MANCHESTER 1824
The University of Manchester

ISTITUTO ITALIANO DI TECNOLOGIA

A thesis submitted to the University of Manchester for the degree of
Doctor of Philosophy
in the Faculty of Science and Engineering

Simulation Architectures
for
Reinforcement Learning applied to Robotics

Diego Ferigo

2022

SCHOOL OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE

CONTENTS

ABSTRACT 17

DECLARATION OF ORIGINALITY 18

COPYRIGHT STATEMENT 19

ACKNOWLEDGMENTS 20

PROLOGUE 22

I BACKGROUND AND FUNDAMENTALS

1 SIMULATORS AND ENABLING TECHNOLOGIES 32

1.1 Simulators for robotics 32

1.2 Enabling technologies 37

2 ROBOT MODELLING 40

2.1 Notation 41

2.2 Points, Frames, Rotations, Transformations 43

2.3 Frame Velocity 44

2.4 Accelerations and Forces 49

2.5 Rigid-body Kinematics 52

2.6 Rigid-body Dynamics 53

2.7 Joint Model 58

2.8 Free-floating Mechanical Systems 62

3 BASICS OF REINFORCEMENT LEARNING 73

3.1 Key concepts 74

3.2 Reinforcement Learning Formalism 82

3.3 Algorithms 86

3.4 Policy Optimization 89

4 STATE-OF-THE-ART AND THESIS CONTENT 96

4.1 Review of Reinforcement Learning for Robot Locomotion 96

4.2 Review of Simulators for Robot Learning 99

4.3 Review of Push-recovery Strategies 101

4.4 Thesis Content 103

II CONTRIBUTION

5 REINFORCEMENT LEARNING ENVIRONMENTS FOR ROBOTICS 112
5.1 Frameworks providing robotic environments 113
5.2 Design Goals 119
5.3 ScenarIO: SCENe interfAces for Robot Input/Output 120
5.4 Gym-Ignition 125
5.5 Conclusions 127

6 LEARNING FROM SCRATCH EXPLOITING ROBOT MODELS 128
6.1 Training Environment 129
6.2 Agent 133
6.3 Reward Shaping 135
6.4 Results 140
6.5 Conclusions 144

7 CONTACT-AWARE MULTIBODY DYNAMICS 145
7.1 Notation 146
7.2 State-space Multibody Dynamics 149
7.3 Contact model 152
7.4 Integrators 160
7.5 Validation 162
7.6 Conclusions 170

8 SCALING RIGID-BODY SIMULATIONS 171
8.1 Floating-base rigid-body dynamics algorithms 172
8.2 Jaxsim 186
8.3 Validation 197
8.4 Conclusions 207

EPILOGUE 209

BIBLIOGRAPHY 218

III APPENDIX

A CENTER OF PRESSURE 236

Word Count: 57000

LIST OF FIGURES

Figure 2.1 Illustration of the joint model. The frame of the parent link P is shown as $\lambda(i)$, and the frame of the child link C is shown as i. The i-th joint connects together the two links, enforcing a motion constraint. The operators $\mathrm{pre}(\cdot)$ and $\mathrm{suc}(\cdot)$ accept a joint number and return, respectively, its predecessor and successor frames. The successor frame of a joint matches with the frame i of the child link. The joint model aims to obtain the transform ${}^{\lambda(i)}\mathbf{H}_i(s)$ as a function of the joint position s. 57

Figure 2.2 Illustration of 6D force propagation through a set of links connected by joints. The link L receives a force $\mathbf{f}_{J_P}$ from its parent P transmitted through the joint J_P, and an external force $\mathbf{f}_E$. The link L transmits to each children C_i a force $\mathbf{f}_{J_{C_i}}$ through their connecting joint J_{C_i}. 62

Figure 3.1 The Reinforcement Learning setting. 74

Figure 3.2 Illustration of the process generating trajectory data. 79

Figure 5.1 Architecture of Scenario and Gym-Ignition. Users of the overall framework just need to provide the URDF or SDF description of their robot and implement the `Task` interface with the desired decision-making logic. The framework, following a top-down approach, exposes to the `Agent` algorithms the unified `gym.Env` interface. The provided `Runtime` classes either instantiate the simulator, or handle soft real-time logic for real-world robots. The runtimes are generic and can operate on any decision-making logic that exposes the `Task` interface. Finally, `Task` implementations use the Scenario APIs to interact with the robots part of the environment. A typical data flow starts with the agent setting the action with `gym.Env.step`. The processing of the action is a combination of logic inside the active runtime and the active task. In particular, the runtime receives the action and directly forwards it to the task for being processed. The task, by operating transparently over the Scenario APIs, applies the action to the robot, and then waits the runtime to perform the time stepping. After this phase, the task computes the reward, packs the observation, detects if the environment reached the terminal state, and returns all this data back to the agent passing through the `gym.Env` APIs. 124

Figure 6.1 The proposed control system. 129

Figure 6.2 Learning curves over 11 training runs. 140

Figure 6.3 (a) Push-recovery success rates on the horizontal plane (forward push: 0 rad, $\mu_c = 1$). (b) Results with $\mu_c = 0.2$. 141

Figure 6.4 (a) The initial joint configuration s_0. (b) Sequences showing ankle, step, and momentum push-recovery strategies. The robot is pushed by a sphere shot from the left side of the image. Impact takes place in the second frame. 141

Figure 6.5 Consecutive counterbalanced forces in random directions over 50 trials for each combination of magnitude and duration. Forces are applied to the base, chest, and elbow links for an increasing duration. 143

Figure 7.1 Illustration of the point-surface soft-contact model for non-planar terrains. The collidable point follow a trajectory $\mathbf{p}_{cp}(t)$, penetrating the ground in $\mathbf{p}^0_{cp} := \big(x^0, y^0, \mathcal{H}(x^0, y^0)\big)$. While penetrating the material, the point reaches a generic point $\mathbf{p}_{cp}$, over which a local contact frame $C = (\mathbf{p}_{cp}, [W])$ is positioned, with a linear velocity ${}^{C[W]}\boldsymbol{v}_{W,C} = {}^{W}\dot{\mathbf{p}}_C \in \mathbb{R}^3$. The figure reports also the *penetration depth* $h \in \mathbb{R}$, the *normal deformation* $\delta \in \mathbb{R}$, and the compounded tangential deformation $\mathbf{m} \in \mathbb{R}^3$ of the material, used for the calculation of the 3D reaction force ${}_{C}\boldsymbol{f}_{cp}$ with the proposed soft-contact model. 153

Figure 7.2 Sphere trajectory of the bouncing experiment. 163

Figure 7.3 Evolution over time of the bouncing ball experiment's data. (7.3a) reports the mechanical energy of the system and the norm of the linear component of the contact forces summed and expressed in the B frame of the spherical link. (7.3b) and (7.3b) report a closer view of the first two impacts. (7.3d) reports the plot of the base height, where both the bouncing and rolling phases can be observed. 164

Figure 7.4 Evolution over time of the sliding box experiment's data. From top to bottom, the first plot shows the x component of the CoM position, the second plot shows the x component of the CoM velocity, and the third plot shows the profile of the applied external force to the CoM frame G. 166

Figure 7.5 Comparison of the box's CoM trajectory simulated with the proposed soft-contact model and the Mujoco simulator, considering a coefficient of friction (a) $\mu = 2$, (b) $\mu = 0$, (c) $\mu = 0.5$. 167

Figure 8.1 Sequence showing four time instants of the astronaut simulation used for both the assessments of the momentum conservation and the energy conservation. In the former experiment, the joints are actuated with random torques, while in the latter, the joints are not actuated and evolve in open-loop accordingly to their initial velocity. 190

Figure 8.2 Momentum drift after 1 simulated second of the iCub humanoid robot in a world without gravity, starting from a configuration with zero velocity and applying random joint forces. The plot shows the norm of the linear and angular component of the momentum computed in inertial-fixed coordinates. Gazebo Sim failed to simulate the configuration with the 100 ms step. 191

Figure 8.3 Mechanical energy drift over 100 simulated seconds of the iCub humanoid robot in a world without gravity, starting from a configuration with a given generalized velocity. 192

Figure 8.4 Benchmark of the RBDAs implemented in JAXsim against those implemented in Pinocchio. (a) shows the results of the 9-DoFs fixed-base Panda manipulator from Franka Emika, (b) the results of the 12-DoFs quadruped ANYmal C from ANYbotics, and (c) the results of the 32-DoFs humanoid iCub from IIT. The execution of JAXsim's algorithms run on average 10 times slower than Pinocchio when executed on CPU, and 100 times when executed on GPU. 194

Figure 8.5 Comparison of parallelization performance of a simulation step executed on CPU and GPUs. The simulated models are 23 DoFs replicas of the iCub humanoid robot, and the simulation step length is $1\ ms$ with a forward Euler scheme. The time taken by the CPU scales mostly linearly with the number of simulated models, while the GPUs are able to exploit the parallel capability almost up to their CUDA cores (640 on the laptop, 4608 on the workstation). For each sample, we show the equivalent RTF. The CPU cannot scale well over the number of models when integrating more than 16 replicas. Instead, the GPUs show an interval that depends on their parallellization capabilities in which the execution time is not affected significantly by the number of integrated models. Also on GPUs, however, the performance start degrading when the number of integrated models exceeds the available CUDA cores. 196

Figure 8.6 Illustration of the cartpole model in the $\theta = d = 0$ configuration. 198

Figure 8.7 Learning curves of the cartpole swing-up task. The plot reports the mean and standard deviation of the average rewards $\hat{r}_t^{(k)}$ computed over $k = 10$ different training executions. For each individual training, the average reward $\hat{r}_t$ in the considered parallel setting is computed by averaging at each time step the 512 rewards received from the vectorized `gym.Env.step`. 202

Figure 8.8 Sim-to-sim comparison of the trajectories obtained by exploiting the swing-up policy learned in a JAXsim environment. The JAXsim curves correspond to an in-distribution setting, where the policy is evaluated in the same simulator that generated training data. Instead, the Mujoco curves correspond to an out-of-distribution setting, where the policy is evaluate in a simulator different from the one that generated training data. Note that θ, due to its range, is projected in the $[-\pi, \pi]$ range. 205

