successful backward and sideways walking is 0.8 and 0.6, respectively.

⁴We define the walking speed as average swing velocity, i.e., as ratio between the measured step length and its duration [Romualdi et al., 2020]. The duration of each step is obtained by summing the duration of its swing phase plus the duration of the subsequent double support (DS) phase.

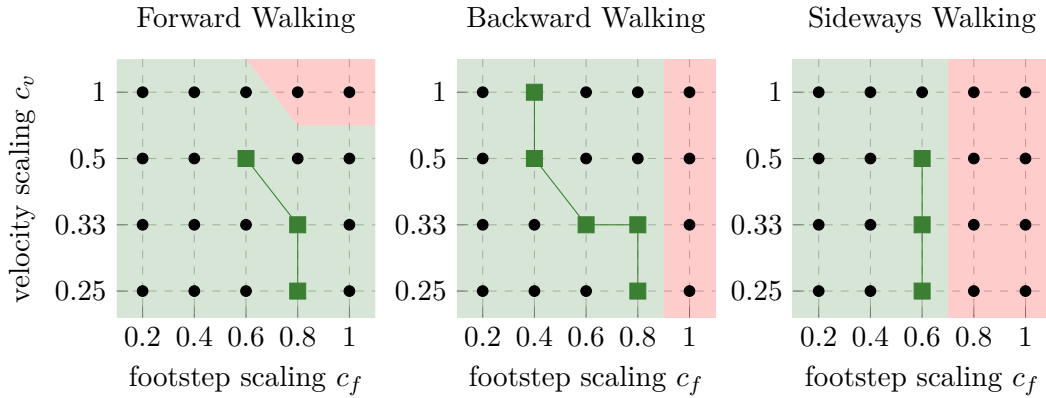


Figure 6.6. Simulated and experimental results of the robustness analysis of ADHERENT on the iCub v2.7 humanoid. For the different walking patterns, each combination $\{c_v, c_f\}$ is included in a red or green area depending on whether it resulted into a failure or a success on the simulated iCub v2.7 humanoid. The green lines connect the most challenging successes (highlighted by green squares) on the real iCub v2.7 humanoid.

As regards experiments on the real robot, the solid green lines in Figure 6.6 connect the most challenging successful ones. Notice that we did not perform an extensive tuning of the locomotion controller on the real robot, which could have reduced the mismatch between the simulated and experimental results. In particular, the most challenging successful experiments on the iCub v2.7 humanoid are characterized by:

- A walking speed \bar{v} of 0.158 ms^{-1} , 0.07 ms^{-1} , and 0.072 ms^{-1} for forward, backward, and side walking, respectively.
- An average step size \bar{f} of 16.8 cm , 7.9 cm , and 10.5 cm for forward, backward, and side walking, respectively.

The nominal walking velocities and step sizes characterizing the different walking motions are summarized, along with the walking velocities and step sizes actually achieved for the most challenging combinations of scaling factors successfully tested on the robot, in Table 6.2.

Moreover, Figure 6.7 and Figure 6.8 report the ZMP, DCM, and CoM tracking for the most challenging forward ($\{c_v = 0.5, c_f = 0.6\}$) and left-side ($\{c_v = 0.5, c_f = 0.6\}$) walking successfully executed on the iCub v2.7 humanoid. For the forward walking, the DCM and the CoM errors are kept below 8 cm and 2 cm, respectively. For the left-side walking, the DCM error reaches peaks of 12 cm but the CoM error still remains below 2 cm.

Table 6.2. Nominal and experimental velocities and step sizes for the iCub v2.7 humanoid.

	Forward		Backward		Sideways	
	nominal	$c_v = 0.5$ $c_f = 0.6$	nominal	$c_v = 1.0$ $c_f = 0.4$	nominal	$c_v = 0.5$ $c_f = 0.6$
$v \text{ (ms}^{-1}\text{)}$	0.525	0.158	0.175	0.07	0.24	0.072
$f \text{ (cm)}$	28.0	16.8	19.5	7.9	17.5	10.5

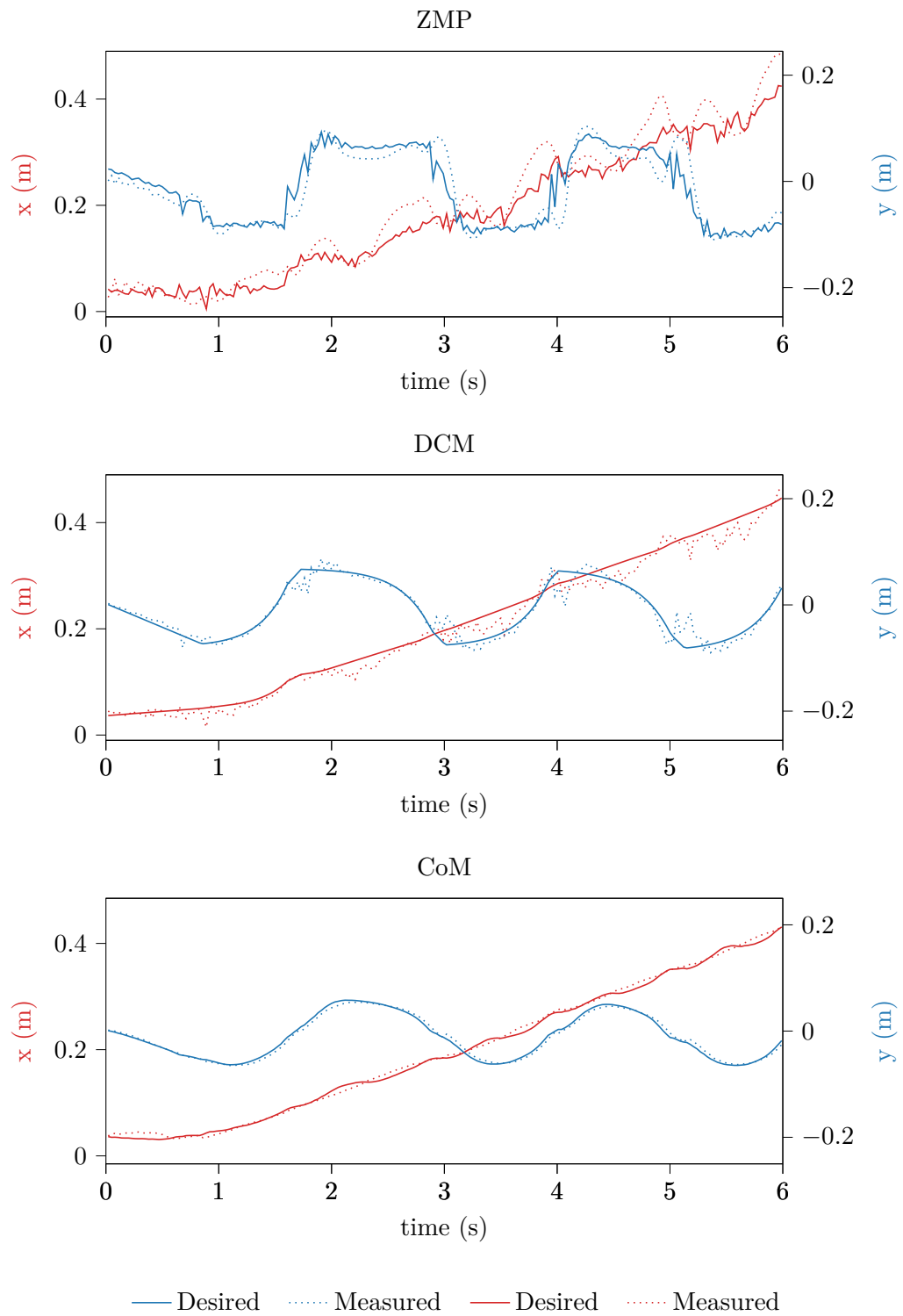


Figure 6.7. ZMP (top), DCM (center) and CoM (bottom) tracking for a *forward* walking performed on the iCub v2.7 humanoid, characterized by a velocity scaling $c_v = 0.5$ and a footstep scaling $c_f = 0.6$.

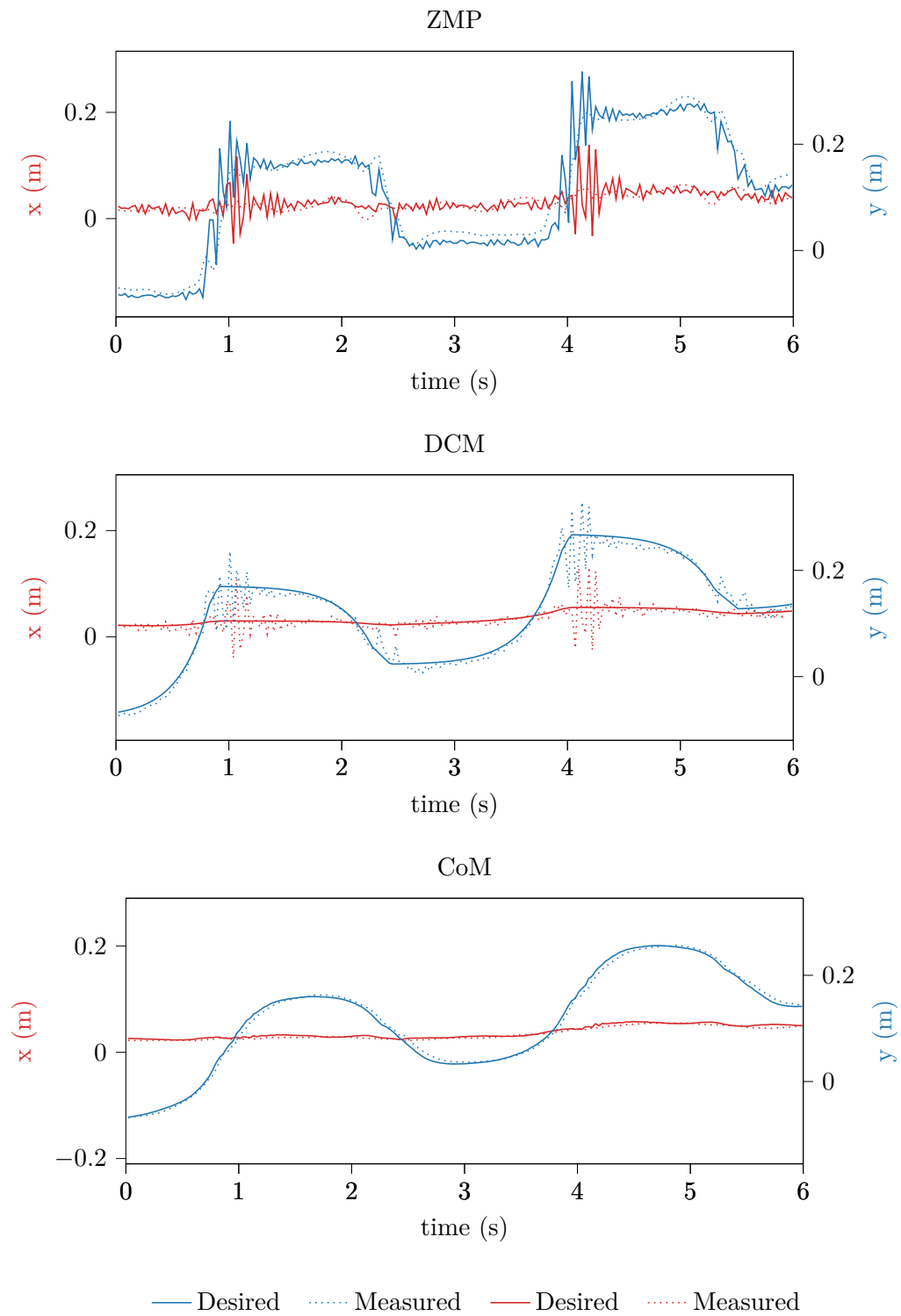


Figure 6.8. ZMP (top), DCM (center) and CoM (bottom) tracking for a *left-side* walking performed on the iCub v2.7 humanoid, characterized by a velocity scaling $c_v = 0.5$ and a footstep scaling $c_f = 0.6$.

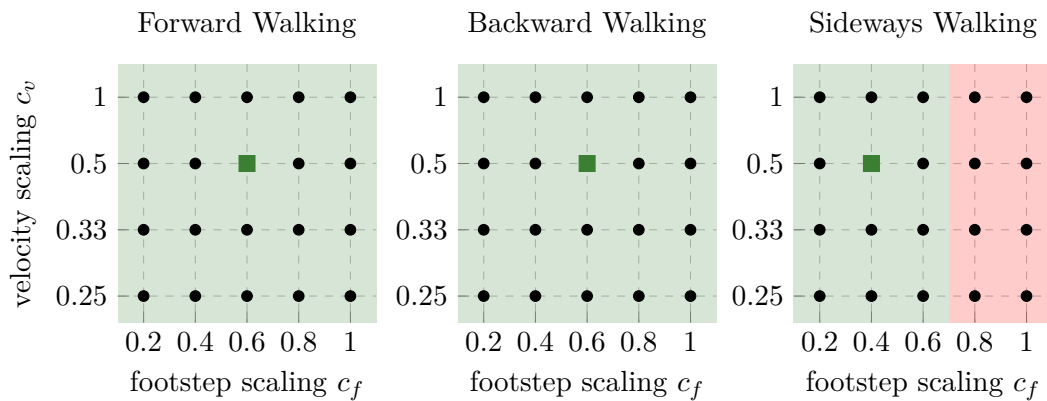


Figure 6.9. Simulated and experimental results of the robustness analysis of ADHERENT on the iCub v3 humanoid. Red and green areas denote failures and successes in simulation. The green squares highlight successful experiments on the real robot.

iCub v3 humanoid

The ADHERENT-learned walking motions on the iCub v3 are characterized by:

- A nominal walking speed v of 0.59 ms^{-1} , 0.346 ms^{-1} , and 0.34 ms^{-1} for forward, backward, and side walking, respectively.
- A nominal average step size f of 38.8 cm , 26.5 cm , and 20.8 cm for forward, backward, and side walking, respectively.

Figure 6.9 summarizes the results of the robustness analysis of ADHERENT on the simulated and real iCub v3 humanoid. Concerning simulations, for forward and backward walking the iCub v3 robot succeeds in the full range of parameters. Its performances in sideways walking are the same of the iCub v2.7 robot. As regards real experiments, the most challenging successes on iCub v3 – with no extensive tuning of the locomotion controller – are characterized by:

- A walking speed \bar{v} of 0.177 ms^{-1} , 0.104 ms^{-1} , and 0.068 ms^{-1} for forward, backward, and side walking, respectively.
- An average step size \bar{f} of 23.3 cm , 15.9 cm , and 8.3 cm for forward, backward, and side walking, respectively.

In comparison with iCub v2.7, iCub v3 achieves therefore better performances in forward and backward walking, while it is slowed down by a footstep scaling factor $c_f = 0.4$ for sideways walking. See also Table 6.3.

Table 6.3. Nominal and experimental velocities and step sizes for the iCub v3 humanoid.

	Forward		Backward		Sideways	
	nominal	$c_v = 0.5$ $c_f = 0.6$	nominal	$c_v = 0.5$ $c_f = 0.6$	nominal	$c_v = 0.5$ $c_f = 0.4$
$v \text{ (ms}^{-1}\text{)}$	0.59	0.177	0.346	0.104	0.34	0.068
$f \text{ (cm)}$	38.8	23.3	26.5	15.9	20.8	8.3

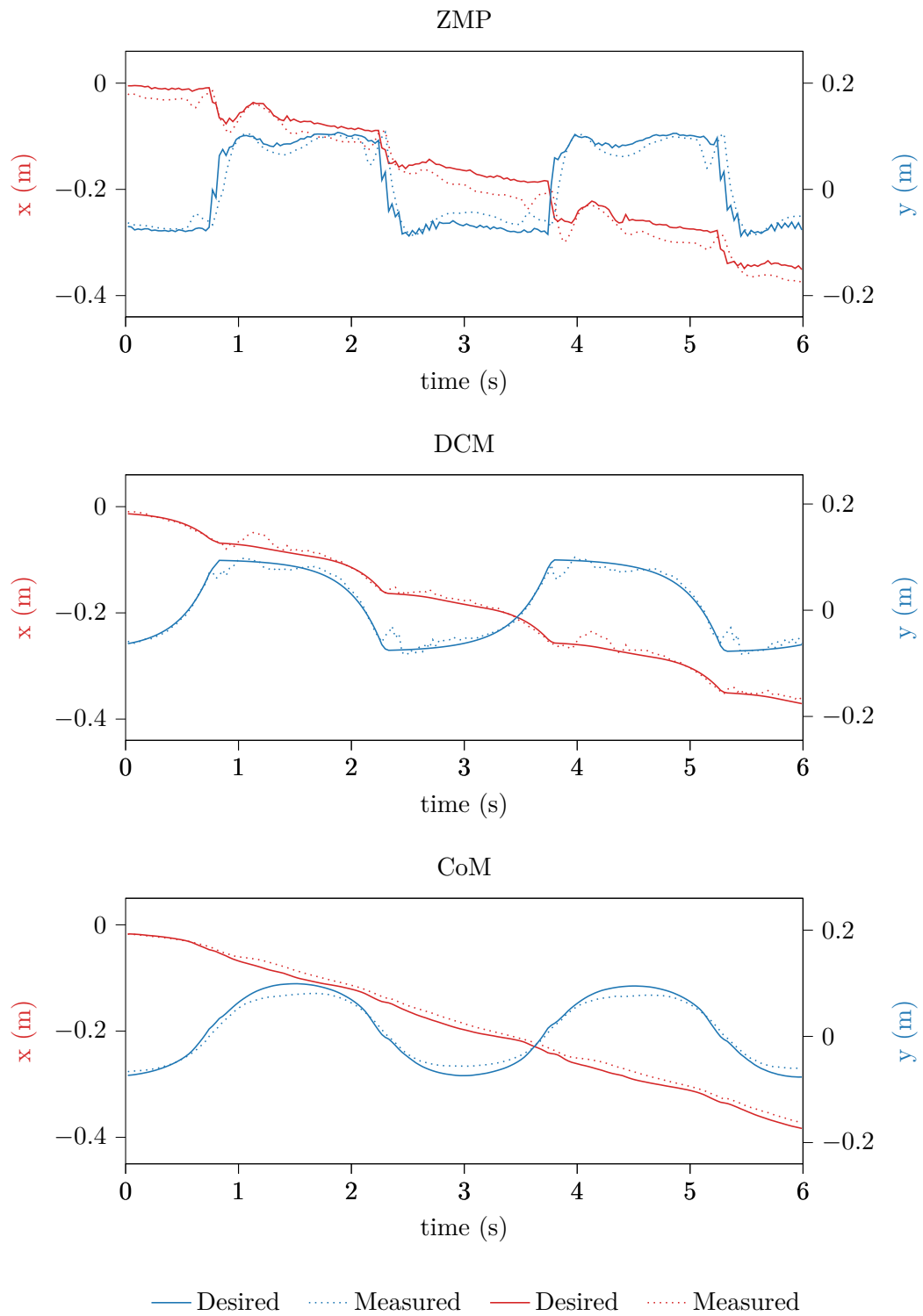


Figure 6.10. ZMP (top), DCM (center) and CoM (bottom) tracking for a *backward* walking performed on the iCub v3 humanoid, characterized by a velocity scaling $c_v = 0.5$ and a footstep scaling $c_f = 0.6$.

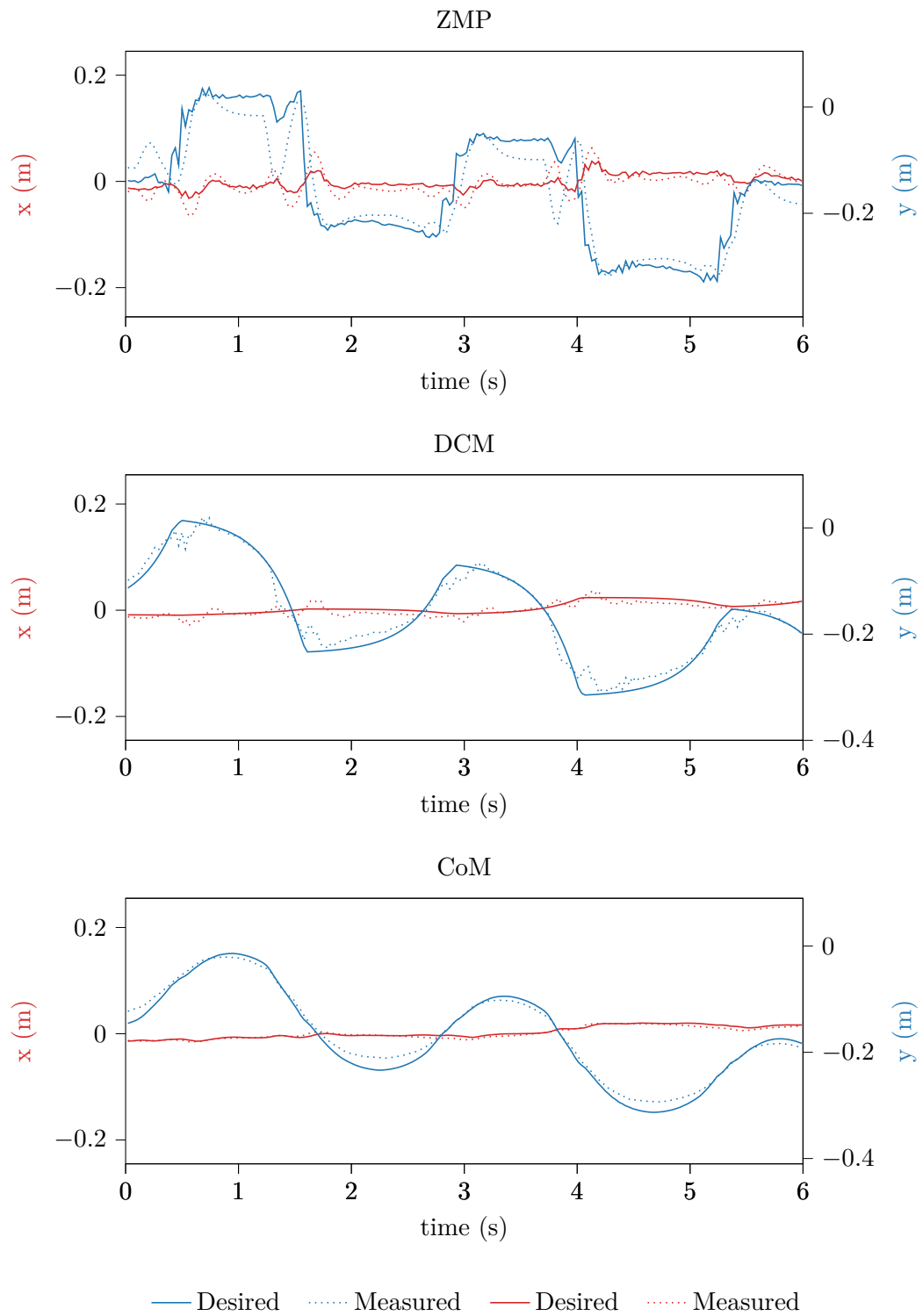


Figure 6.11. ZMP (top), DCM (center) and CoM (bottom) tracking for a *right-side* walking performed on the iCub v3 humanoid, characterized by a velocity scaling $c_v = 0.5$ and a footstep scaling $c_f = 0.4$.

Finally, Figure 6.10 and Figure 6.11 report the ZMP, DCM and CoM tracking for the most challenging backward ($\{c_v = 0.5, c_f = 0.6\}$) and right-side ($\{c_v = 0.5, c_f = 0.4\}$) walking successfully executed on the iCub v3 humanoid. In both cases, the DCM and CoM errors are kept below 6 cm and 3 cm, respectively. When comparing these results with the tracking results reported for the iCub v2.7 in Figure 6.7 and Figure 6.8, keep in mind that while the iCub v3 backward and the iCub v2.7 forward walking are characterized by the same scaling parameters, the iCub v3 right-side and the iCub v2.7 left-side walking are not (in particular, the footstep scaling is higher for iCub v3, which makes only seem performances better).

In general, we can conclude that the tracking performances among the two robots (which also strongly depend on the tuning of the high-level and low-level controllers) are comparable, and the slight differences in terms of walking velocities and step sizes mainly reflect the structural differences among the robots (see Section 3.4).