Figure 8.9 Trajectories of the cartpole swing-up policy acting on the out-of-distribution environment simulated in Mujoco. The *nominal* curves are obtained by running the policy on a cartpole model having nominal masses of both the cart and the pole, and with no joint damping. The *mass* curves show the obtained trajectories with the model having the masses of both bodies multiplied by $2x$. The *mass+damping* curves are generated in a setting that extends the *mass* one by also considering for both joints a damping with $k_v = 0.015$. Note that in this case, we removed the bounds of the pole angle, showing more clearly the number of swings used by the policy to reach the balancing state. 206

Figure A.1 Illustration of the setting in which the CoP is calculated. 236

LIST OF TABLES

Table 2.1 List of motion subspaces for the supported 1 DoF joints. 60

Table 5.1 Comparison of frameworks that provide robotic environments compatible with OpenAI Gym. 118

Table 6.1 Observation components. 130

Table 6.2 PPO, policy, and training parameters. 133

Table 6.3 Reward function details. Terms with a defined cutoff are processed by the RBF kernel. 139

Table 7.1 Mujoco configuration considered in the experiments of the sliding box on inclined surface matching as close as possible the setting and properties of our soft-contact model. Refer to the official documentation at `https://mujoco.readthedocs.io` for a detailed explanation of the options. 168

Table 8.1 Comparison of modern physics engines similar to JAXsim. [*] JAXsim is developed with a differentiable framework, but this functionality has to be finalised. 188

Table 8.2 Specifications of the settings in which the benchmarks are executed. 189

Table 8.3 Specifications of the machine used to execute the validation experiments. 198

Table 8.4 Properties of the environment implementing the cartpole swing-up task. 199

Table 8.5 PPO parameters for the the cartpole swing-up environment. 199

Table 8.6 Mujoco properties used for the sim-to-sim evaluation of the trained cartpole swing-up policy. Refer to the official documentation at `https://mujoco.readthedocs.io` for a detailed explanation of the options. 203

ACRONYMS

ABA	Articulated Body Algorithm
AD	Automatic Differentiation
AI	Artificial Intelligence
API	Application Programming Interface
CH	Convex Hull
CoM	Center of Mass
CoP	Center of Pressure
CP	Capture Point
CPU	Central Processing Unit
CRBA	Composite Rigid Body Algorithm
DCM	Divergent Component of Motion
DL	Deep Learning
DoF	Degree of Freedom
DRL	Deep Reinforcement Learning
DS	Double Support
EoM	Equation of Motion
F/T	Force/Torque
GAE	Generalized Advantage Estimator
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
IMU	Inertial Measurement Unit
JIT	Just-in-time
KL	Kullback–Leibler

LIP	Linear Inverted Pendulum
MDP	Markov Decision Process
MJCF	Mujoco XML Format
ML	Machine Learning
NN	Neural Network
ODE	Ordinary Differential Equation
PDF	Probability Density Function
PG	Policy Gradient
PPO	Proximal Policy Optimization
RK4	Runge-Kutta 4
RNG	Random Number Generator
RBDA	Rigid Body Dynamics Algorithm
RBF	Radial Basis Function
RTF	Real-Time Factor
RL	Reinforcement Learning
RNEA	Recursive Newton-Euler Algorithm
SAC	Soft Actor-Critic
SDF	Simulation Description Format
SP	Support Polygon
TPU	Tensor Processing Unit
TRPO	Trust Region Policy Optimization
URDF	Unified Robot Description Format
USD	Universal Scene Description
XLA	Accelerated Linear Algebra
ZMP	Zero Moment Point

LIST OF SYMBOLS

Robot Kinematics and Dynamics

$\mathbf{p}$ Point in space

$A, B, C, \ldots$ Frame names

W World (inertial) frame

$[A]$ Orientation frame of frame A

$\mathbf{o}_A \in \mathbb{R}^3$ Origin of frame A

$A = (\mathbf{o}_A, [A])$ Definition of frame A

${}^A\mathbf{p} \in \mathbb{R}^3$ Coordinate vector of point $\mathbf{p}$ expressed in frame A

${}^A\mathbf{R}_B \in \mathrm{SO}(3)$ Rotation matrix from orientation frame $[B]$ to $[A]$

${}^A\mathbf{H}_B \in \mathrm{SE}(3)$ Homogeneous transformation from frame B to A

${}^A\tilde{\mathbf{p}} \in \mathbb{R}^4$ Homogeneous representation of coordinate vector ${}^A\mathbf{p}$

${}^C\boldsymbol{v}_{A,B} \in \mathbb{R}^3$ Linear velocity of frame B relative to frame A, expressed in C

${}^C\boldsymbol{\omega}_{A,B} \in \mathbb{R}^3$ Angular velocity of frame B relative to frame A, expressed in C

${}^C\mathbf{v}_{A,B} \in \mathbb{R}^6$ 6D velocity of frame B relative to frame A, expressed in C

${}^C\mathbf{v}^\wedge_{A,B} \in \mathfrak{se}(3)$ Matrix representation of 6D velocity ${}^C\mathbf{v}_{A,B}$

${}^A\mathbf{X}_B \in \mathbb{R}^{6\times6}$ Velocity transformation from frame B to frame A

${}^C\mathbf{v}_{A,B}\times \in \mathbb{R}^{6\times6}$ Cross product operator on $\mathbb{R}^6$ for 6D velocities

${}^C\dot{\mathbf{v}}_{A,B} \in \mathbb{R}^3$ Apparent acceleration of frame B relative to A, expressed in C

${}^C\mathbf{a}_{A,B} \in \mathbb{R}^3$ Intrinsic acceleration of frame B relative to A, expressed in C

${}^C\bar{\mathbf{a}}_{A,B} \in \mathbb{R}^3$ Proper acceleration of frame B relative to A, expressed in C

${}_A\boldsymbol{f} \in \mathbb{R}^3$ Linear force expressed in frame A

${}_A\boldsymbol{m} \in \mathbb{R}^3$ Angular force (torque) expressed in frame A

${}_A\mathbf{f} \in \mathbb{R}^6$ 6D force expressed in frame A

${}_A\mathbf{X}^B \in \mathbb{R}^{6\times6}$ Force transformation from frame B to frame A

${}^C\mathbf{v}_{A,B}\bar{\times}^* \in \mathbb{R}^{6\times6}$ Cross product operator on $\mathbb{R}^6$ for 6D forces

$I \in \mathbb{R}^{3\times3}$ Inertia tensor

${}_B\mathbb{M} \in \mathbb{R}^{6\times6}$ 6D inertia matrix, expressed in frame B

$g \in \mathbb{R}^+$ Standard gravity

${}^W\mathbf{g} \in \mathbb{R}^3$ Gravitational acceleration vector

${}^X\mathbf{S}_{P,C} \in \mathbb{R}^6$ Joint motion subspace between frame P and C, expressed in X

$n \in \mathbb{N}$ Number of degrees of freedom of a multibody system

$\mathbf{q} \in \mathrm{SE}(3) \times \mathbb{R}^n$ Floating-base position

${}^X\boldsymbol{\nu} \in \mathbb{R}^{6+n}$ Floating-base velocity having base velocity ${}^X\mathbf{v}_{W,B}$

${}^Y J_{B,E} \in \mathbb{R}^{6\times n}$ Relative Jacobian of frame E w.r.t. B, expressed in Y

$\mathbf{s} \in \mathbb{R}^n$ Joint positions

$\dot{\mathbf{s}} \in \mathbb{R}^n$ Joint velocities

$\ddot{\mathbf{s}} \in \mathbb{R}^n$ Joint accelerations

${}^Y J_{W,E} \in \mathbb{R}^{6\times(6+n)}$ Free-floating Jacobian of frame E, expressed in Y

${}^Y J_{W,E/X} \in \mathbb{R}^{6\times(6+n)}$ Free-floating Jacobian of frame E, expressed in Y, for base velocity expressed in X

$M(\mathbf{q}) \in \mathbb{R}^{(6+n)\times(6+n)}$ Mass matrix

$C(\mathbf{q}, \boldsymbol{\nu}) \in \mathbb{R}^{(6+n)\times(6+n)}$ Coriolis matrix

$g(\mathbf{q}) \in \mathbb{R}^{6+n}$ Potential force vector (or gravity vector)

$h(\mathbf{q}, \boldsymbol{\nu}) \in \mathbb{R}^{6+n}$ Bias forces vector

$\boldsymbol{\tau} \in \mathbb{R}^n$ Joint generalized forces

$\mathcal{L}$ Set of link indices

n_L Number of links

$\mathbf{f}_{\mathcal{L}}^{\text{ext}} \in \mathbb{R}^{6 \times n_L}$ Vector stacking external forces applied to all links

$J_{\mathcal{L}} \in \mathbb{R}^{6n_L \times (6+n)}$ Matrix stacking floating-base Jacobians of all links

$q \in \mathbb{H}$ A quaternion

$\bar{q} \in \mathrm{Spin}(3)$ A unit quaternion

$\mathtt{Q} = (w, \mathbf{r}) \in \mathbb{R}^4$ Quaternion coefficients

$\mathcal{H}$ Function providing terrain height

$\mathcal{S}$ Function providing terrain normal

$\mathbf{m}$ 3D tangential deformation of the terrain's material

$(\cdot)^{\parallel}$, $(\cdot)_{\parallel}$ Component parallel to the terrain

$(\cdot)^{\perp}$, $(\cdot)_{\perp}$ Component normal to the terrain

Reinforcement Learning

$\mathcal{S}$ The state space

$\mathcal{A}$ The action space

$s_t \in \mathcal{S}$ State of the environment at time t

$a_t \in \mathcal{A}$ Action applied to the environment at time t

$\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Pr[\mathcal{S}]$ State-transition probability density function

$\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ Reward function

$r_t \in \mathbb{R}$ Immediate reward at time t

$a = \mu(s)$ Action taken from deterministic policy in state s

$a \sim \pi(\cdot \mid s)$ Action sampled from stochastic policy in state s

$\pi_{\boldsymbol{\theta}}$ Stochastic policy parameterized with $\boldsymbol{\theta}$

$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ Trajectory of states and actions

$s_0 \sim \rho_0(\cdot)$ Sampling state from initial state distribution

$\hat{R}_t$ Reward-to-go at time t

R_t Return at time t

$R(\tau)$ Discounted return of trajectory τ

$J(\pi)$ Performance function of stochastic policy π

π^* Optimal policy

$\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \rho_0 \rangle$ Tuple defining a Markov Decision Process

$Q^\pi(s, a)$ Action-value function for policy π at state-action pair (s, a)

$V^\pi(s)$ State-value function for policy π at state s

$A^\pi(s, a)$ Advantage function for policy π at state-action pair (s, a)

$\mathbb{E}[\cdot]$ Expected value of a random variable

$\hat{\mathbb{E}}[\cdot]$ Empirical average estimating the expected value of a random variable from samples

THE UNIVERSITY OF MANCHESTER

SIMULATION ARCHITECTURES FOR REINFORCEMENT LEARNING APPLIED TO ROBOTICS

DIEGO FERIGO
DOCTOR OF PHILOSOPHY
JULY 2022

There is no doubt that we are living in the age of data. In the last two decades, the scientific community has been able to produce systems with superhuman capabilities through the combination of modern hardware advancements, novel learning algorithms and architectures, and advances in software frameworks. Such progress revolutionised domains like computer vision and language processing, showing performance previously out of reach. One may think that results could transfer straightforwardly to other fields like robotics until realising the existence of domain-specific characteristics and limitations hindering the potential of these learning methods. Generating enough data from real-world robots is often too expensive or not even possible to the desired scale. Data sampled from robots has a sequential nature, and not all families of learning algorithms are effective in this context. Furthermore, most algorithms that excel in this sequential setting, such as those belonging to the Reinforcement Learning (RL) family, learn by a trial-and-error process, which could lead to trajectories that damage either the robots or their surroundings.

In this thesis, we attempt to answer the question, *"How can modern technology help us generate synthetic data for humanoid robot planning and control?"*. Motivated by the advancements in hardware accelerators that are revolutionising scientific computing, we limit our analysis to the simulation realm. In this context, we first introduce a software architecture allowing to structure learning environments for robotics that can be adopted to train and run RL policies regardless of the simulated or real-world setting. With its underlying simulation technology and exploiting a scheme based on reward shaping, we validate the architecture by training with RL a push-recovery controller capable of synthesising whole-body references for the humanoid robot iCub. Then, motivated by overcoming the bottlenecks related to the poor sampling performance of traditional rigid-body simulators, we present a new physics engine in reduced coordinates that can simulate robots interacting with a ground surface on hardware accelerators like Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs). To this end, we present a contact-aware continuous state-space representation describing the dynamical evolution of floating-base robots that can be numerically integrated for simulation purposes. We adopt the new general-purpose Gazebo Sim simulator as our first solution to sample synthetic data, and exploit JAX and its hardware support to scale the sampling performance for highly parallel problems. Furthermore, we implement and benchmark common Rigid Body Dynamics Algorithms (RBDAs) part of the proposed physics engine on hardware accelerators and assess their scalability properties on different GPUs. These pieces of technology help to lower the computational barriers that nowadays are still among the main bottlenecks for obtaining intelligent agents, democratising the applicability of this family of learning-based methods.

DECLARATION OF ORIGINALITY

I hereby confirm that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

COPYRIGHT STATEMENT

i The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made *only* in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy[1], in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations[2] and in The University's policy on Presentation of Theses.

1 http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420
2 http://www.library.manchester.ac.uk/about/regulations/

The meeting of two personalities
is like the contact of two chemical substances:
if there is any reaction, both are transformed.
— Carl Gustav Jung

ACKNOWLEDGMENTS

Ups and downs. Research, much like life, is a process. We often expect a smooth and straightforward path, only to find it filled with unexpected hurdles, surprising twists, and – occasionally – those truly remarkable moments at the summit.

As I embarked on this journey, I was already immersed in the vibrant world of research at the Dynamic Interaction Control laboratory, guided by Francesco Nori, within the prestigious walls of IIT. To him, I extend my heartfelt gratitude for giving me the opportunity to be part of this beautiful, unique, and enriching community, which has been my home for the past seven years. The memories forged in Genova – professional and personal – will forever shape the narrative of my life.

My deepest appreciation goes to my supervisor, Daniele Pucci, head of the Artificial Mechanical Intelligence laboratory. Daniele, you have been not only a supervisor but also a colleague, a friend, and, as time passed, a mentor. I am profoundly thankful for the freedom you granted me to explore uncharted territories without the weight of mere numerical publications. Your support, even for the craziest ideas – some good, some not so much :) – and your guidance in navigating the most intricate of situations have been invaluable. I must also express my gratitude to Professor Angelo Cangelosi for his supervision from the Manchester side and his constant availability to clarify doubts and provide insights. For the necessary funding that sustained my research project, I owe an immense debt of gratitude to Giorgio Metta. His support was pivotal in bringing my research to fruition. I cannot thank Professor Bruno Adorno and Professor Michael Mistry enough for their roles as examiners during my viva. Your feedback has been instrumental in achieving the final quality of this thesis. I'm grateful for your friendly approach in the rigorous context of the defense, you managed to transform it into a meaningful and enjoyable discussion among peers. I couldn't have asked for a more supportive and constructive examination.

Listing all the individuals who have played pivotal roles in my journey is an impossible task. In AMI, we've grown too vast for individual mentions, but it's essential to convey how profoundly meaningful your impact has been. Extending this sentiment to the broader IIT community, I've had the privilege of learning from brilliant scientists and talented engineers. I've absorbed so much knowledge from our interactions, and I hope to have given back even a fraction of what I received. Navigating this world of research together has been challenging, but it's a journey that's enriched us all. I'm confident that each of you is destined for great success, and our future accomplishments will be a direct result of this collaborative journey. Thank you, friends.

Zooming out for a broader perspective, the contributions highlighted in this thesis owe much to the continuous efforts of the entire robotics and open-source communities. These communities provide essential tools and shared knowledge. Research is done by tiny bits from individuals, but these bits would have no meaning if not built into a broader, well-established context. I hope that with this work, I succeeded in contributing my grain to this vast sandy beach of knowledge. GitHub, in particular, has become a cozy home where to collaborate, strengthen our efforts, make connections, and foster growth. I wish other domains could embrace such a positive spirit.

I hold a strong belief that I wouldn't have been able to weather with the countless lows that characterize most doctoral journeys and maintain my mental well-being without the relief that sports and the great outdoors provided. When I felt out of place, amidst the rigors of day-to-day life, spending time in nature became essential to ground me down again. Out there, I feel more alive than words can convey; it's as if I'm deeply rooted to my origins, and these roots don't thrive within the confines of concrete. I'm sincerely thankful for the tranquility of the mountains, which reflects the inner peace I aim to experience daily and represents the place where I feel most at home. Those who can't understand this connection may find it challenging to comprehend who I am. In the winter, I find comfort in the Alps, rejuvenating through the simple act of skiing with its corresponding rush of adrenaline. As warmer seasons approach, I embark on countless hikes, exploring the stunning landscapes of Liguria or my beloved Friuli mountains back home. Thanks to Italy, with its incredible diversity. Food, nature, culture, people, weather - life outside the office is truly where it's at. There's no better place in the world where all of these elements come together so harmoniously.

Throughout the year, CrossFit has always been a constant presence, providing a refuge for both mental and physical renewal, and offering short-term goals that kept me on track. Amidst the challenges, whether the demands of academia or the disruptions of a global pandemic, the comfort drawn from these activities has been my saving grace. To Canteri CrossFit, you hold a special place in my heart. I found teammates, passionate coaches, good friends, and a new family within your four walls. There were times when I walked into the box feeling well over my head, I admit, but all doubts and worries dissipated as I walked out, stronger in every way. You've been my lifeline, safeguarding my mental health during these years, and I cannot express enough gratitude for your constant support.

And last but certainly not least, I want to conclude by extending a heartfelt thank you to my family: my mom Silvana, my sister Erica, and my brother Giulio. Your consistent support and unwavering acceptance of every choice I've made throughout my life, without judgment, has been a source of immeasurable strength. Dad, even though you couldn't witness this journey firsthand, I always felt my back covered by your usual silent approval. I love you all.

In theory, there is no difference between practice and theory.
In practice, there is.
— Benjamin Brewster

PROLOGUE

Technology and automation became, over time, seamlessly integrated commodities in the daily life of any person living in modern society. We became so used to their presence to the extent that we often forget their impact on our daily routine. We cruise the world either entering or driving highly automated machines endowed with motors and sensors. We load dishwashers and washing machines with dirty dishes and clothes, finding them clean when their cycle ends. We rely so much on the functionality of devices we constantly keep in our pockets that we would struggle to reach any new destination in their absence. We carry technology inside our bodies to prevent the fatal collapse of organs necessary to sustain life.

Robots represent one of the categories of objects that better describe the combination of technology and automation. The Oxford Dictionary provides the following definition:

robot *a machine that can perform a complicated series of tasks by itself.*

We are surrounded by robots, even if we do not realise it. In fact, as soon as their intelligence becomes embodied with their functionality, we stop to call them robots.

Our modern language considers robots those systems able to manipulate or navigate their surrounding environment with a degree of autonomy. Though, contrary to society's expectations built from science fiction, modern robotic systems can reliably operate only within controlled environments and perform narrow tasks. While we have already succeeded in deploying robots in environments with these characteristics like industry, bringing them outside production lines requires a different level of autonomy, dexterity, agility, and decision-making capabilities. Actions like interacting with a

constantly-changing environment, predicting people's intents, or generalising prior knowledge are still far out of today's robots' reach.

Machine intelligence is among the most fascinating problems currently researched by our society. Deep Learning (DL), which consists of the recent combination of Machine Learning (ML) with deep Neural Networks (NNs) recently enabled by the computational power of modern computing, revolutionised domains like computer vision and natural language processing. Under the assumption of large enough datasets, the most advanced algorithms belonging to the Supervised Learning family have been demonstrated to be capable of training systems that have often shown super-human performance. One could think of applying similar techniques to the field of robotics to obtain comparable success without realising the inherent challenges posed by this domain. Robots are physical systems. Generating datasets as large as those that characterise the most recent research in computer vision and language processing would either take too long or not even be possible, without even considering the wear and tear of the hardware and the operational and maintenance cost. Furthermore, regardless of the feasibility of data collection, robotics is an interactive domain, and data sampled from robots has a sequential conformation. The application of Supervised Learning that excels on static and offline datasets has been either ineffective or not an option.