6.4.4 Human-likeness

Given the human-like postural provided by the MANN-based trajectory generator (see Section 5.5.2), we evaluate the human-likeness of the actual robot motion obtained using the ADHERENT framework by comparing the joint positions measured on the robot with those in the MANN-generated postural. Showing that the MANN-generated postural is closely tracked by the measured robot joints (despite the action of the whole-body QP controller) indicates indeed that also the final robot motion exhibits a certain degree of human-likeness.

As an additional evaluation, we compare the walking motions obtained by using the data-driven MANN-generated postural with those obtained by adopting a *fixed* postural for the upper body, as is often the case in classical humanoid robot locomotion [Righetti et al., 2011; Nava et al., 2016; Romualdi et al., 2018, 2020].

iCub v2.7 humanoid

Figure 6.12 compares, for a representative set of joints, the reference postural and the measured joint positions for two forward walking trajectories executed on the iCub v2.7 humanoid. For one of the trajectories, the data-driven MANN-generated postural is active. The good tracking performances in this case demonstrate that the reference motion produced by the trajectory generator trained on human-retargeted data is actually realized on the robot, obtaining therefore an overall *human-like* robot motion. The other trajectory uses instead a fixed postural. In this case, measured joint positions obviously oscillate in proximity of the constant reference.

This emphasizes how ADHERENT, as opposed to most state-of-the-art controllers for humanoid locomotion, exploits humanoid upper-body redundancy when tackling locomotion tasks to improve the human-likeness of the overall robot motion.

iCub v3 humanoid

Figure 6.13 presents a similar analysis (without fixed postural) for the iCub v3 humanoid. Apart from a tracking delay which remains to be investigated (possibly leading to better overall walking performances), also in this case the human-like MANN-generated postural is followed quite accurately by the measured joints.

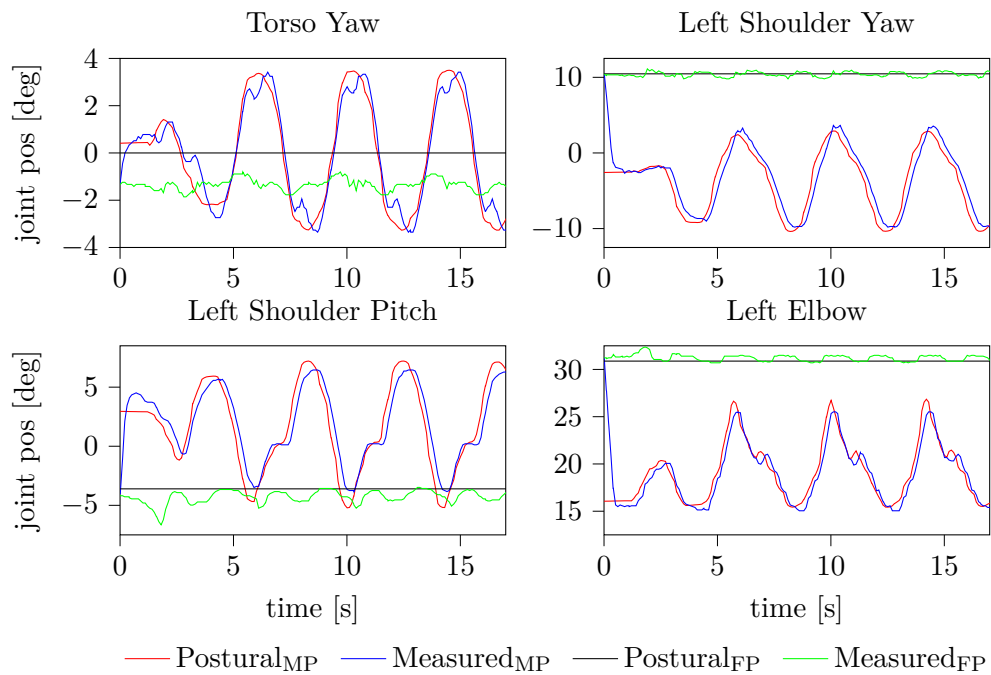


Figure 6.12. Reference postural and measured joint positions with MANN-generated Postural (MP) vs. Fixed Postural (FP) for four representative upper-body joints of the iCub v2.7 humanoid.

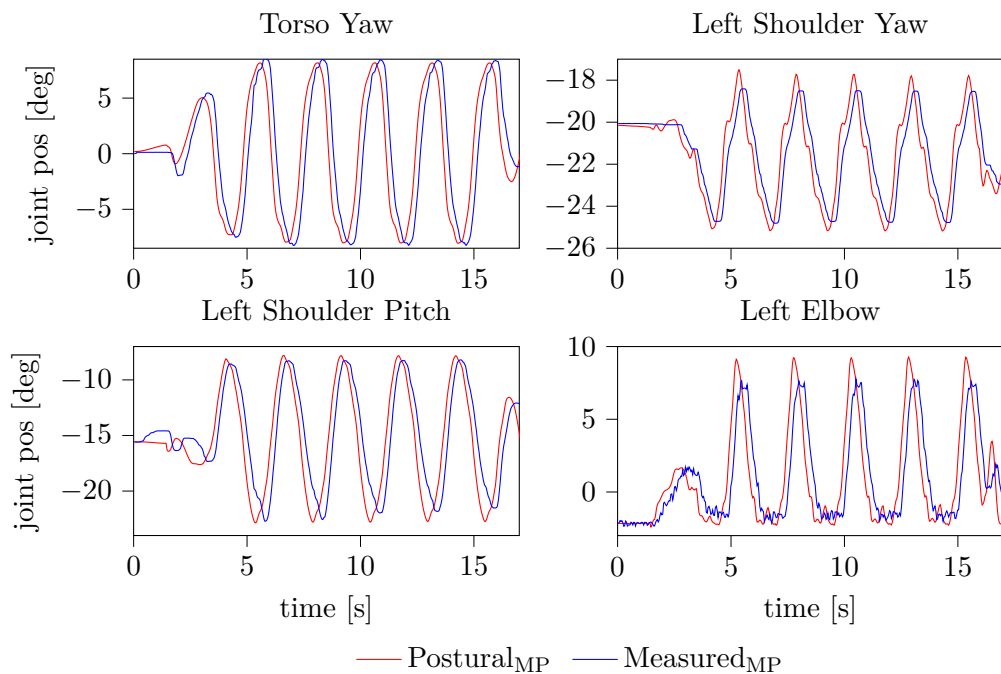


Figure 6.13. Reference postural and measured joint positions with MANN-generated Postural (MP) for four representative upper-body joints of the iCub v3 humanoid.

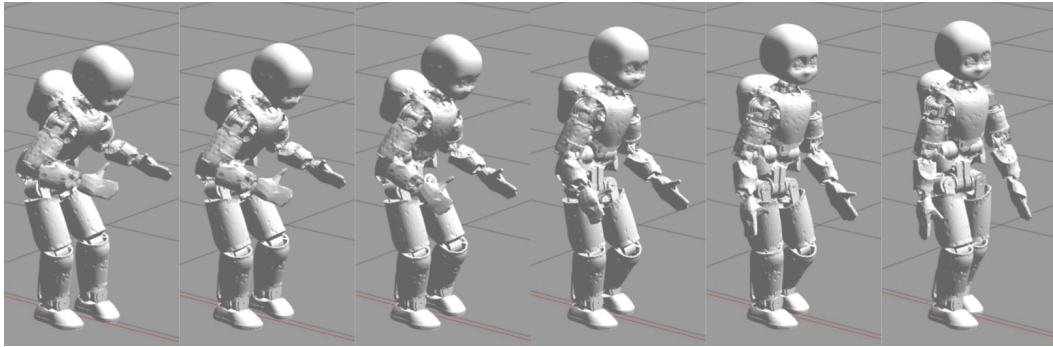


Figure 6.14. Crouching-to-upright walking transition of the iCub v2.7 simulated in Gazebo. Notice how the transition during stepping demonstrated in the training data (see Figure 6.3) is effectively learned and successfully executed.

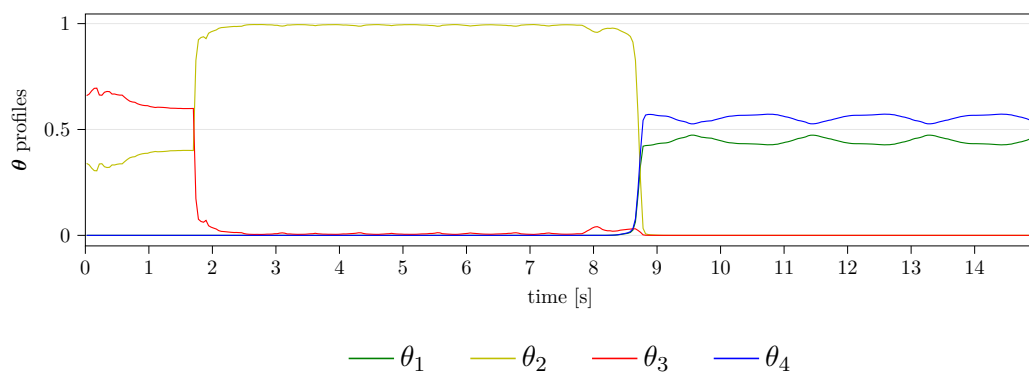


Figure 6.15. Blending coefficients θ profiles for a 15 s forward walking which includes crouching motions. The robot is standing still at the beginning (0-1.8s), walks forward upright in the middle (1.8-8.7s), and then proceeds forward while crouching (8.7-15s).

6.4.5 Crouching

Finally, we evaluate the performances of the ADHERENT framework extended to include crouching motions after training it on the motion capture dataset resulting from the combination of the collected upright (see Section 5.2) and crouching (see Section 6.3.1) trajectories. We use the very same architecture (see Section 5.4) and training parameters (see Section 5.5) of the original ADHERENT implementation, retrained from scratch to learn the additional task. In particular, we maintain $K = 4$ experts no matter the additional motions to be learned.

In terms of trajectory generation, the performances obtained with the extended ADHERENT framework are close but not equivalent to those demonstrated by the original implementation. Once trained, the extended framework is able to generate forward and backward walking while crouching, with changes of direction and smooth transitions between crouching and upright locomotion. The increased complexity of the task, however, makes it difficult to learn symmetric gaits and perform combinations such as walking sideways while crouching, for which the current implementation of the extended framework does not succeed.

Table 6.4. Comparison of the crouching features ${}^{\mathcal{B}}x_h^x$ and ${}^{\mathcal{I}}x_h^z$ averaged over a 20 s forward walking directly retargeted from human data (left) vs generated and controlled using ADHERENT (right). Notice that the x axis for iCub v2.7 points backward, and therefore the more negative ${}^{\mathcal{B}}x_h^x$ the more the head is leaning forward.

	Retargeted data		ADHERENT-controlled	
	${}^{\mathcal{B}}x_h^x$ (m)	${}^{\mathcal{I}}x_h^z$ (m)	${}^{\mathcal{B}}x_h^x$ (m)	${}^{\mathcal{I}}x_h^z$ (m)
Crouching	-0.1790	0.7092	-0.1405	0.7808
Upright	-0.0643	0.8592	-0.0540	0.8435
Difference	0.1147	0.1500	0.0970	0.0663

Compared to the original implementation, the $K = 4$ experts specialize differently. Figure 6.15 shows the profiles of the blending coefficients θ determining the experts activations for a forward upright walking followed by a forward walking while crouching. In this case, θ_2 only activates during the former, while alternate periodic activations of θ_1 and θ_4 characterize the latter. Not having a periodic alternation of experts for forward upright walking, as opposed to the original implementation (see Figure 5.11), would motivate the idea of increasing the number of experts for better performance here, although that would also increase the training time.

Concerning trajectory control performances, we observe that, if the postural task $\Psi_{T_{SO(3)}}$ is deactivated while crouching (see Section 6.3.2), the trajectory controller is able to track the ADHERENT-generated references while maintaining balance. Also smooth transitions between upright and crouching walking without stopping, such as the one shown in Figure 6.14, are successfully executed. Further results of trajectories including crouching controlled on the simulated iCub v2.7 are shown in a dedicated video available at <https://youtu.be/Dor1hMqAAmo>. Given the bottleneck at the trajectory generation level (asymmetric gaits, inability to generate sideways walking while crouching) which leads to trajectories at least occasionally challenging to track, we decide not to perform an experimental validation of the crouching-extended ADHERENT framework on real robots. As future work, we plan to further investigate solutions (e.g., use more experts) which mitigate the current issues and enable proper experimental validation.