Supervised Learning is just one of the subdomains belonging to ML. The 1990s experienced an increased research interest in a new family of methods specialised in sequential decision-making problems, that nowadays belong to the RL subdomain. The technological progress triggered by the advent of DL had a strong impact also to unlock the RL potential, which was initially constrained by computational limits. The transfer to robotics was inevitable. However, RL methods learn sequentially following a process similar to trial-and-error. When applied to robotics, during their training phase, these methods could generate control actions that may damage either the robot or its surroundings. For this reason, the accomplishments in the past decade of RL applied to robotics have been demonstrated mainly in simulation rather than with real-world robots. Generating synthetic data from simulators is cheap and effective, can be scaled to dozens or hundreds of concurrent instances, and there is no risk of damage.

Undeniably, synthetic data can only capture physical effects that can be described through equations executed by a computer program, and similarly to other domains, also robotic simulations suffer from the trade-off between accuracy and speed. Particularly, contact dynamics has always been challenging to capture with high fidelity, and usually simulators neglect non-ideal effects like motor dynamics, actuation delays, and backlash. However, the past decades have shown that advances in computing hardware and software methodologies have always had a substantial impact on reducing the effects of this trade-off –also called *reality gap*– and we believe that also research in robotic simulations will continue in this direction.

In the 2010s, the RL community has been particularly prolific, producing a considerable number of new algorithms. Despite their improved properties like increased sample efficiency, faster and more stable convergence, etc. they always rely on a given amount of trial-and-error experience, with no exception. If, on the one hand, better algorithms are lower-bounded by the least amount of data describing the decision-making logic to learn, on the other hand, the amount of synthetic data that simulators can generate has no theoretical upper bound. The rapid adoption of RL in the robotics domain forced practitioners to use the simulation technology from either robotics or gaming available at that time, which was never optimised for maximising execution speed. In fact, a single simulation instance executing in real-time has always been more than enough for the original purpose. It is not uncommon to find RL experiments that require days, weeks, or even months worth of data, giving those with access to large computational resources a significant advantage. The robot learning community is aware of this limitation to the extent that today's research on domain-specific simulators that can be massively scaled on modern hardware accelerators, not necessarily as fully-fledged as in the past, is surging.

This thesis approaches the problem of generating synthetic data for robot planning and control, focusing particularly on the needs of data-hungry methodologies like RL, and considering as targets multi-articulated robots like humanoids. It attempts to address questions regarding what kind of modern technology currently available, both hardware and software, suits this context best, taking into account domain-specific characteristics of legged

robotics. To this end, we first evaluate how general-purpose simulators widely adopted by the robotics community can be integrated into complex learning pipelines, guaranteeing the reproducibility of sampled trajectories. Then, motivated by the limited sampling performance of traditional general-purpose simulators running on Central Processing Units (CPUs), we explore available technology for maximising sampling throughput and propose a solution that, by exploiting modern hardware accelerators, enables to scale rigid-body simulations horizontally over a massive amount of parallel instances. We believe that the progress of research in this domain is one of the critical factors that could lead to the emergence of the next generation of robots seamlessly operating around us.

This research project is part of a split-site Ph.D. programme between the Istituto Italiano di Tecnologia (Italian Institute of Technology) and the University of Manchester, from November 2018 to July 2022. In the continuation of this section, we provide a short outline describing the structure of this thesis.

Part I: Background and Fundamentals

This part introduces the reader to the fundamental concepts behind the contributions of this thesis, reviews the state-of-the-art of corresponding domains, and provides a detailed overview of research output supporting the contributions to knowledge.

- Chapter 1 introduces robotic simulators, defining their main components and properties. It also describes the enabling technologies that made the work of this thesis possible, such as the Gazebo Sim simulator and the JAX framework.
- Chapter 2 introduces the notation and the equations governing rigid multibody dynamics, and how relevant dynamics and kinematics quantities could be computed and propagated through the model of a robotic mechanical structure.
- Chapter 3 introduces the main concepts and notation of RL. It formulates the structure of the RL problem, describes the main families of algorithms that could be used to compute a solution, and develops the theory behind

policy optimisation, reaching the formulation of the Proximal Policy Optimization algorithm.

- Chapter 4 reviews the state-of-the-art of the domains of Reinforcement Learning applied to robot locomotion, simulators for robot learning, and methodologies for push recovery. It also details open problems that characterise these domains, and how the contributions to knowledge of this thesis aim to solve them.

Part II: Contributions

This part presents the contributions to knowledge provided by this thesis, whose motivations will be discussed in more detail in Section 4.4.

- Chapter 5 presents a software architecture for creating robotic environments for RL research. It shows how to obtain environments that can be executed in both simulated and real-world settings, without the need to rewrite the decision-making logic. Sampling data from simulated environments is performed with the Gazebo Sim general-purpose simulator.

- Chapter 6, by sampling experience with the framework presented in the previous chapter, studies the problem of synthesising the appropriate control signals for balancing a simulated humanoid robot iCub in the presence of external disturbances. It frames the objective as a RL problem, guiding the exploration process during policy training with a reward shaping methodology. Terms computed from the dynamical description of the robot are included in the reward signal, introducing prior knowledge to the problem. Results show that after training, multiple push-recovery strategies emerge, and the policy is capable of selecting the most appropriate one as a consequence of external pushes.

- Chapter 7 starts addressing the problem of optimising the generation of synthetic experience for robot locomotion, whose performance was a bottleneck in the experiment presented in the previous chapter. This chapter provides a state-space representation that models the dynamics of a floating-base robot that can be integrated numerically to simulate its

evolution. It formulates a soft-contacts model to compute the interaction forces between the robot and the terrain surface, supporting both static (sticking) and dynamic (slipping) regimes, having a friction cone boundary without approximations. The dynamics of the contact model are then included in an extended state-space representation, obtaining a system of differential equations that describe the contact-aware dynamics of a floating-base robot.

- Chapter 8, by exploiting the contact-aware state-space representation formulated in the previous chapter, presents a new physics engine in reduced coordinates that can be executed on hardware accelerators like GPUs and TPUs for maximising sampling throughput. To this end, with the notation introduced in Chapter 2, this chapter also formulates canonical Rigid Body Dynamics Algorithms that can be executed in this accelerated context. The physics engine performance is then benchmarked, assessing the accuracy and speed of its algorithms and the scaling properties when executed in a highly parallel setting integrating hundreds or thousands of robot models concurrently. Finally, we validate the performance of the proposed physics engine by training a policy by sampling experience from hundreds of parallel environments running on GPU and evaluate its performance in a sim-to-sim setting representing an out-of-distribution environment.

RESEARCH PUBLICATIONS

The content of Chapter 5 appears in

> **Gym-Ignition: Reproducible Robotic Simulations for Reinforcement Learning**
> Diego Ferigo, Silvio Traversaro, Daniele Pucci
> *Robotics: Science and Systems (RSS) - Workshop on Closing the Reality Gap in Sim2real Transfer for Robotic Manipulation*, 2019

> **Gym-Ignition: Reproducible Robotic Simulations for Reinforcement Learning**
> Diego Ferigo, Silvio Traversaro, Giorgio Metta, Daniele Pucci
> *International Symposium on System Integration (SII)*, 2020

The content of Chapter 6 appears in

> **On the Emergence of Whole-body Strategies from Humanoid Robot Push-recovery Learning**
> Diego Ferigo, Raffaello Camoriano, Paolo Maria Viceconte, Daniele Calandriello, Silvio Traversaro, Lorenzo Rosasco, Daniele Pucci
> *Robotics and Automation Letters*, 2021

Beyond the previous main contributions directly supporting the contents of this thesis, during the research project I also contributed to other works whose content has not been included:

> **Simultaneous Floating-Base Estimation of Human Kinematics and Joint Torques**
> Claudia Latella, Silvio Traversaro, Diego Ferigo, Yeshasvi Tirupachuri, Lorenzo Rapetti, Francisco Javier Andrade Chavez, Francesco Nori, Daniele Pucci
> *Sensors*, 2019

> **A Human Wearable Framework for Physical Human-Robot Interaction**
> Claudia Latella, Yeshasvi Tirupachuri, Lorenzo Rapetti, Diego Ferigo, Silvio Traversaro, Ines Sorrentino, Francisco Javier Andrade Chavez, Francesco Nori, Daniele Pucci
> *I-RIM*, 2019

Learning to Sequence Multiple Tasks with Competing Constraints

Anqing Duan, Raffaello Camoriano, Diego Ferigo, Yanlong Huang, Daniele Calandriello, Lorenzo Rosasco, Daniele Pucci

International Conference on Intelligent Robots and Systems (IROS), 2019

Whole-Body Geometric Retargeting for Humanoid Robots

Kourosh Darvish, Yeshasvi Tirupachuri, Giulio Romualdi, Lorenzo Rapetti, Diego Ferigo, Francisco Javier Andrade Chavez, Daniele Pucci

International Conference on Humanoid Robots (Humanoids), 2019

A generic synchronous dataflow architecture to rapidly prototype and deploy robot controllers

Diego Ferigo, Silvio Traversaro, Francesco Romano, Daniele Pucci

International Journal of Advanced Robotic Systems, 2020

Towards Partner-Aware Humanoid Robot Control Under Physical Interactions

Yeshasvi Tirupachuri, Gabriele Nava, Claudia Latella, Diego Ferigo, Lorenzo Rapetti, Luca Tagliapietra, Francesco Nori, Daniele Pucci

Advances in Intelligent Systems and Computing, 2020

Learning to Avoid Obstacles With Minimal Intervention Control

Anqing Duan, Raffaello Camoriano, Diego Ferigo, Yanlong Huang, Daniele Calandriello, Lorenzo Rosasco, Daniele Pucci

Frontiers in Robotics and AI, 2020

ADHERENT: Learning Human-like Trajectory Generators for Whole-body Control of Humanoid Robots

Paolo Maria Viceconte, Raffaello Camoriano, Giulio Romualdi, Diego Ferigo, Stefano Dafarra, Silvio Traversaro, Giuseppe Oriolo, Lorenzo Rosasco, Daniele Pucci

Robotics and Automation Letters, 2022

Robot Platforms and Simulators

Diego Ferigo, Alberto Parmiggiani, Elena Rampone, Vadim Tikhanoff, Silvio Traversaro, Daniele Pucci, Lorenzo Natale

Cognitive Robotics, Chapter 7, 2022

SOFTWARE PROJECTS

The results of the research conducted for this thesis produced the following software projects that I developed and I am maintaining:

`scenario` is an abstraction layer providing APIs to interact with simulated and real robots. The goal of the project is to allow developing high-level code that can target all the available implementations using the same function calls. Currently, it provides a complete implementation for interfacing with robots simulated using the Gazebo Sim general-purpose simulator. The APIs are developed and available in C++. Python bindings are also provided. The library is open-source, released under the *LGPL v2.1 or any later version*, and it is available at `https://github.com/robotology/gym-ignition/tree/master/scenario`.

`gym-ignition` is a Python framework to create reproducible robotics environment for RL research. It exposes an abstraction layer providing APIs that enable the development of RL environment compatible with `gym.Env`. The resulting environments, if they exploit `scenario`, become agnostic from the setting in which they execute (either simulated or in real-time). The project also provides helpful utilities to compute common quantities commonly used by robotic environments, includes support of environment randomization, and handles the correct propagation of randomness. The library is open-source, released under the *LGPL v2.1 or any later version*, and it is available at `https://github.com/robotology/gym-ignition`.

`gym-ignition-models` is a Python project providing model descriptions tuned to be used in the Gazebo Sim simulator supported by `gym-ignition`. The library is open-source, released under the *LGPL v2.1 or any later version*, and it is available at `https://github.com/robotology/gym-ignition-models`.

`jaxsim` is a scalable physics engine in reduced coordinates implemented with JAX. It is developed in Python and supports most of the JAX features, including JIT compilation and auto-vectorization. Simulations can be executed on all the hardware supported by JAX, including CPUs, GPUs, and TPUs. The library is open-source, released under the *BSD-3-Clause*, and it is available at `https://github.com/ami-iit/jaxsim`.

Part I

BACKGROUND AND FUNDAMENTALS

1 SIMULATORS AND ENABLING TECHNOLOGIES

1.1 SIMULATORS FOR ROBOTICS

Real robots, in all their existing variety, consist of a complex, interconnected, and diverse set of systems communicating with each other. In order to get a robotic platform running, its mechanics, electronics, actuation, sensing, control system, and data transport have to be carefully designed, prototyped, assembled, calibrated, individually and collectively tested. Due to the tight interconnection of all the components part of a robotic platform, robot development requires multidisciplinary teams to work closely with each other throughout the entire process, from design to field deployment. In the past decades, all modern engineering fields operating in similar interconnected settings started employing mathematical models to describe their target systems.

In the robotics domain, rigid-body simulators became a standard component included in any practitioner's toolbox. Simulators allow studying the properties of robotics systems starting from their early design stage, anticipating possible errors that could lead to delays and failures, together with their corresponding cost. Restricting the domain to robot control, simulations became an important technology enabler thanks to the possibility of designing and prototyping algorithms on a model that captures the principal dynamics of the real system. Other important benefits include the execution in a controlled and repeatable environment, not subject to wear and tear, and not being vulnerable to costly damage in case of design failures.

This section describes the high-level software architecture of a robot simulator. We introduce and describe the main components existing simulators typically implement, and then identify a list of properties that could be used as a means of comparison.

1.1.1 Components

A *robotics simulator* is a collection of different independent components that, when combined, expose to the user a virtual environment where simulated robots can move and interact. Below we list and describe the main components that form a simulator.

DESCRIPTION PARSER A robot can be described as a set of *links* (or bodies) interconnected by a set of *joints*, which apply motion constraints and can provide actuation. Simulating a robot requires knowledge of its *kinematics*, encoding the topology and geometry of the links and joints, and its *dynamics*, encoding their physical properties. This information is usually provided in textual form by a structured *robot description*. Examples of common descriptions are the Unified Robot Description Format (URDF), Simulation Description Format (SDF), Mujoco XML Format (MJCF), Universal Scene Description (USD). These descriptions also typically include additional information about the robot, for example, how its visual appearance is rendered, how it collides with either itself or other simulated objects, the type and location of the on-board sensors, the joint actuation type and limits, etc. Simulators typically support one or more of these description formats, allowing them to import robots and create scenes where they can operate. The *description parser* is the component that reads the supported textual descriptions and imports the robot properties into the simulation.

PHYSICS ENGINE The central component of a simulator is its *physics engine*, responsible for implementing the behaviours governed by the physics laws of motion. It uses the information parsed from the robot description to predict how the dynamics evolve over time. Depending on how the joint constraints are simulated, we can categorise physics engines in *maximal coordinates* and *reduced coordinates*. Using maximal coordinates, each simulated body is treated separately in the Cartesian space, and the overall robot's dynamics is computed by solving an optimisation problem that applies explicit constraints to enforce on the kinematics the effects of the joints. Instead, using reduced coordinates, the system dynamics considers the mechanical structure as a whole and it implicitly enforces motion constraints induced by the joints. The physics engine

usually also includes routines of *collision detection* that, exploiting geometrical properties of the link's collisions shapes, allow to assess if bodies are in contact, and *contact models*, which compute interaction forces between colliding bodies. A general-purpose simulator either implements or interfaces with at least one physics engine, and it is not uncommon to find simulators exposing multiple physics engines.

PUBLIC APIS Simulators allow users to interface with their physics engine through a set of public Application Programming Interfaces (APIs). They typically expose the relevant quantities computed by the physics engine at any simulated time instant. Depending on the architecture, they could either allow to step forward the simulation manually and interact with it programmatically, or trigger an asynchronous starting signal and interact through a network transport layer. Simulators also typically expose relevant model[3] kinematics and dynamics properties computed by the physics engine.

SENSORS The most advanced simulators include the possibility to generate data from virtual sensors that mimic the behaviour of those mounted on the real robot, like Inertial Measurement Units (IMUs), cameras, Force/Torque (F/T) sensors, etc. Often, in order to reduce differences with the actual setup, they might allow injecting noise to sensor readings.

RENDERING The simulation of sensors like cameras requires the inclusion of a *rendering engine* to draw the environment where the simulated robot operates and captures its information. Many simulators implement rendering either by integrating external rendering engines or exposing new custom functionalities. Depending on the selected engine, rendering could be more or less realistic, with the cost of becoming the overall bottleneck of the simulation in case of too detailed rendering.

GRAPHICAL USER INTERFACE Simulators with rendering capabilities usually also implement a Graphical User Interface (GUI) to simplify the visualisation and, possibly, also the interaction with the simulated scene.

3 Simulated robots are often referred to as *models* in this thesis.

1.1.2 Properties

In this section, we provide a set of simulator properties that will be used in the following chapters as comparison metrics. We distill the properties introduced by Ferigo et al. [2020], maintaining only those related to generic simulators. In the next chapters, we will further specialise the analysis to simulators for robot learning .

MULTIPLE PHYSICS ENGINES Simulators could interface with either one or multiple physics engines, that can be selected by the user before launching the simulation. Having multiple choices is often beneficial because there are multiple methodologies to perform similar computations, and some of them could be more optimized for the target simulated context. Furthermore, it is common to have physics engines that outperform their alternatives in a narrow range of problems.

REPRODUCIBILITY A simulator is reproducible if consecutive executions of a scene starting from the same initial state and applying the same inputs yield exactly the same trajectory and final state. The main component that typically undermines reproducibility is a subtle consequence of the client-server architecture widely used by many simulators. Often, the physics engine and the user code that read data and sends commands reside on different threads or processes. The communication between them relies on sockets whose processes, depending on the system's load and the operating system's scheduler, can be preempted. Without complex synchronisation protocols, the user code might think to have stepped the simulator and read the most recent measurement even if the data might have been buffered. Therefore, even if the underlying physics engines, when called programmatically, would provide reproducible trajectories, simulators exposing their functionality using socket-based protocols could affect their reproducibility.

PARALLEL SIMULATION Most traditional robotic simulators have been designed to be executed in a single instance. Some of them, however, allow executing multiple parallel simulations, allowing user code to interface with

any of them. This property is typically implemented in those simulators that can be stepped programmatically, primarily because of limitations or challenges of network segmentation for those that rely on network transport as a mean for interfacing with the simulation.

ACCELERATED SIMULATION The ratio between real and simulated time is known as Real-Time Factor (RTF). A RTF of 1 means that one simulated second is executed in 1 real-world second, and a RTF of 2 means that the same simulated second is executed in 0.5 real-world seconds. Considering the typical usage for traditional robotic applications, many simulators aim to achieve a RTF of 1 when executed with all their features, rendering included. Usually, the RTF value defaults to 1 even when the simulation could run faster, and it is a configurable simulation parameter.

HEADLESS SIMULATION A simulation is defined as headless if it can be executed on machines without any display, like a server or a data center. Usually, there is no significant difference between a normal and a headless simulation. The physics can be executed in both settings regardless. However, not all simulators can render the scene on a headless setup when rendering is involved.

1.2 ENABLING TECHNOLOGIES

In this section, we describe the technology enablers that made developing the experiments presented in this thesis possible.

1.2.1 Gazebo Sim

Gazebo [Koenig et al., 2004], developed by Open Robotics, is among the most used and widely adopted simulators by the robotics community. It interfaces with multiple physics engines like ODE[4], bullet [Coumans et al., 2016], and DART [Jeongseok Lee et al., 2018]. It supports loading descriptions of robots defined either with the URDF or the SDF. It also supports the simulation of a wide range of commonly used sensors like IMUs, cameras, F/T sensors, etc.