As an additional evaluation of the learned model performances in reproducing the crouching ability demonstrated in the training data, Table 6.4 compares the selected head-related crouching features ${}^{\mathcal{B}}x_h^x$ and ${}^{\mathcal{I}}x_h^z$ (see Section 6.3) for a sample upright and crouching forward walking from the retargeted data and a trajectory controlled on the simulated iCub v2.7 humanoid. It can be noticed that the head position difference between crouching and upright walking, from the retargeted data to the trajectory control, is in the same range for ${}^{\mathcal{B}}x_h^x$ but not for ${}^{\mathcal{I}}x_h^z$. In other words, the final crouching motion obtained on the robot is characterized by a head height that is not as low as the one demonstrated by the human in the training crouching trajectories. This could be related to the action of the low-level whole-body QP controller on the reference postural as well as to the footstep scaling applied at trajectory control stage. Despite lowering the head less than the training data, the controlled crouching trajectories properly exhibit torso leaning and subsequent head lowering, as shown in the initial frames of Figure 6.14.

6.5 Conclusions

In this chapter, we present ADHERENT, an end-to-end architecture for whole-body trajectory generation and control of humanoid robots. ADHERENT joins together the advantages of learning-based trajectory generation and state-of-the-art hierarchical locomotion control. By doing so, it allows for *human-like* whole-body motions, more predictable and interpretable for humans, to be *efficiently* computed and executed on complex humanoids.

We demonstrate the robustness of ADHERENT through an extensive validation in simulation and real experiments on both the iCub v2.7 and the iCub v3 humanoid robots, showing how, as long as meaningful correspondences between the human and the robot links can be defined to retarget the training motion capture dataset, the proposed approach can be easily transferred to other humanoids. Moreover, through the use case of crouching motions, we investigate the modularity of ADHERENT to the integration of additional skills.

The proposed approach exhibits several limitations that we would like to address in future works. First, when controlling the data-driven trajectories on real robots, a scaling of the walking velocity and step size is required. This could be due to the limitations of the simple instantaneous controllers employed in the current ADHERENT implementation, that we would like to replace with more advanced MPC-based or RL-based controllers in future implementations, hopefully enabling to track higher speeds and longer steps without losing balance. Secondly, the proposed approach assumes flat terrains. By exploiting motion capture data collected on uneven terrains and including measurements perceived from the external environment in the input and output features for the MANN-based trajectory generator, we would like to relax this assumption and target data-driven locomotion in more complex terrains. Finally, our learning-based generator needs to be trained from scratch every time a new motion skill or a new robot is targeted. As a future work, we would like to investigate network architectural changes as well as continual/lifelong learning methods to increase the modularity of the proposed architecture.

Epilogue

This thesis attempted to investigate the application of different learning-based approaches to planning and control of humanoid robot locomotion, by combining methods that proved their effectiveness in different research areas. Part I contains indeed an overview of the state-of-the-art relevant for this work from both robotics and computer graphics research. In particular, the literature related to bipedal locomotion in robotics and character animation in computer graphics is reviewed, trying to highlight the many contact points between these two fields. Both humanoid robots, in the physical world, and humanoid characters, in the virtual world, are expected to resemble, to a certain degree, the human beings they are inspired to. As a result, the correspondent research areas share a certain number of issues and potential solutions, despite their research being driven by different targets, namely physical applications in robotics and virtual applications in computer graphics. Part I also includes the fundamental background required for a correct interpretation of the methods developed in the thesis.

Part II presents instead the learning-based methods proposed in the thesis, along with their validation on simulated and real robots. Motivated by the curiosity of investigating the emergence of learning-based control policies for humanoid locomotion, in Chapter 4 we design a deep reinforcement learning agent for learning balancing and push-recovery strategies, fundamental skills for more complex locomotion tasks. To guide the agent towards the development of effective whole-body strategies, we shape the reward function using domain knowledge in humanoid control. Validated on a simulated version of the iCub v2.7 humanoid, the proposed approach proves robust in generating highly-dynamic whole-body policies to counteract even significantly out-of-sample external pushes. However, during validation we observe the emergence of robot motions which, despite being effective for the simulated humanoid, look unnatural and above all far from being feasible.

Rather than tackling walking (as a natural extension of the balancing task) using deep reinforcement learning, inspired by the impressive performances of character animation systems generating natural-looking behaviours from motion capture data, in Chapter 5 we introduce a whole-body trajectory generator for humanoid locomotion that exploits deep supervised learning. After collecting a rich and wide-ranging motion capture dataset, properly retargeted on the selected robot, we train our trajectory generator on suitable features extracted from the retargeted dataset, thus enabling an interactive trajectory generation with minimal user inputs. Validated in a kinematic setting on the iCub v2.7 humanoid model, the proposed approach proves capable of efficiently producing diverse walking patterns and smooth transitions among them, also exhibiting a certain degree of human-likeness.

The performances of the learning-based trajectory generator presented in Chapter 5, although limited to flat terrains, motivate us to integrate it with a state-of-the-art locomotion controller for humanoid robots. Chapter 6 introduces the end-to-end architecture, named ADHERENT, resulting from such an integration. Within ADHERENT, the footstep plan and the whole-body postural produced by our learning-based trajectory generator are exploited by the locomotion controller to improve the human-likeness of the overall robot motion. We experimentally validate ADHERENT on both the iCub v2.7 and the iCub v3 humanoids, thus also proving the transferability of our architecture across different platforms.

In summary, satisfactory experiments on the real robot could only be achieved when integrating the proposed learning-based methods with state-of-the-art hierarchical control architectures for humanoid locomotion. Having enough feasibility guarantees to adopt on the real platform purely learning-based approaches, such as the RL-based control policy for balancing and push-recovery, remains an open question. Even the approach that proves successful on the real robot (i.e., supervised-learning-based trajectory generation combined with model-based trajectory control) does not allow for the execution of the data-driven walking trajectories without scaling their velocity or footstep length. We believe that this is mainly related to the simple instantaneous controllers integrated, in the current implementation, with the learning-based trajectory generator. As a future work, we would like to replace them with more advanced controllers, e.g., MPC-based ones, to fully exploit the potential of the human-driven trajectories.

In the Prologue, we mentioned that humanoid locomotion, as a general problem, is still far from being a trivial task, and that its solution could be addressed from several perspectives. The attempt made in this thesis to make the best of both the model-free learning-based approach and the model-based hierarchical approach is only a preliminary step, still suffering from many limitations. Nevertheless, we hope that the work presented in this thesis may inspire further developments towards more natural behaviours for humanoid robots at no expense (actually, in favour) of their robustness and safety guarantees.

Bibliography

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: a system for large-scale machine learning. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*.
- Aceituno-Cabezas, B., Mastalli, C., Dai, H., Focchi, M., Radulescu, A., Caldwell, D. G., Cappelletto, J., Grieco, J. C., Fernández-López, G., and Semini, C. (2018). Simultaneous Contact, Gait, and Motion Planning for Robust Multilegged Locomotion via Mixed-Integer Convex Optimization. *IEEE Robotics and Automation Letters*, 3(3):2531–2538.
- Agrawal, S. and van de Panne, M. (2016). Task-based locomotion. *ACM Transactions on Graphics*, 35(4):82:1–82:11.
- Amari, S.-i. (1998). Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276.
- Apicella, A., Donnarumma, F., Isgrò, F., and Prevete, R. (2021). A survey on modern trainable activation functions. *Neural Networks*, 138:14–32.
- Arakawa, T. and Fukuda, T. (1997). Natural motion generation of biped locomotion robot using hierarchical trajectory generation method consisting of GA, EP layers. In *Proceedings of International Conference on Robotics and Automation*, volume 1, pages 211–216.
- Arikan, O. and Forsyth, D. A. (2002). Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490.
- Beaudoin, P., Coros, S., van de Panne, M., and Poulin, P. (2008). Motion-motif graphs. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.
- Bergamin, K., Clavet, S., Holden, D., and Forbes, J. R. (2019). DReCon: data-driven responsive control of physics-based characters. *ACM Transactions on Graphics*, 38(6):1–11.
- Bin Peng, X., Coumans, E., Zhang, T., Lee, T.-W., Tan, J., and Levine, S. (2020). Learning Agile Robotic Locomotion Skills by Imitating Animals. In *Robotics: Science and Systems XVI*.

- Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- Bloesch, M., Humplik, J., Patraucean, V., Hafner, R., Haarnoja, T., Byravan, A., Siegel, N. Y., Tunyasuvunakool, S., Casarini, F., Batchelor, N., Romano, F., Saliceti, S., Riedmiller, M., Eslami, S. M. A., and Heess, N. (2022). Towards Real Robot Learning in the Wild: A Case Study in Bipedal Locomotion. In *Proceedings of the 5th Conference on Robot Learning*. ISSN: 2640-3498.
- Bombile, M. and Billard, A. (2017). Capture-point based balance and reactive omnidirectional walking controller. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics*.
- Borst, C., Wimböck, T., Schmidt, F., Fuchs, M., Brunner, B., Zacharias, F., Giordano, P., Konietzschke, R., Sepp, W., Fuchs, S., Rink, C., Albu-Schäeffler, A., and Hirzinger, G. (2009). Rollin' Justin - Mobile Platform with Variable Base.
- Bouyarmane, K. and Kheddar, A. (2018). On Weight-Prioritized Multitask Control of Humanoid Robots. *IEEE Transactions on Automatic Control*, 63(6):1632–1647.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym.
- Budhiraja, R., Carpentier, J., Mastalli, C., and Mansard, N. (2018). Differential Dynamic Programming for Multi-Phase Rigid Contact Dynamics. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*.
- Cannata, G., Maggiali, M., Metta, G., and Sandini, G. (2008). An embedded artificial skin for humanoid robots. In *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*.
- Caron, S. and Kheddar, A. (2016). Multi-contact walking pattern generation based on model preview control of 3D COM accelerations. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.
- Caron, S. and Kheddar, A. (2017). Dynamic walking over rough terrains by nonlinear predictive control of the floating-base inverted pendulum. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Caron, S. and Pham, Q.-C. (2017). When to make a step? Tackling the timing problem in multi-contact locomotion by TOPP-MPC. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*.
- Caron, S., Pham, Q.-C., and Nakamura, Y. (2015). Stability of Surface Contacts for Humanoid Robots: Closed-Form Formulae of the Contact Wrench Cone for Rectangular Support Areas. In *Proceedings - IEEE International Conference on Robotics and Automation*.
- Caron, S., Pham, Q.-C., and Nakamura, Y. (2017). ZMP Support Areas for Multi-contact Mobility Under Frictional Constraints. *IEEE Transactions on Robotics*, 33(1):67–80.

- Carpentier, J., Del Prete, A., Tonneau, S., Flayols, T., Forget, F., Mifsud, A., Giraud-Esclasse, K., Atchuthan, D., Fernbach, P., Budhiraja, R., Geisert, M., Stasse, O., Mansard, N., and Solà, J. (2017). Multi-contact Locomotion of Legged Robots in Complex Environments – The Loco3D project. *RSS Work- shop on Challenges in Dynamic Legged Locomotion*.
- Carpentier, J., Tonneau, S., Naveau, M., Stasse, O., and Mansard, N. (2016). A versatile and efficient pattern generator for generalized legged locomotion. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*.
- Castillo, G. A., Weng, B., Zhang, W., and Hereid, A. (2020). Hybrid Zero Dynamics Inspired Feedback Control Policy Design for 3D Bipedal Locomotion using Reinforcement Learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*.
- Castillo, G. A., Weng, B., Zhang, W., and Hereid, A. (2021). Robust Feedback Motion Policy Design Using Reinforcement Learning on a 3D Digit Bipedal Robot. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Chai, J. and Hodgins, J. K. (2005). Performance animation from low-dimensional control signals. *ACM Transactions on Graphics*, 24(3):686–696.
- Chai, J. and Hodgins, J. K. (2007). Constraint-based motion optimization using a statistical dynamic model. *ACM Transactions on Graphics*, 26(3):8–es.
- Chentanez, N., Müller, M., Macklin, M., Makoviychuk, V., and Jeschke, S. (2018). Physics-based motion capture imitation with deep reinforcement learning. In *Proceedings of the 11th ACM SIGGRAPH Conference on Motion, Interaction and Games*.
- Choi, Y., Kim, D., and You, B.-J. (2006). On the walking control for humanoid robot based on the kinematic resolution of CoM Jacobian with embedded motion. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. ISSN: 1050-4729.
- Clavet, S. (2016). Motion Matching and The Road to Next-Gen Animation. *Proc. of GDC*.
- Clever, D., Harant, M., Mombaur, K., Naveau, M., Stasse, O., and Endres, D. (2017). COCoMoPL: A Novel Approach for Humanoid Walking Generation Combining Optimal Control, Movement Primitives and Learning and its Transfer to the Real Robot HRP-2. *IEEE Robotics and Automation Letters*, 2(2):977–984.
- Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *4th International Conference on Learning Representations*.
- Cognetti, M., De Simone, D., Lanari, L., and Oriolo, G. (2016). Real-time planning and execution of evasive motions for a humanoid robot. In *2016 IEEE International Conference on Robotics and Automation*.