In the Artificial Mechanical Intelligence[5] laboratory, we have always based our simulation stack on Gazebo. Over the years, we developed the model description of our robots[6] and built an entire infrastructure[7] for this simulator. However, while previous attempts[8] [Zamora et al., 2017; Lopez et al., 2019] tried to integrate Gazebo into a RL training pipeline, the performance that could be obtained together with the need to execute it in a separated process, always prevented its wide adoption by the robot learning community.

After more than 15 years of development, Open Robotics started the development of a new simulator, representing the next generation of Gazebo, that from now on we will call *Gazebo Classic* for the sake of clarity. The new simulator, initially known as Ignition Gazebo and later rebranded as *Gazebo Sim*, is a modular suite of libraries partially extracted from the monolithic architecture of its predecessor. Gazebo Sim, contrarily to its predecessor, offers programmatic APIs to instantiate and step the simulator, enabling users to obtain a finer control of the simulation cycle.

One of the simulation architectures presented in this thesis is based on Gazebo Sim. A more detailed overview of the features that motivated the adoption of the simulator and why they represent a valid choice for the contributed architecture is discussed in more detail in Section 5.3.1.

4 https://www.ode.org
5 https://ami.iit.it/
6 https://github.com/robotology/icub-models
7 https://github.com/robotology/gazebo-yarp-plugins
8 http://wiki.ros.org/openai_ros

1.2.2 The iCub humanoid robot

The iCub humanoid robot [Natale et al., 2017] is an open-source robot platform developed and produced by iCub Tech at the Italian Institute of Technology. It was first developed as part of the RobotCup project [Metta et al., 2005], and nowadays more than 40 replicas have been built and distributed worldwide.

iCub v2.5 is 104 cm tall and weighs approximately 33 kg. Its mechanical structure is characterised by 53 Degrees of Freedom (DoFs), including those belonging to the hands and the eyes. For motion control applications, and particularly whole-body locomotion, typically only 23 DoFs of its body are considered: 6 in each leg, 4 in each arm, and 3 in the torso.

For the work presented in this thesis, only details about its description for simulation purposes are necessary to be specified. The kinematics, the inertial parameters, and the visual meshes of the robot are automatically generated[9] from its CAD design. Their accuracy has always been sufficient for developing highly-dynamic balancing controllers [Pucci et al., 2016] and walking algorithms [Dafarra et al., 2018]. The official URDF models[10] have been slightly updated[11] to be imported in Gazebo Sim.

1.2.3 JAX

JAX [Frostig et al., 2018; Bradbury, James et al., 2018] is a software framework for high-performance numerical computing and machine learning research. Combining the features of Autograd [Maclaurin et al., 2015] and Accelerated Linear Algebra (XLA) [Sabne, 2020], JAX enables the development of fast algorithms with native support of Automatic Differentiation (AD) [Baydin et al., 2018] with the Python programming language. These features are exposed when operating on JAX arrays, having an interface compatible with NumPy [Harris et al., 2020].

Libraries and functions developed with JAX benefit from the following key features of the framework:

9 https://github.com/robotology/icub-models-generator
10 https://github.com/robotology/icub-models/
11 https://github.com/ami-iit/gym-ignition-models

- Function calls developed in Python are compiled in Just-in-time (JIT) at their first execution by XLA. JAX inherits all the hardware supported by XLA, therefore the same Python code can be compiled and executed transparently on CPUs, GPUs, and TPUs Being a domain-specific linear algebra compiler, XLA can generate high-performance kernels for scientific computing. Advanced branching is also supported, including loops, ifs, recursions, and closures.

- Any logic operating on JAX arrays can be parallelized on the target hardware accelerator thanks to the native support of *auto-vectorization*. This feature enables the development of algorithms that can be seamlessly scaled horizontally by just providing inputs with an additional dimension representing the batch axis. To maximise performance, vectorized code can also be combined with JIT compilation.

- Any logic operating on JAX arrays can be automatically differentiated using either forward or reverse mode [Blondel et al., 2022]. JAX allows differentiating all the code that can be JIT-compiled, including logic that uses advanced branching. The AD implementation also allows the computation of higher-order derivatives.

At the time of writing, the most advanced XLA backends are those targeting TPUs and GPUs. The JIT compilation of code for CPUs, depending on the complexity of the logic, could take a long time as it can use just one compilation thread.

All models are wrong.
But some are useful.
— George Box

2 ROBOT MODELLING

In this chapter, we introduce the mathematical formulation used throughout the thesis to describe a *floating-base multibody system*. Robots can be modelled as a set of rigid bodies connected by joints constraining their relative motion. After an initial overview of the necessary mathematical background, we introduce the notion of reference frames and how they relate to the kinematics of the bodies belonging to the system. We continue by introducing the inertial properties of a rigid body, and derive the Newton-Euler equation that represent the dynamics of its motion. Then, we show how the relative motion between rigid bodies can be constrained by providing the joint model considered in this thesis. Finally, by combining the properties of bodies and joints, we describe how a multibody system can be modelled and derive its Equations of Motion. We focus our analysis on floating-base systems, since fixed-base systems could be seen as a particular case in which the base is anchored to the world.

The purpose of the resulting floating-base model is twofold. Firstly, assuming we know the kinematics and dynamics properties of a real robot, it allows us to compute many relevant quantities required by control algorithms. Secondly, it enables the definition of a dynamic system that can be used to create and perform simulations of real robots operating in an environment. We also provide the necessary notions to develop widely used RBDA that efficiently compute relevant quantities of the system's dynamics.

We adopt the unified view of the Equations of Motion (EoMs) proposed by Traversaro et al. [2017], slightly adapted to employ the notation summarised in Traversaro et al. [2019]. Minor modifications are introduced to suit the simulation setting better.

2.1 NOTATION

- The set of real numbers is denoted by $\mathbb{R}$. Let $\mathbf{u}$ and $\mathbf{v}$ be two n-dimensional column vectors of real numbers, i.e. $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, then their inner product is denoted as $\mathbf{u}^\top \mathbf{u}$, where $(\cdot)^\top$ is the transpose operator.
- The identity matrix of size n is denoted by $\mathbf{I}_n \in \mathbb{R}^{n \times n}$; the zero column vector of dimension n is denoted by $\mathbf{0}_n \in \mathbb{R}^n$; the zero matrix of dimension $n \times m$ is denoted by $\mathbf{0}_{n \times m} \in \mathbb{R}^{n \times m}$; the all-ones matrix of dimension $n \times m$ is denoted by $\mathbf{1}_{n \times m} \in \mathbb{R}^{n \times m}$.
- The set $\mathrm{SO}(3)$ is the set of $\mathbb{R}^{3\times 3}$ orthogonal matrices with determinant equal to one, namely:

$$\mathrm{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3\times 3} \,|\, \mathbf{R}^\top \mathbf{R} = \mathbf{I}_3, \det(\mathbf{R}) = 1\}.$$

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

$$\mathfrak{so}(3) = \{S \in \mathbb{R}^{3\times 3} \,|\, S^\top = -S\}.$$

- The set $\mathrm{SE}(3)$ is defined as

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

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

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

- Given a vector $\mathbf{w} = (x, y, z) \in \mathbb{R}^3$, we define $\mathbf{w}^\wedge \in \mathfrak{so}(3)$ (read $\mathbf{w}$ *hat*) as the 3×3 *skew-symmetric* matrix

$$\mathbf{w}^\wedge = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^\wedge = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}.$$

Given the *skew-symmetric* matrix $\mathbf{W} = \mathbf{w}^\wedge$, we define $\mathbf{W}^\vee$ (read $\mathbf{W}$ *vee*) as

$$\mathbf{W}^\vee = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}^\vee = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

The vee operator is the inverse of the hat operator.

- Given two 3D vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$, the hat operator can be used to compute their cross product:

$$\mathbf{a} \times \mathbf{b} = \mathbf{a}^\wedge \, \mathbf{b} = -\mathbf{b}^\wedge \, \mathbf{a}.$$

- Given a vector $\mathbf{w} \in \mathbb{R}^3$ and a matrix $\mathbf{R} \in \mathrm{SO}(3)$, the following property of the hat operator holds:

$$(\mathbf{R}\mathbf{w})^\wedge = \mathbf{R}\mathbf{w}^\wedge \, \mathbf{R}^\top.$$

- Given a vector $\mathbf{v} = (\boldsymbol{v}, \boldsymbol{\omega}) \in \mathbb{R}^6$ with $\boldsymbol{v}, \boldsymbol{\omega} \in \mathbb{R}^3$, we define

$$\mathbf{v}^\wedge = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}^\wedge = \begin{bmatrix} \boldsymbol{\omega}^\wedge & \boldsymbol{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \in \mathfrak{se}(3),$$

and its inverse

$$\begin{bmatrix} \boldsymbol{\omega}^\wedge & \boldsymbol{v} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix}^\vee = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} = \mathbf{v} \in \mathbb{R}^6.$$

2.2 POINTS, FRAMES, ROTATIONS, TRANSFORMATIONS

Let's consider a *point* $\mathbf{p}$. Its existence is absolute, meaning it occupies a given position in space. For practical reasons, describing its location in space with a tuple of real numbers is convenient. Therefore, we introduce a *reference frame* A, defined as the combination of a point $\mathbf{o}_A$, called origin, and an orthogonal 3D orientation frame $[A]$, defined by the unit vectors $(\vec{\mathbf{x}}_A, \vec{\mathbf{y}}_A, \vec{\mathbf{z}}_A)$. More compactly, we write $A = (\mathbf{o}_A, [A])$. If $\vec{\mathbf{r}}_{\mathbf{o}_A,\mathbf{p}}$ is the geometric 3D vector that connects the origin of frame A with the point $\mathbf{p}$, having direction from $\mathbf{o}_A$ to $\mathbf{p}$, we can obtain the *coordinate vector* ${}^A\mathbf{p} \in \mathbb{R}^3$ of point $\mathbf{p}$ expressed in the orientation frame $[A]$ as follows:

$$
{}^A\mathbf{p} = \begin{bmatrix} \vec{\mathbf{r}}_{\mathbf{o}_A,\mathbf{p}} \cdot \vec{\mathbf{x}}_A \\ \vec{\mathbf{r}}_{\mathbf{o}_A,\mathbf{p}} \cdot \vec{\mathbf{y}}_A \\ \vec{\mathbf{r}}_{\mathbf{o}_A,\mathbf{p}} \cdot \vec{\mathbf{z}}_A \end{bmatrix},
$$

where $(\cdot)$ denotes the scalar product between vectors. The same notation applies to denote the coordinates of a frame's origin w.r.t. a different frame. If $B = (\mathbf{o}_B, [B])$, the coordinates of its origin with respect to frame A are denoted as ${}^A\mathbf{o}_B$.