- Coros, S., Beaudoin, P., and van de Panne, M. (2009). Robust task-based control policies for physics-based characters. *ACM Transactions on Graphics*, 28(5):1–9.
- Coros, S., Beaudoin, P., and van de Panne, M. (2010). Generalized biped walking control. In *ACM SIGGRAPH 2010 papers*.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.
- Da Silva, M., Abe, Y., and Popović, J. (2008). Simulation of Human Motion Data using Short-Horizon Model-Predictive Control. *Computer Graphics Forum*, 27(2):371–380.
- Dafarra, S., Darvish, K., Grieco, R., Milani, G., Pattacini, U., Rapetti, L., Romualdi, G., Salvi, M., Scalzo, A., Sorrentino, I., Tomè, D., Traversaro, S., Valli, E., Viceconte, P. M., Metta, G., Maggiali, M., and Pucci, D. (2022a). iCub3 Avatar System. arXiv:2203.06972.
- Dafarra, S., Nava, G., Charbonneau, M., Guedelha, N., Andrade Chavez, F., Traversaro, S., Fiorio, L., Romano, F., Nori, F., Metta, G., and Pucci, D. (2018). A Control Architecture with Online Predictive Planning for Position and Torque Controlled Walking of Humanoid Robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Dafarra, S., Romano, F., and Nori, F. (2016). Torque-controlled stepping-strategy push recovery: Design and implementation on the iCub humanoid robot. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.
- Dafarra, S., Romualdi, G., Metta, G., and Pucci, D. (2020). Whole-Body Walking Generation using Contact Parametrization: A Non-Linear Trajectory Optimization Approach. In *IEEE International Conference on Robotics and Automation*.
- Dafarra, S., Romualdi, G., and Pucci, D. (2022b). Dynamic Complementarity Conditions and Whole-Body Trajectory Optimization for Humanoid Robot Locomotion. *IEEE Transactions on Robotics*, 38(6):3414–3433.
- Dai, H., Valenzuela, A., and Tedrake, R. (2014). Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*.
- Dantec, E., Budhiraja, R., Roig, A., Lembono, T., Saurel, G., Stasse, O., Fernbach, P., Tonneau, S., Vijayakumar, S., Calinon, S., Taix, M., and Mansard, N. (2021). Whole Body Model Predictive Control with a Memory of Motion: Experiments on a Torque-Controlled Talos. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X.
- Darvish, K., Tirupachuri, Y., Romualdi, G., Rapetti, L., Ferigo, D., Chavez, F. J. A., and Pucci, D. (2019). Whole-Body Geometric Retargeting for Humanoid Robots. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*.

- De Simone, D., Scianca, N., Ferrari, P., Lanari, L., and Oriolo, G. (2017). MPC-based humanoid pursuit-evasion in the presence of obstacles. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Deisenroth, M. P., Neumann, G., and Peters, J. (2013). A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142.
- Deits, R. and Tedrake, R. (2014). Footstep planning on uneven terrain with mixed-integer convex optimization. In *2014 IEEE-RAS International Conference on Humanoid Robots*.
- Diedam, H., Dimitrov, D., Wieber, P.-B., Mombaur, K., and Diehl, M. (2008). Online walking gait generation with adaptive foot positioning through Linear Model Predictive control. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Dubey, S. R., Singh, S. K., and Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.
- Engelsberger, J., Koolen, T., Bertrand, S., Pratt, J., Ott, C., and Albu-Schäffer, A. (2014). Trajectory generation for continuous leg forces during double support and heel-to-toe shift based on divergent component of motion. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Engelsberger, J., Mesesan, G., Ott, C., and Albu-Schäffer, A. (2018a). DCM-Based Gait Generation for Walking on Moving Support Surfaces. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*.
- Engelsberger, J., Mesesan, G., Werner, A., and Ott, C. (2018b). Torque-Based Dynamic Walking - A Long Way from Simulation to Experiment. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*.
- Engelsberger, J., Ott, C., and Albu-Schäffer, A. (2013). Three-dimensional bipedal walking control using Divergent Component of Motion. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Engelsberger, J., Ott, C., and Albu-Schäffer, A. (2015). Three-Dimensional Bipedal Walking Control Based on Divergent Component of Motion. *IEEE Transactions on Robotics*, 31(2):355–368.
- Engelsberger, J., Ott, C., Roa, M. A., Albu-Schäffer, A., and Hirzinger, G. (2011). Bipedal walking control based on Capture Point dynamics. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Fahmi, S., Mastalli, C., Focchi, M., and Semini, C. (2019). Passive Whole-Body Control for Quadruped Robots: Experimental Validation Over Challenging Terrain. *IEEE Robotics and Automation Letters*, 4(3):2553–2560.

- Faragasso, A., Oriolo, G., Paolillo, A., and Vendittelli, M. (2013). Vision-based corridor navigation for humanoid robots. In *2013 IEEE International Conference on Robotics and Automation*.
- Featherstone, R. (2014). *Rigid Body Dynamics Algorithms*. Springer.
- Feng, S., Whitman, E., Xinjilefu, X., and Atkeson, C. G. (2015). Optimization-based Full Body Control for the DARPA Robotics Challenge. *Journal of Field Robotics*, 32(2):293–312.
- Ferigo, D., Traversaro, S., Metta, G., and Pucci, D. (2020). Gym-Ignition: Reproducible Robotic Simulations for Reinforcement Learning. In *2020 IEEE/SICE International Symposium on System Integration (SII)*.
- Fernbach, P., Tonneau, S., and Taïx, M. (2018). CROC: Convex Resolution of Centroidal Dynamics Trajectories to Provide a Feasibility Criterion for the Multi Contact Planning Problem. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Ficht, G. and Behnke, S. (2021). Bipedal Humanoid Hardware Design: a Technology Review. *Current Robotics Reports*, 2(2):201–210.
- Fragkiadaki, K., Levine, S., Felsen, P., and Malik, J. (2015). Recurrent Network Models for Human Dynamics. In *2015 IEEE International Conference on Computer Vision (ICCV)*.
- Fumagalli, M., Ivaldi, S., Randazzo, M., Natale, L., Metta, G., Sandini, G., and Nori, F. (2012). Force feedback exploiting tactile and proximal force/torque sensing. *Autonomous Robots*, 33(4):381–398.
- Geijtenbeek, T. and Pronost, N. (2012). Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. *Computer Graphics Forum*, 31(8):2492–2515.
- Giraud-Esclasse, K., Fernbach, P., Buondonno, G., Mastalli, C., and Stasse, O. (2020). Motion Planning with Multi-Contact and Visual Servoing on Humanoid Robots. In *2020 IEEE/SICE International Symposium on System Integration (SII)*.
- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*.

- Griffin, R. J. and Leonessa, A. (2016). Model predictive control for dynamic footstep adjustment using the divergent component of motion. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*.
- Griffin, R. J., Wiedebach, G., Bertrand, S., Leonessa, A., and Pratt, J. (2017). Walking stabilization using step timing and location adjustment on the humanoid robot, Atlas. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Grochow, K., Martin, S. L., Hertzmann, A., and Popović, Z. (2004). Style-based inverse kinematics. *ACM Transactions on Graphics*, 23(3):522–531.
- Habibie, I., Holden, D., Schwarz, J., Yearsley, J., and Komura, T. (2017). A Recurrent Variational Autoencoder for Human Motion Synthesis. In *Proceedings of the British Machine Vision Conference 2017*.
- Harvey, F. G. and Pal, C. (2018). Recurrent transition networks for character locomotion. In *SIGGRAPH Asia 2018 Technical Briefs*.
- Harvey, F. G., Yurick, M., Nowrouzezahrai, D., and Pal, C. (2020). Robust Motion In-betweening. *ACM Transactions on Graphics*, 39(4).
- Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M., and Silver, D. (2017). Emergence of Locomotion Behaviours in Rich Environments.
- Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*.
- Herzog, A., Rotella, N., Schaal, S., and Righetti, L. (2015). Trajectory generation for multi-contact momentum control. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots*.
- Hirai, K., Hirose, M., Haikawa, Y., and Takenaka, T. (1998). The development of Honda humanoid robot. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 2, pages 1321–1326. ISSN: 1050-4729.
- Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*.
- Hof, A. L. (2008). The ‘extrapolated center of mass’ concept suggests a simple control of balance in walking. *Human Movement Science*, 27(1):112–125.
- Holden, D., Kanoun, O., Perepichka, M., and Popa, T. (2020). Learned motion matching. *ACM Transactions on Graphics*, 39(4).
- Holden, D., Komura, T., and Saito, J. (2017). Phase-functioned neural networks for character control. *ACM Transactions on Graphics*, 36(4):1–13.

- Holden, D., Saito, J., and Komura, T. (2016). A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics*, 35(4):138:1–138:11.
- Holden, D., Saito, J., Komura, T., and Joyce, T. (2015). Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*.
- Hong, S., Han, D., Cho, K., Shin, J. S., and Noh, J. (2019). Physics-based full-body soccer motion control for dribbling and shooting. *ACM Transactions on Graphics*, 38(4):74:1–74:12.
- Hopkins, M. A., Hong, D. W., and Leonessa, A. (2014). Humanoid locomotion on uneven terrain using the time-varying divergent component of motion. In *2014 IEEE-RAS International Conference on Humanoid Robots*. ISSN: 2164-0580.
- Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.
- Hutter, M., Gehring, C., Jud, D., Lauber, A., Bellicoso, C. D., Tsounis, V., Hwangbo, J., Bodie, K., Fankhauser, P., Bloesch, M., Diethelm, R., Bachmann, S., Melzer, A., and Hoepflinger, M. (2016). ANYmal - a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872.
- Ibanez, A., Bidaud, P., and Padois, V. (2014). Emergence of humanoid walking behaviors from mixed-integer model predictive control. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Iwasaki, T., Venture, G., and Yoshida, E. (2012). Identification of the inertial parameters of a humanoid robot using grounded sole link. In *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. ISSN: 2164-0580.
- Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87.
- Jeong, H., Lee, I., Oh, J., Lee, K. K., and Oh, J.-H. (2019). A Robust Walking Controller Based on Online Optimization of Ankle, Hip, and Stepping Strategies. *IEEE Transactions on Robotics*, 35(6):1367–1386.
- Joe, H.-M. and Oh, J.-H. (2018). Balance recovery through model predictive control based on capture point dynamics for biped walking robot. *Robotics and Autonomous Systems*, 105:1–10.
- Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6(2):181–214.

- Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa, H. (2003). Biped walking pattern generation by using preview control of zero-moment point. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1620–1626.
- Kajita, S., Kanehiro, F., Kaneko, K., Yokoi, K., and Hirukawa, H. (2001). The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern generation. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 1, pages 239–246.
- Kakade, S. M. (2001). A Natural Policy Gradient. In *Advances in Neural Information Processing Systems*, volume 14.
- Kamioka, T., Kaneko, H., Takenaka, T., and Yoshiike, T. (2018). Simultaneous Optimization of ZMP and Footsteps Based on the Analytical Solution of Divergent Component of Motion. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*.
- Khadij, M., Herzog, A., Moosavian, S. A. A., and Righetti, L. (2016). Step timing adjustment: A step toward generating robust gaits. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.
- Kim, H., Seo, D., and Kim, D. (2019). Push Recovery Control for Humanoid Robot Using Reinforcement Learning. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*.
- Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference for Learning Representations*.
- Koenig, N. and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Koolen, T., de Boer, T., Rebula, J., Goswami, A., and Pratt, J. (2012). Capturability-Based Analysis and Control of Legged Locomotion, Part 1: Theory and Application to Three Simple Gait Models. *The International Journal of Robotics Research*, 31(9):1094–1113.
- Koolen, T., Posa, M., and Tedrake, R. (2016). Balance control using center of mass height variation: Limitations imposed by unilateral contact. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.
- Kovar, L., Gleicher, M., and Pighin, F. (2002). Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.
- Latella, C. (2019). *Human Whole-Body Dynamics Estimation for Enhancing Physical Human-Robot Interaction*. PhD thesis.

- Latella, C., Lorenzini, M., Lazzaroni, M., Romano, F., Traversaro, S., Akhras, M. A., Pucci, D., and Nori, F. (2019). Towards real-time whole-body human dynamics estimation through probabilistic sensor fusion algorithms. *Autonomous Robots*, 43(6):1591–1603.
- Lee, J., Chai, J., Reitsma, P. S. A., Hodgins, J. K., and Pollard, N. S. (2002). Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics*, 21(3):491–500.
- Lee, J., Grey, M. X., Ha, S., Kunz, T., Jain, S., Ye, Y., Srinivasa, S. S., Stilman, M., and Liu, C. K. (2018a). DART: Dynamic Animation and Robotics Toolkit. *Journal of Open Source Software*, 3(22):500.
- Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986.
- Lee, K., Lee, S., and Lee, J. (2018b). Interactive character animation by learning multi-objective control. *ACM Transactions on Graphics*, 37(6):180:1–180:10.
- Lee, S., Lee, S., Lee, Y., and Lee, J. (2021). Learning a family of motor skills from a single motion clip. *ACM Transactions on Graphics*, 40(4):93:1–93:13.
- Lee, S., Park, M., Lee, K., and Lee, J. (2019). Scalable muscle-actuated human simulation and control. *ACM Transactions on Graphics*, 38(4):73:1–73:13.
- Lee, S.-H. and Goswami, A. (2012). A momentum-based balance controller for humanoid robots on non-level and non-stationary ground. *Autonomous Robots*, 33(4):399–414.
- Lee, Y., Kim, S., and Lee, J. (2010a). Data-driven biped control. *ACM Transactions on Graphics*, 29(4):129:1–129:8.
- Lee, Y., Lee, K., Kwon, S.-S., Jeong, J., O’Sullivan, C., Park, M. S., and Lee, J. (2015). Push-recovery stability of biped locomotion. *ACM Transactions on Graphics*, 34(6):180:1–180:9.
- Lee, Y., Wampler, K., Bernstein, G., Popović, J., and Popović, Z. (2010b). Motion fields for interactive character locomotion. *ACM Transactions on Graphics*, 29(6):138:1–138:8.
- Legg, S. and Hutter, M. (2007). Universal Intelligence: A Definition of Machine Intelligence. *Minds and Machines*, 17(4):391–444.
- Leng, X., Piao, S., Chang, L., He, Z., and Zhu, Z. (2020). Universal Walking Control Framework of Biped Robot Based on Dynamic Model and Quadratic Programming. *Complexity*.
- Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867.

- Levine, S. and Popović, J. (2012). Physically plausible simulation for character animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.
- Levine, S., Wang, J. M., Haraux, A., Popović, Z., and Koltun, V. (2012). Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics*, 31(4):28:1–28:10.
- Li, J., Yin, Y., Chu, H., Zhou, Y., Wang, T., Fidler, S., and Li, H. (2020). Learning to Generate Diverse Dance Motions with Transformer. *CoRR*.
- Li, P., Aberman, K., Zhang, Z., Hanocka, R., and Sorkine-Hornung, O. (2022). GANimator: Neural Motion Synthesis from a Single Sequence. *ACM Transactions on Graphics*, 41(4):1–12.
- Li, Q., Takanishi, A., and Kato, I. (1993). Learning control for a biped walking robot with a trunk. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, volume 3, pages 1771–1777.
- Li, Z., Cheng, X., Peng, X. B., Abbeel, P., Levine, S., Berseth, G., and Sreenath, K. (2021). Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*.
- Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. (2018). RLlib: Abstractions for Distributed Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations*.
- Ling, H. Y., Zinno, F., Cheng, G., and van de Panne, M. (2020). Character Controllers Using Motion VAEs. *ACM Transactions on Graphics*, 39(4).
- Liu, L. and Hodgins, J. (2017). Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Transactions on Graphics*, 36(4):42a:1.
- Liu, L. and Hodgins, J. (2018). Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Transactions on Graphics*, 37(4):1–14.
- Liu, L., Panne, M. V. D., and Yin, K. (2016). Guided Learning of Control Graphs for Physics-Based Characters. *ACM Transactions on Graphics*, 35(3):29:1–29:14.
- Liu, L., Yin, K., van de Panne, M., Shao, T., and Xu, W. (2010). Sampling-based contact-rich motion control. *ACM Transactions on Graphics*, 29(4):128:1–128:10.
- Loshchilov, I. and Hutter, F. (2017). Decoupled Weight Decay Regularization.

- Macchietto, A., Zordan, V., and Shelton, C. R. (2009). Momentum control for balance. *ACM Transactions on Graphics*, 28(3):80:1–80:8.
- Maiolino, P., Maggiali, M., Cannata, G., Metta, G., and Natale, L. (2013). A Flexible and Robust Large Scale Capacitive Tactile System for Robots. *IEEE Sensors Journal*, 13(10):3910–3917.
- Marsden, J. E. and Ratiu, T. S. (2010). *Introduction to Mechanics and Symmetry: A Basic Exposition of Classical Mechanical Systems*. Springer.
- Mason, S., Rotella, N., Schaal, S., and Righetti, L. (2018). An MPC Walking Framework with External Contact Forces. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*.
- Mastalli, C., Merkt, W., Marti-Saumell, J., Ferrolho, H., Solà, J., Mansard, N., and Vijayakumar, S. (2022). A feasibility-driven approach to control-limited DDP. *Autonomous Robots*, 46(8):985–1005.
- McGreavy, C., Yuan, K., Gordon, D., Tan, K., Wolfslag, W. J., Vijayakumar, S., and Li, Z. (2020). Unified Push Recovery Fundamentals: Inspiration from Human Study. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*.
- Mesanan, G., Engelsberger, J., Garofalo, G., Ott, C., and Albu-Schäffer, A. (2019). Dynamic Walking on Compliant and Uneven Terrain using DCM and Passivity-based Whole-body Control. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*.
- Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: Yet Another Robot Platform. *International Journal of Advanced Robotic Systems*, 3(1):8.
- Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., and Montesano, L. (2010). The iCub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23(8):1125–1134.
- Miki, T., Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2022). Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822.
- Min, J. and Chai, J. (2012). Motion graphs++: a compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics*, 31(6):153:1–153:12.
- Min, J., Chen, Y.-L., and Chai, J. (2009). Interactive generation of human animation with deformable motion models. *ACM Transactions on Graphics*, 29(1):9:1–9:12.
- Mingo Hoffman, E., Traversaro, S., Rocchi, A., Ferrati, M., Settini, A., Romano, F., Natale, L., Bicchi, A., Nori, F., and Tsagarakis, N. G. (2014). Yarp Based Plugins for Gazebo Simulator. In Hodicky, J., editor, *Modelling and Simulation for Autonomous Systems*.

- Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.
- Mirjalili, R., Yousefi-Korna, A., Shirazi, F. A., Nikkhah, A., Nazemi, F., and Khadiv, M. (2018). A Whole-Body Model Predictive Control Scheme Including External Contact Forces and CoM Height Variations. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*.
- Miura, K., Morisawa, M., Nakaoka, S., Kanehiro, F., Harada, K., Kaneko, K., and Kajita, S. (2009). Robot motion remix based on motion capture data towards human-like locomotion of humanoid robots. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*.
- Modugno, V., Chervet, U., Oriolo, G., and Ivaldi, S. (2016a). Learning soft task priorities for safe control of humanoid robots with constrained stochastic optimization. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.
- Modugno, V., Neumann, G., Rueckert, E., Oriolo, G., Peters, J., and Ivaldi, S. (2016b). Learning soft task priorities for control of redundant robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*.
- Morin, P. and Samson, C. (2008). Motion Control of Wheeled Mobile Robots. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 799–826. Springer.
- Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., and Stoica, I. (2018). Ray: a distributed framework for emerging AI applications. In *Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation*.
- Mukai, T. (2011). Motion rings for interactive gait synthesis. In *Symposium on Interactive 3D Graphics and Games*.
- Mukai, T. and Kuriyama, S. (2005). Geostatistical motion interpolation. In *ACM SIGGRAPH 2005 Papers*.
- Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines.
- Natale, L., Bartolozzi, C., Pucci, D., Wykowska, A., and Metta, G. (2017). iCub: The not-yet-finished story of building a robot child. *Science Robotics*, 2(13):eaaq1026.
- Nava, G., Romano, F., Nori, F., and Pucci, D. (2016). Stability analysis and design of momentum-based controllers for humanoid robots. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Nori, F., Traversaro, S., Eljaik, J., Romano, F., Del Prete, A., and Pucci, D. (2015). iCub Whole-Body Control through Force Regulation on Rigid Non-Coplanar Contacts. *Frontiers in Robotics and AI*, 2.

- Nowlan, S. and Hinton, G. E. (1990). Evaluation of Adaptive Mixtures of Competing Experts. In *Advances in Neural Information Processing Systems*, volume 3.
- Ordonez-Appaez, D., Agudo, A., Moreno-Noguer, F., and Martin, M. (2022). An Adaptable Approach to Learn Realistic Legged Locomotion without Examples. In *2022 International Conference on Robotics and Automation (ICRA)*.
- Orin, D. E. and Goswami, A. (2008). Centroidal Momentum Matrix of a humanoid robot: Structure and properties. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Orin, D. E., Goswami, A., and Lee, S.-H. (2013). Centroidal dynamics of a humanoid robot. *Autonomous Robots*, 35(2-3):161–176.
- Ott, C., Lee, D., and Nakamura, Y. (2008). Motion capture based human motion recognition and imitation by direct marker control. In *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*.
- Ott, C., Roa, M. A., and Hirzinger, G. (2011). Posture and balance control for biped robots based on contact force optimization. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*.
- Padois, V., Ivaldi, S., Babič, J., Mistry, M., Peters, J., and Nori, F. (2017). Whole-body multi-contact motion in humans and humanoids: Advances of the CoDyCo European project. *Robotics and Autonomous Systems*, 90:97–117.
- Paine, N., Mehling, J. S., Holley, J., Radford, N. A., Johnson, G., Fok, C.-L., and Sentis, L. (2015). Actuator Control for the NASA-JSC Valkyrie Humanoid Robot: A Decoupled Dynamics Approach for Torque Control of Series Elastic Robots. *Journal of Field Robotics*, 32(3):378–396.
- Park, J. and Khatib, O. (2006). Contact consistent control framework for humanoid robots. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*.
- Parmiggiani, A., Maggiali, M., Natale, L., Nori, F., Schmitz, A., Tsagarakis, N., Victor, J. S., Becchi, F., Sandini, G., and Metta, G. (2012). The design of the iCub humanoid robot. *International Journal of Humanoid Robotics*, 09(04):1250027.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*.
- Penco, L., Clement, B., Modugno, V., Mingo Hoffman, E., Nava, G., Pucci, D., Tsagarakis, N. G., Mouret, J. B., and Ivaldi, S. (2018). Robust Real-Time Whole-Body Motion Retargeting from Human to Humanoid. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*.

- Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018). DeepMimic: example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4):1–14.
- Peng, X. B., Berseth, G., and van de Panne, M. (2015). Dynamic terrain traversal skills using reinforcement learning. *ACM Transactions on Graphics*, 34(4):80:1–80:11.
- Peng, X. B., Berseth, G., and van de Panne, M. (2016). Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics*, 35(4):81:1–81:12.
- Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. (2017). DeepLoco: dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics*, 36(4):1–13.
- Peng, X. B., Guo, Y., Halper, L., Levine, S., and Fidler, S. (2022). ASE: Large-Scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters. *ACM Transactions on Graphics*, 41(4):1–17. arXiv:2205.01906 [cs].
- Peng, X. B., Ma, Z., Abbeel, P., Levine, S., and Kanazawa, A. (2021). AMP: adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics*, 40(4):144:1–144:20.
- Peng, X. B. and van de Panne, M. (2017). Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter? In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*.
- Pollard, N., Hodgins, J., Riley, M., and Atkeson, C. (2002). Adapting human motion for the control of a humanoid robot. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*.
- Ponton, B., Herzog, A., Schaal, S., and Righetti, L. (2016). A convex model of humanoid momentum dynamics for multi-contact motion generation. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.
- Poulakakis, I., Smith, J., and Buehler, M. (2005). Modeling and Experiments of Untethered Quadrupedal Running with a Bounding Gait: The Scout II Robot. *The International Journal of Robotics Research*, 24:239–256.
- Pratt, J., Carff, J., Drakunov, S., and Goswami, A. (2006). Capture Point: A Step toward Humanoid Push Recovery. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*.
- Pratt, J., Koolen, T., de Boer, T., Rebula, J., Cotton, S., Carff, J., Johnson, M., and Neuhaus, P. (2012). Capturability-based analysis and control of legged locomotion, Part 2: Application to M2V2, a lower-body humanoid. *The International Journal of Robotics Research*, 31(10):1117–1133.
- Pucci, D., Romano, F., Traversaro, S., and Nori, F. (2016). Highly dynamic balancing via force control. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.

- Ramuzat, N., Buondonno, G., Boria, S., and Stasse, O. (2021). Comparison of Position and Torque Whole-Body Control Schemes on the Humanoid Robot TALOS. In *20th International Conference on Advanced Robotics (ICAR)*.
- Rapetti, L., Tirupachuri, Y., Darvish, K., Latella, C., and Pucci, D. (2020). Model-Based Real-Time Motion Tracking using Dynamical Inverse Kinematics on $SO(3)$. *Algorithms*.
- Righetti, L., Buchli, J., Mistry, M., and Schaal, S. (2011). Inverse dynamics control of floating-base robots with external constraints: A unified view. In *2011 IEEE International Conference on Robotics and Automation*.
- Rodriguez, D. and Behnke, S. (2021). DeepWalk: Omnidirectional Bipedal Gait by Deep Reinforcement Learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X.
- Roetenberg, D., Luinge, H., and Slycke, P. (2009). Xsens MVN: Full 6DOF Human Motion Tracking Using Miniature Inertial Sensors. *Xsens Motion Technol. BV, Tech. Rep.*
- Romano, F., Nava, G., Azad, M., Čamernik, J., Dafarra, S., Dermý, O., Latella, C., Lazzaroni, M., Lober, R., Lorenzini, M., Pucci, D., Sigaud, O., Traversaro, S., Babič, J., Ivaldi, S., Mistry, M., Padois, V., and Nori, F. (2018). The CoDyCo Project Achievements and Beyond: Toward Human Aware Whole-Body Controllers for Physical Human Robot Interaction. *IEEE Robotics and Automation Letters*, 3(1):516–523.
- Romualdi, G. (2022). *Online Control of Humanoid Robot Locomotion*. PhD thesis.
- Romualdi, G., Dafarra, S., Hu, Y., and Pucci, D. (2018). A Benchmarking of DCM Based Architectures for Position and Velocity Controlled Walking of Humanoid Robots. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. arXiv:1809.02167 [cs].
- Romualdi, G., Dafarra, S., Hu, Y., Ramadoss, P., Chavez, F. J. A., Traversaro, S., and Pucci, D. (2020). A Benchmarking of DCM Based Architectures for Position, Velocity and Torque Controlled Humanoid Robots. *International Journal of Humanoid Robotics*, 17(01):1950034.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.
- Rummery, G. and Niranjan, M. (1994). Online Q-Learning Using Connectionist Systems. *Technical Report CUED/F-INFENG/TR 166*.
- Saab, L., Ramos, O. E., Keith, F., Mansard, N., Souères, P., and Fourquet, J.-Y. (2013). Dynamic Whole-Body Motion Generation Under Rigid Contacts and Other Unilateral Constraints. *IEEE Transactions on Robotics*, 29(2):346–362.

- Scardapane, S., Scarpiniti, M., Comminiello, D., and Uncini, A. (2019). Learning Activation Functions from Data Using Cubic Spline Interpolation. In Esposito, A., Faundez-Zanuy, M., Morabito, F. C., and Pasero, E., editors, *Neural Advances in Processing Nonlinear Dynamic Signals*, Smart Innovation, Systems and Technologies. Springer International Publishing.
- Schaul, T., Antonoglou, I., and Silver, D. (2014). Unit Tests for Stochastic Optimization. In *International Conference on Learning Representations*.
- Schuller, R., Mesesan, G., Engelsberger, J., Lee, J., and Ott, C. (2021). Online Centroidal Angular Momentum Reference Generation and Motion Optimization for Humanoid Push Recovery. *IEEE Robotics and Automation Letters*, 6(3):5689–5696.
- Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*.
- Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *4th International Conference for Learning Representations*.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. In *Conference on Robot Learning*.
- Scianca, N., Cognetti, M., De Simone, D., Lanari, L., and Oriolo, G. (2016). Intrinsically stable MPC for humanoid gait generation. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.
- Scianca, N., De Simone, D., Lanari, L., and Oriolo, G. (2020). MPC for Humanoid Gait Generation: Stability and Feasibility. *IEEE Transactions on Robotics*, 36(4):1171–1188.
- Sciavicco, L. and Siciliano, B. (1998). A solution algorithm to the inverse kinematic problem for redundant manipulators. *IEEE Journal on Robotics and Automation*, 4(4):403–410.
- Shafiee, M., Romualdi, G., Dafarra, S., Chavez, F. J. A., and Pucci, D. (2019). Online DCM Trajectory Generation for Push Recovery of Torque-Controlled Humanoid Robots. In *IEEE-RAS International Conference on Humanoid Robots*.
- Shafiee-Ashtiani, M., Yousefi-Koma, A., Mirjalili, R., Maleki, H., and Karimi, M. (2017a). Push Recovery of a Position-Controlled Humanoid Robot Based on Capture Point Feedback Control. In *2017 5th RSI International Conference on Robotics and Mechatronics (ICRoM)*.
- Shafiee-Ashtiani, M., Yousefi-Koma, A., and Shariat-Panahi, M. (2017b). Robust bipedal locomotion control based on model predictive control and divergent component of motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*.

- Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2008). *Robotics: Modelling, Planning and Control*. Springer.
- Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299:103535.
- Singh, R. P., Benallegue, M., Morisawa, M., Cisneros, R., and Kanehiro, F. (2022). Learning Bipedal Walking On Planned Footsteps For Humanoid Robots. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*.
- Smaldone, F. M., Scianca, N., Lanari, L., and Oriolo, G. (2021). Feasibility-Driven Step Timing Adaptation for Robust MPC-Based Gait Generation in Humanoids. *IEEE Robotics and Automation Letters*, 6(2):1582–1589.
- Smaldone, F. M., Scianca, N., Lanari, L., and Oriolo, G. (2022). From Walking to Running: 3D Humanoid Gait Generation via MPC. *Frontiers in Robotics and AI*, 9.
- Smaldone, F. M., Scianca, N., Modugno, V., Lanari, L., and Oriolo, G. (2019). Gait Generation using Intrinsically Stable MPC in the Presence of Persistent Disturbances. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*.
- Smaldone, F. M., Scianca, N., Modugno, V., Lanari, L., and Oriolo, G. (2020). ZMP Constraint Restriction for Robust Gait Generation in Humanoids. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*.
- Spong, M. W. (1998). Underactuated mechanical systems. In Siciliano, B. and Valavanis, K. P., editors, *Control Problems in Robotics and Automation*, Lecture Notes in Control and Information Sciences, pages 135–150. Springer.
- Starke, S., Zhang, H., Komura, T., and Saito, J. (2019). Neural state machine for character-scene interactions. *ACM Transactions on Graphics*, 38(6):1–14.
- Starke, S., Zhao, Y., Komura, T., and Zaman, K. (2020). Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics*, 39(4).
- Starke, S., Zhao, Y., Zinno, F., and Komura, T. (2021). Neural animation layering for synthesizing martial arts movements. *ACM Transactions on Graphics*, 40(4):1–16.
- Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. (2020). OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672.
- Stephens, B. (2007). Humanoid push recovery. In *2007 7th IEEE-RAS International Conference on Humanoid Robots*.
- Stephens, B. J. and Atkeson, C. G. (2010). Push Recovery by stepping for humanoid robots with force controlled joints. In *2010 10th IEEE-RAS International Conference on Humanoid Robots*.

- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: an introduction. Second edition.* MIT press.
- Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems.*
- Takenaka, T., Matsumoto, T., and Yoshiike, T. (2009). Real time motion generation and control for biped robot 1st report: Walking gait pattern generation. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems.*
- Tautges, J., Zinke, A., Krüger, B., Baumann, J., Weber, A., Helten, T., Müller, M., Seidel, H.-P., and Eberhardt, B. (2011). Motion reconstruction using sparse accelerometer data. *ACM Transactions on Graphics*, 30(3):18:1–18:12.
- Taylor, G. W. and Hinton, G. E. (2009). Factored conditional restricted Boltzmann Machines for modeling motion style. In *Proceedings of the 26th Annual International Conference on Machine Learning.*
- Taylor, M., Bashkirov, S., Rico, J. F., Toriyama, I., Miyada, N., Yanagisawa, H., and Ishizuka, K. (2021). Learning Bipedal Robot Locomotion from Human Movement. In *2021 IEEE International Conference on Robotics and Automation (ICRA).*
- Tirupachuri, Y. (2020). *Enabling Human-Robot Collaboration via Holistic Human Perception and Partner-Aware Control.* PhD thesis.
- Tirupachuri, Y., Nava, G., Latella, C., Ferigo, D., Rapetti, L., Tagliapietra, L., Nori, F., and Pucci, D. (2020). Towards Partner-Aware Humanoid Robot Control Under Physical Interactions. In Bi, Y., Bhatia, R., and Kapoor, S., editors, *Intelligent Systems and Applications, Advances in Intelligent Systems and Computing*, pages 1073–1092.
- Traversaro, S. (2017). *Modelling, Estimation and Identification of Humanoid Robots Dynamics.* PhD thesis.
- Traversaro, S., Pucci, D., and Nori, F. (2017). A Unified View of the Equations of Motion used for Control Design of Humanoid Robots. *Online.*
- Traversaro, S. and Saccon, A. (2019). Multibody dynamics notation (version 2). *Technische Universiteit Eindhoven.*
- Truong, T.-V.-A., Flavigne, D., Pettrée, J., Mombaur, K., and Laumond, J.-P. (2010). Reactive synthesizing of human locomotion combining nonholonomic and holonomic behaviors. In *2010 3rd IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics*, pages 632–637.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, , and Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems.*
- Vukobratović, M. and Borovac, B. (2004). Zero-moment point — thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173.

- Wampler, K. and Popović, Z. (2009). Optimal gait and form for animal locomotion. In *ACM SIGGRAPH 2009 papers*.
- Wampler, K., Popović, Z., and Popović, J. (2014). Generalizing locomotion style to new animals with inverse optimal regression. *ACM Transactions on Graphics*, 33(4):49:1–49:11.
- Wang, J. M., Fleet, D. J., and Hertzmann, A. (2008). Gaussian Process Dynamical Models for Human Motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):283–298.
- Wang, Z., Albarghouthi, A., Prakriya, G., and Jha, S. (2022). Interval universal approximation for neural networks. *Proceedings of the ACM on Programming Languages*, 6(POPL):14:1–14:29.
- Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis.
- Wensing, P. M. and Orin, D. E. (2013). Generation of dynamic humanoid behaviors through task-space control with conic optimization. In *2013 IEEE International Conference on Robotics and Automation*.
- Wensing, P. M. and Orin, D. E. (2016). Improved Computation of the Humanoid Centroidal Dynamics and Application for Whole-Body Control. *International Journal of Humanoid Robotics*, 13(01):1550039.
- Wensing, P. M., Palmer, L. R., and Orin, D. E. (2015). Efficient recursive dynamics algorithms for operational-space control with application to legged locomotion. *Autonomous Robots*, 38(4):363–381.
- Wieber, P.-b. (2006). Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*.
- Wieber, P.-B., Tedrake, R., and Kuindersma, S. (2016). Modeling and Control of Legged Robots. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 1203–1234. Springer.
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256.
- Winkler, A. W., Bellicoso, C. D., Hutter, M., and Buchli, J. (2018). Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567.
- Won, J., Gopinath, D., and Hodgins, J. (2020). A scalable approach to control diverse behaviors for physically simulated characters. *ACM Transactions on Graphics*, 39(4).
- Xie, Z., Clary, P., Dao, J., Morais, P., Hurst, J., and van de Panne, M. (2019). Iterative Reinforcement Learning Based Design of Dynamic Locomotion Skills for Cassie. In *Conference on Robot Learning*.