Definition 2.2.1 (World frame)**.** Newton's mechanics requires the existence of an *inertial frame*. We denote this special frame as W, and call it *world* frame. As a common practice, we ignore the non-inertial effects caused by the Earth's motion, and assume the world frame to be fixed on the surface. □

Given two frames A and B, we can introduce the *coordinate transformation* from frame B to A as ${}^A\mathbf{R}_B \in \mathrm{SO}(3)$, also referred as *rotation matrix*. This transformation depends only on the orientation of the frames, $[A]$ and $[B]$, and not on their origins. Given a point $\mathbf{p}$ and assuming $\mathbf{o}_A = \mathbf{o}_B$, it follows that ${}^A\mathbf{p} = {}^A\mathbf{R}_B\, {}^B\mathbf{p}$.

Similarly, if we want to describe in compact form both the position and the orientation of frame B w.r.t. frame A, we can use the 4×4 *homogeneous transformation matrix*, also referred to more concisely as *transform*:

$$ {}^{A}\mathbf{H}_{B} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{o}_{B} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \in \mathrm{SE}(3). $$

If we introduce the *homogeneous representation* of two coordinate vectors ${}^{A}\mathbf{p}$ and ${}^{B}\mathbf{p}$ as ${}^{A}\tilde{\mathbf{p}} = ({}^{A}\mathbf{p}; 1) \in \mathbb{R}^4$ and ${}^{B}\tilde{\mathbf{p}} = ({}^{B}\mathbf{p}; 1) \in \mathbb{R}^4$, the transform can also be used as a map between their coordinates ${}^{A}\tilde{\mathbf{p}} = {}^{A}\mathbf{H}_{B}\,{}^{B}\tilde{\mathbf{p}}$, allowing a compact representation of the roto-translation ${}^{A}\mathbf{p} = {}^{A}\mathbf{o}_{B} + {}^{A}\mathbf{R}_{B}\,{}^{B}\mathbf{p}$.

It can be shown that, given a transform ${}^{A}\mathbf{H}_{B}$, we can express its inverse as follows:

$$ {}^{B}\mathbf{H}_{A} = {}^{A}\mathbf{H}_{B}^{-1} = \begin{bmatrix} {}^{B}\mathbf{R}_{A} & {}^{B}\mathbf{o}_{A} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} {}^{A}\mathbf{R}_{B}^{\top} & -{}^{B}\mathbf{R}_{A}\,{}^{A}\mathbf{o}_{B} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}. \tag{2.1} $$

2.3 FRAME VELOCITY

The velocity of a frame B relative to another frame A can be obtained by taking the time derivative of the transform that defines its pose w.r.t. A:

$$ {}^{A}\dot{\mathbf{H}}_{B} = \frac{\mathrm{d}}{\mathrm{d}t}\left({}^{A}\mathbf{H}_{B}\right) = \begin{bmatrix} {}^{A}\dot{\mathbf{R}}_{B} & {}^{A}\dot{\mathbf{o}}_{B} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix}. \tag{2.2} $$

In this section, we show how we can obtain the 6D velocity $\mathbf{v}_{A,B} \in \mathbb{R}^6$ of a frame composed of vertically stacked linear and angular parts by manipulating Equation (2.2). We will see that, through the process of *trivialization*, we can obtain different types of 6D velocities called *velocity representations* depending on the considered reference frame. Finally, we show how to change the reference frame of the 6D velocity, obtaining an equation comparable to the one used to roto-translate the coordinates of points for the velocities.

Before proceeding, we need to show how the term ${}^{A}\dot{\mathbf{R}}_{B}$ can be formulated in terms of an angular velocity $\boldsymbol{\omega}_{A,B}$, that can be expressed either in A or B coordinates.

Definition 2.3.1 (Time derivative of the rotation matrix)**.** A commonly used formula to express the time derivative of a rotation matrix is the following:

$$
{}^{A}\dot{\mathbf{R}}_{B} = {}^{A}\boldsymbol{\omega}_{A,B}^{\wedge}\, {}^{A}\mathbf{R}_{B} = {}^{A}\mathbf{R}_{B}\, {}^{B}\boldsymbol{\omega}_{A,B}^{\wedge},
$$

where $\boldsymbol{\omega}_{A,B} \in \mathbb{R}^3$ is the angular velocity of frame B relative to frame A, that could be expressed either in A coordinates (${}^{A}\boldsymbol{\omega}_{A,B}$) or in B coordinates (${}^{B}\boldsymbol{\omega}_{A,B}$).

Proof. The time derivative of the orthogonality properties $\mathbf{R}^{\top}\mathbf{R} = \mathbf{I}_3$ and $\mathbf{R}\,\mathbf{R}^{\top} = \mathbf{I}_3$ of rotation matrices leads to:

$$
\begin{cases} \dot{\mathbf{R}}^{\top}\mathbf{R} + \mathbf{R}^{\top}\dot{\mathbf{R}} = \mathbf{0}_{3\times 3} \\ \dot{\mathbf{R}}\,\mathbf{R}^{\top} + \mathbf{R}\,\dot{\mathbf{R}}^{\top} = \mathbf{0}_{3\times 3} \end{cases} \rightarrow \begin{cases} S_l^{\top} + S_l = \mathbf{0}_{3\times 3} \\ S_r + S_r^{\top} = \mathbf{0}_{3\times 3} \end{cases},
$$

where we have introduced the left $S_l = \mathbf{R}^{\top}\dot{\mathbf{R}} \in \mathfrak{so}(3)$ and the right $S_r = \dot{\mathbf{R}}\,\mathbf{R}^{\top} \in \mathfrak{so}(3)$. Since both matrices are skew-symmetric, they can be parameterized by a vector $S^{\vee} = \boldsymbol{\omega}_{A,B} \in \mathbb{R}^3$. The matrix product is not commutative, therefore the vectors of the left and right cases must be different:

$$
\begin{aligned} \dot{\mathbf{R}} &= \mathbf{R}S_l = \mathbf{R}\, {}^{B}\boldsymbol{\omega}_{A,B}^{\wedge}, \\ \dot{\mathbf{R}} &= S_r\,\mathbf{R} = {}^{A}\boldsymbol{\omega}_{A,B}^{\wedge}\,\mathbf{R}, \end{aligned} \tag{2.3}
$$

where we have introduced the angular velocity $\boldsymbol{\omega}_{A,B}$ between frames A and B, expressed either in A or B coordinates. □

2.3.1 Left-trivialized velocity

The terms forming the left-trivialized velocity ${}^{B}\mathbf{v}_{A,B}$ of frame B relative to frame A can be obtained by left multiplying Equation (2.2) with ${}^{B}\mathbf{H}_{A}$:

$$
\begin{aligned}
{}^{B}\mathbf{H}_{A}\,{}^{A}\dot{\mathbf{H}}_{B} &= \begin{bmatrix} {}^{A}\mathbf{R}_{B}^{\top} & -{}^{A}\mathbf{R}_{B}^{\top}\,{}^{A}\mathbf{o}_{B} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} {}^{A}\dot{\mathbf{R}}_{B} & {}^{A}\dot{\mathbf{o}}_{B} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \\
&= \begin{bmatrix} {}^{A}\mathbf{R}_{B}^{\top}\,{}^{A}\dot{\mathbf{R}}_{B} & {}^{A}\mathbf{R}_{B}^{\top}\,{}^{A}\dot{\mathbf{o}}_{B} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \\
&= \begin{bmatrix} {}^{B}\boldsymbol{\omega}_{A,B}^{\wedge} & {}^{B}\boldsymbol{v}_{A,B} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \in \mathfrak{se}(3),
\end{aligned}
$$

where we exploited the form of the inverse transform introduced in Equation (2.1). The left-trivialized 6D velocity is obtained by stacking the linear and angular components of the left trivialization process:

$${}^{B}\mathbf{v}_{A,B} = \begin{bmatrix} {}^{B}\boldsymbol{v}_{A,B} \\ {}^{B}\boldsymbol{\omega}_{A,B} \end{bmatrix} = \begin{bmatrix} {}^{A}\mathbf{R}_{B}^{\top}\,{}^{A}\dot{\mathbf{o}}_{B} \\ \left({}^{A}\mathbf{R}_{B}^{\top}\,{}^{A}\dot{\mathbf{R}}_{B}\right)^{\vee} \end{bmatrix} \in \mathbb{R}^{6},$$

from which, by construction, it also follows:

$${}^{B}\mathbf{v}_{A,B}^{\wedge} = {}^{B}\mathbf{H}_{A}\,{}^{A}\dot{\mathbf{H}}_{B} \in \mathfrak{se}(3). \tag{2.4}$$

From this relation, we can start considering $\mathbf{v}_{A,B}$ as the 6D representation of an element of $\mathfrak{se}(3)$. The left-trivialized velocity is also called *body-fixed representation* of $\mathbf{v}_{A,B}$.