- Xu, P. and Karamouzas, I. (2021). A GAN-Like Approach for Physics-Based Imitation Learning and Interactive Character Control. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 4(3):1–22. arXiv:2105.10066 [cs].
- Yang, C., Komura, T., and Li, Z. (2017). Emergence of human-comparable balancing behaviours by deep reinforcement learning. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*.
- Yang, C., Yuan, K., Heng, S., Komura, T., and Li, Z. (2020a). Learning Natural Locomotion Behaviors for Humanoid Robots Using Human Bias. *IEEE Robotics and Automation Letters*, 5(2):2610.
- Yang, C., Yuan, K., Merkt, W., Komura, T., Vijayakumar, S., and Li, Z. (2018). Learning Whole-Body Motor Skills for Humanoids. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*.
- Yang, C., Yuan, K., Zhu, Q., Yu, W., and Li, Z. (2020b). Multi-expert learning of adaptive legged locomotion. *Science Robotics*, 5(49):eabb2174.
- Yin, K., Loken, K., and van de Panne, M. (2007). SIMBICON: simple biped locomotion control. *ACM Transactions on Graphics*, 26(3):105–es.
- Yin, Z., Yang, Z., Van De Panne, M., and Yin, K. (2021). Discovering diverse athletic jumping strategies. *ACM Transactions on Graphics*, 40(4):1–17.
- Yoon, J., Son, B., and Lee, D. (2023). Comparative Study of Physics Engines for Robot Simulation with Mechanical Interaction. *Applied Sciences*, 13(2):680.
- Yuksel, S. E., Wilson, J. N., and Gader, P. D. (2012). Twenty Years of Mixture of Experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1177–1193.
- Zhang, H., Starke, S., Komura, T., and Saito, J. (2018). Mode-adaptive neural networks for quadruped motion control. *ACM Transactions on Graphics*, 37(4):1–11.

Appendices

Appendix A

Texture Task for the ANA Avatar XPRIZE Finals

This Appendix presents the implementation of the algorithm adopted by the iCub Team to address the *Texture Task* for the finals of the *ANA Avatar XPRIZE* competition¹, whose purpose was to create an avatar system capable of transporting human presence to a remote location in real time.

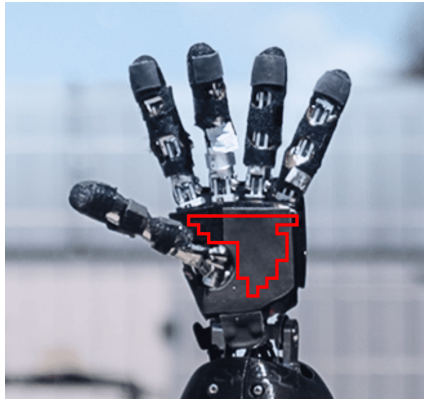
The ANA Avatar XPRIZE finals took place at the Long Beach Convention Center in Los Angeles, California, in the period 1-5 November 2022. They involved a total of 17 teams and focused on testing the avatar system (operated by an XPRIZE judge) in terms of complex locomotion and manipulation of heavy and textured objects in a scenario themed on the exploration of an alien planet. As iCub Team, we participated with the iCub3 Avatar System [Dafarra et al., 2022a], using the iCub v3 humanoid robot (see Section 3.4.2).

Among the tasks of the finals testing, the Texture Task is summarized as: the Avatar reaches through a curtain to identify a rough textured rock and retrieve it. Unfortunately, because of system failure on the first day of the competition, we did not get to test the Texture Task on stage during the finals. However, we illustrate in the following the method employed to attempt to address it.

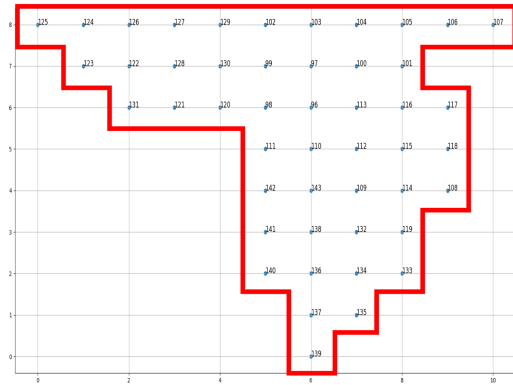
Texture Task

For the identification of the rough textured rock, we take advantage of the artificial skin covering the iCub v3 hand palms. In particular, each sensorized palm includes 48 tactile sensors, shown in Figure A.1. Since the rocks are supposed to be light and not fastened to the table during the tests, they can easily slip away if the operator tries to scan their surface using the avatar sensorized fingers. This is why we instruct the operator to approach the rocks from the top and make contact with them through the sensorized robot hand palm as shown in Figure A.2. When a contact is detected, a vibration pattern resembling either plain or rough texture is triggered on the operator's hand by means of a dedicated motor on the haptic glove worn by the operator. The rock texture is classified as follows.

¹<https://www.xprize.org/prizes/avatar>

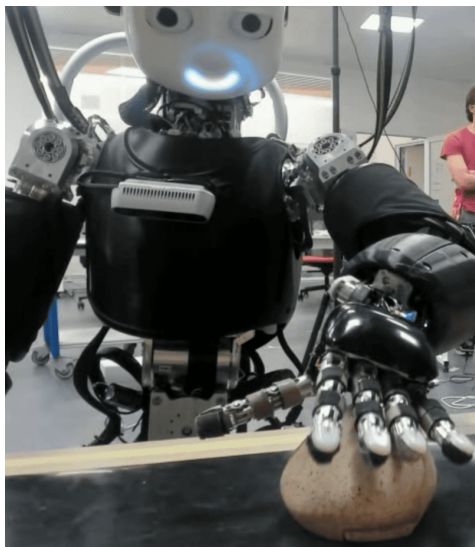


(a) The hand of iCub v3.

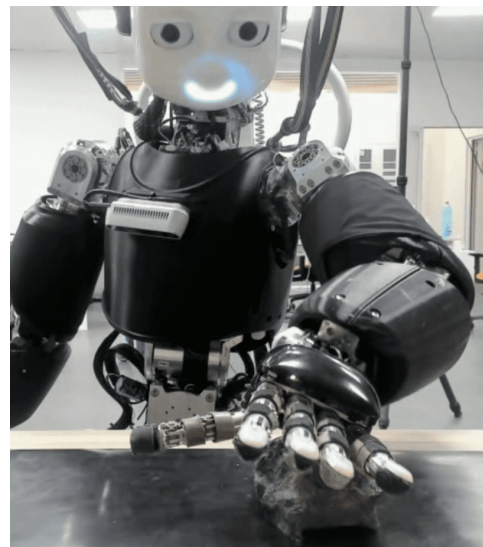


(b) The skin sensors on the iCub v3 hand.

Figure A.1. Detail of the sensorized hand of the iCub v3 humanoid, with the area covered by the artificial skin highlighted in red. The distribution of the 48 tactile sensors within their coverage area is reported (sensors indexes vary in [96,143]).



(a) Plain rock.



(b) Rough rock.

Figure A.2. Examples of training data collection for the rock texture classifier. The teleoperated robot reaches the rock from the top and tries to make contact through the palm of its hand.

For the classification of the rock texture which triggers the vibration pattern for the operator, we rely on a neural network trained for a binary classification task on the type of contact (i.e., rough or plain) from the tactile sensors' activations. In particular, the 48 tactile sensors of each robot hand provide measurements in $[0,255]$. The higher the value, the higher the measured pressure.

Given the location of the sensors on the iCub v3 hand shown in Figure A.1b, we decide to interpret their measurements as a 9x11-pixels grayscale image, where each pixel – excluding padding – corresponds to a tactile sensor. Figure A.3 shows sample images retrieved from the palm in contact with a plain and a rough rock, respectively. You can notice how in this case the active tactile sensors are more sparse during the contact with the rough rock.

Images of this kind represent the input for our binary classifier. In order to compose our training dataset with such images, we teleoperate the robot and make contact with each of the sample rocks while continuously changing their location on the table and approaching them from different directions with the robot hand. An example of data collection for both the plain and the rough rock is shown in Figure A.2. Our final dataset consists of around 150 contacts per class, where each contact includes the whole sequence of images collected throughout its duration. In this simple implementation, however, the time correlation between the subsequent instants of the same contact is not exploited, since the classifier, as detailed in the following, is not characterized by a recursive neural network architecture nor receives in input a temporal sequence.

As a classifier, we adopt a customized version of the Deep Convolutional Neural Network (DCNN) architecture for image classification known as AlexNet [Krizhevsky et al., 2017], composed in its original implementation by 5 2D-convolutional layers followed by 3 fully-connected layers. In particular, we customize it as follows:

- We use 3x3 convolutional filters from the very first 2D-convolutional layer (rather than the 11x11 and 5x5 filters applied in the first and second layer of the original implementation, respectively). This is to cope with our very low-dimensional input consisting of 9x11-pixels grayscale images.
- We remove the max pooling layers placed after the first three 2D-convolutional layers in the original implementation. We keep instead the max pooling after the fifth 2D-convolutional layer. Again, this is because of the our specific low-dimensional input.
- We scale the network in size (both in terms of number of filters for the 2D-convolutional layers and in terms of number of units of the fully-connected layers) by a factor of 32. This is to meet the real-time inference constraints.
- We return a single output from the last fully-connected layer, with a final sigmoid activation function, as required by our binary classification problem.

We train our classifier for 25 epochs on the training dataset, using batches of 32 samples and the Adam optimizer [Kingma and Ba, 2015]. On our test dataset which includes around 40 contacts in total, the overall trained model accuracy is 78%, although the misclassification rate is not completely balanced among the two classes. Please refer to Figure A.4 for a visualization of the achieved performances.

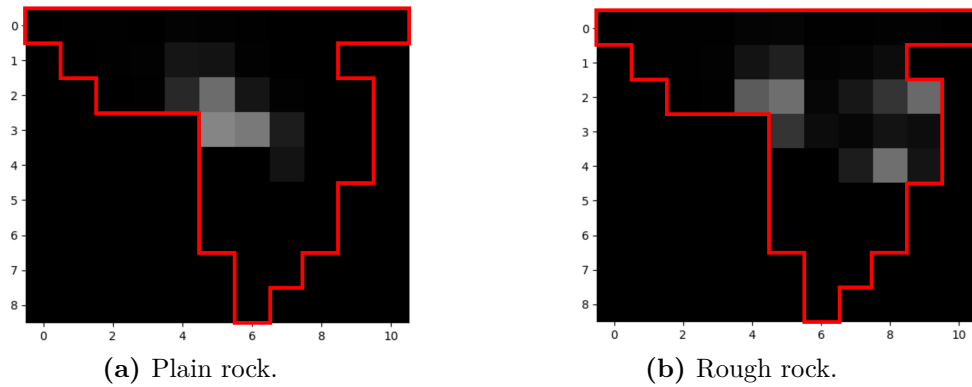


Figure A.3. Grayscale 9x11-pixels images extracted from the sensorized robot palm in contact with a plain and a rough rock. The higher the pressure measured by each tactile sensor, the whiter the correspondent pixel. The pixels outside the area covered by the skin sensors (highlighted in red) are simply padded in black.



Figure A.4. The rock classifier's performances on the 36 contacts of our test dataset. Each plot corresponds to a separate contact, either plain P_i or rough R_i , and shows a comparison of the ground truth and the predicted class for the entire duration of the contact. When the prediction and the ground truth do not coincide, the gap is filled in red to highlight the error. Although the misclassification rate increases for rough contacts, the overall accuracy on the test set, i.e., the average of the per-contact accuracies labeling each plot, reaches 78%.