2.3.2 Right-trivialized velocity

The terms forming the right-trivialized velocity ${}^{A}\mathbf{v}_{A,B}$ of frame B relative to frame A, can be obtained by right multiplying Equation (2.2) with ${}^{B}\mathbf{H}_{A}$:

$$
\begin{aligned}
{}^{A}\dot{\mathbf{H}}_{B}\,{}^{B}\mathbf{H}_{A} &= \begin{bmatrix} {}^{A}\dot{\mathbf{R}}_{B} & {}^{A}\dot{\mathbf{o}}_{B} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \begin{bmatrix} {}^{A}\mathbf{R}_{B}^{\top} & -{}^{A}\mathbf{R}_{B}^{\top}\,{}^{A}\mathbf{o}_{B} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \\
&= \begin{bmatrix} {}^{A}\dot{\mathbf{R}}_{B}\,{}^{A}\mathbf{R}_{B}^{\top} & {}^{A}\dot{\mathbf{o}}_{B} - {}^{A}\dot{\mathbf{R}}_{B}\,{}^{A}\mathbf{R}_{B}^{\top}\,{}^{A}\mathbf{o}_{B} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \\
&= \begin{bmatrix} {}^{A}\boldsymbol{\omega}_{A,B}^{\wedge} & {}^{A}\boldsymbol{v}_{A,B} \\ \mathbf{0}_{1\times 3} & 0 \end{bmatrix} \in \mathfrak{se}(3).
\end{aligned}
$$

The right-trivialized 6D velocity is obtained by stacking the linear and angular components of the right trivialization process:

$${}^{A}\mathbf{v}_{A,B} = \begin{bmatrix} {}^{A}\boldsymbol{v}_{A,B} \\ {}^{A}\boldsymbol{\omega}_{A,B} \end{bmatrix} = \begin{bmatrix} {}^{A}\dot{\mathbf{o}}_{B} - {}^{A}\dot{\mathbf{R}}_{B}\,{}^{A}\mathbf{R}_{B}^{\top}\,{}^{A}\mathbf{o}_{B} \\ \left({}^{A}\dot{\mathbf{R}}_{B}\,{}^{A}\mathbf{R}_{B}^{\top}\right)^{\vee} \end{bmatrix} \in \mathbb{R}^{6},$$

from which, by construction, it also follows:

$${}^{A}\mathbf{v}_{A,B}^{\wedge} = {}^{A}\dot{\mathbf{H}}_{B}\,{}^{B}\mathbf{H}_{A} \in \mathfrak{se}(3).$$

The right-trivialized velocity is also called *inertial-fixed representation* of $\mathbf{v}_{A,B}$.

The linear component of the right-trivialized velocity can also be written in the following alternative form:

$${}^{A}\boldsymbol{v}_{A,B} = {}^{A}\dot{\mathbf{o}}_{B} + {}^{A}\mathbf{o}_{B}^{\wedge}\,{}^{A}\boldsymbol{\omega}_{A,B} = {}^{A}\dot{\mathbf{o}}_{B} + {}^{A}\mathbf{o}_{B} \times {}^{A}\boldsymbol{\omega}_{A,B}, \tag{2.5}$$

showing explicitly that, in this representation, the linear velocity is the sum of the linear velocity of the frame B origin and the external product between the angular velocity and the distance between frame origins. This result is compatible with the setting of a rotating non-inertial reference B frame relative to an inertial frame A.

2.3.3 Expressing 6D velocities in different frames

Let us consider a generic 6D velocity ${}^{B}\mathbf{v}_{C,D}$ between frames C and D, expressed in coordinates of frame B. It can be shown that it is possible to express the velocity in a new frame A as follows:

$$ {}^{A}\mathbf{v}_{C,D} = {}^{A}\mathbf{X}_{B}\, {}^{B}\mathbf{v}_{C,D}, $$

where we introduced the following linear transformation between frames A and B:

$$ {}^{A}\mathbf{X}_{B} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{o}_{B}^{\wedge}\, {}^{A}\mathbf{R}_{B} \\ \mathbf{0}_{3\times 3} & {}^{A}\mathbf{R}_{B} \end{bmatrix} \in \mathbb{R}^{6\times 6}. \tag{2.6} $$

Recalling that ${}^{A}\mathbf{o}_{B} = -\,{}^{A}\mathbf{R}_{B}\, {}^{B}\mathbf{o}_{A}$, it can be shown that the inverse velocity transformation is simply ${}^{B}\mathbf{X}_{A} = {}^{A}\mathbf{X}_{B}^{-1}$.

2.3.4 Mixed velocity representation

We have seen that the angular component of the left- and right-trivialized velocities correspond to the classic concept of *angular* velocity, as introduced in Definition 2.3.1. The different representations only relate the angular velocity $\boldsymbol{\omega}_{A,B}$ to reference frames, which could either be A or B.

The *linear* velocity, instead, is less intuitive. While it corresponds to the time derivative of $\mathbf{o}_B$ in the left-trivialized representation, the expression in the right-trivialization includes an additional term as reported in Equation (2.5) to account for non-inertial effects.

There are situations in which we desire to express the 6D velocity of a frame with just the time derivatives ${}^{A}\dot{\mathbf{o}}_{B}$ and ${}^{A}\boldsymbol{\omega}_{A,B}$. We can obtain such special 6D velocity by introducing a new frame $B[A] = (\mathbf{o}_B, [A])$, that is a frame whose origin coincides with the origin of frame B, with the orientation frame of A. In this frame, we can define the *mixed* velocity as follows:

$$ {}^{B[A]}\mathbf{v}_{A,B} = {}^{B[A]}\mathbf{X}_{B}\, {}^{B}\mathbf{v}_{A,B} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & 0 \\ 0 & {}^{A}\mathbf{R}_{B} \end{bmatrix} \begin{bmatrix} {}^{B}\mathbf{R}_{A}\, {}^{A}\dot{\mathbf{o}}_{B} \\ {}^{B}\boldsymbol{\omega}_{A,B} \end{bmatrix} = \begin{bmatrix} {}^{A}\dot{\mathbf{o}}_{B} \\ {}^{A}\boldsymbol{\omega}_{A,B} \end{bmatrix}. \tag{2.7} $$

2.3.5 Cross product on $\mathbb{R}^6$

From the definition of the left-trivialized velocity of Equation (2.4), we can obtain the relation:

$$ {}^{A}\dot{\mathbf{H}}_{B} = {}^{A}\mathbf{H}_{B}\, {}^{B}\mathbf{v}_{A,B}^{\wedge}. $$

We want to formulate a comparable expression also for the velocity transformation ${}^{A}\mathbf{X}_{B}$. Differentiating with respect to time its definition from Equation (2.6), it can be shown that the following expression can be obtained:

$$ {}^{A}\dot{\mathbf{X}}_{B} = {}^{A}\mathbf{X}_{B}\, {}^{B}\mathbf{v}_{A,B}\times, \tag{2.8} $$

where the last term is the matrix representation of the cross-product on $\mathbb{R}^6$, defined as:

$$ {}^{B}\mathbf{v}_{A,B}\times = \begin{bmatrix} {}^{B}\boldsymbol{\omega}_{A,B} & {}^{B}\boldsymbol{v}_{A,B}^{\wedge} \\ \mathbf{0}_{3\times 3} & {}^{B}\boldsymbol{\omega}_{A,B} \end{bmatrix}. $$

2.4 ACCELERATIONS AND FORCES

2.4.1 Accelerations

The acceleration of a frame B relative to another frame A and expressed in a generic frame C can be obtained by taking the time-derivative of the corresponding velocity:

$$ {}^{C}\dot{\mathbf{v}}_{A,B} = \frac{\mathrm{d}}{\mathrm{d}t}\left({}^{C}\mathbf{v}_{A,B}\right). $$

Extracting the left-trivialized velocity and using Equation (2.8), we can obtain:

$$ \begin{aligned} {}^{C}\dot{\mathbf{v}}_{A,B} &= \frac{\mathrm{d}}{\mathrm{d}t}\left({}^{C}\mathbf{X}_{B}\, {}^{B}\mathbf{v}_{A,B}\right) = {}^{C}\mathbf{X}_{B}\, {}^{B}\dot{\mathbf{v}}_{A,B} + {}^{C}\dot{\mathbf{X}}_{B}\, {}^{B}\mathbf{v}_{A,B} \\ &= {}^{C}\mathbf{X}_{B}\left({}^{B}\dot{\mathbf{v}}_{A,B} + {}^{B}\mathbf{v}_{C,B}\times\, {}^{B}\mathbf{v}_{A,B}\right) \end{aligned}, \tag{2.9} $$

where it can be noticed that in this case the expected mnemonic-friendly form ${}^{C}\dot{\mathbf{v}}_{A,B} = {}^{C}\mathbf{X}_{B}\,{}^{B}\dot{\mathbf{v}}_{A,B}$ does *not* hold. However, if either $C = A$ or $C = B$, the cross-product term is zero, and the equality holds. Under these conditions, we can define the following acceleration:

$$ {}^{C}\mathbf{a}_{A,B} = {}^{C}\mathbf{X}_{A}\,{}^{A}\dot{\mathbf{v}}_{A,B} = {}^{C}\mathbf{X}_{B}\,{}^{B}\dot{\mathbf{v}}_{A,B}. $$

We refer to ${}^{C}\dot{\mathbf{v}}_{A,B}$ as *apparent acceleration*, and to ${}^{C}\mathbf{a}_{A,B}$ as *intrinsic acceleration*. We can see that intrinsic accelerations are always built from either inertial-fixed or body-fixed apparent accelerations. Their usage is convenient because the relation ${}^{C}\mathbf{a}_{A,B} = {}^{C}\mathbf{X}_{B}\,{}^{B}\mathbf{a}_{A,B}$ always holds regardless to A, B, and C.

For some computation, there is another helpful formulation of the frame acceleration, that we will call *proper acceleration*. It consists of the intrinsic acceleration minus the gravitational effects, and it can be defined as follows:

$$ {}^{C}\bar{\mathbf{a}}_{A,B} = {}^{C}\mathbf{a}_{A,B} - {}^{C}\mathbf{X}_{A} \begin{bmatrix} {}^{A}\mathbf{R}_{W}\,{}^{W}\mathbf{g} \\ \mathbf{0}_3 \end{bmatrix}. \tag{2.10} $$

The bar notation can be thought mnemonically as minus gravity, whose definition is ${}^{W}\mathbf{g} = (0, 0, -g) \in \mathbb{R}^3$ with $g \in \mathbb{R}^+$.

2.4.2 Forces

Let's consider a 6D force $\mathbf{f}$ composed by stacking a linear force $\boldsymbol{f} \in \mathbb{R}^3$ and a torque $\boldsymbol{m} \in \mathbb{R}^3$. Its coordinates with respect to a frame B are denoted as:

$$ {}_{B}\mathbf{f} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{m} \end{bmatrix} \in \mathbb{R}^6. $$

Compared to 6D velocities, this notation only needs the specification of the frame where the force is *expressed*. Note that this does not mean that the force is applied to the origin of the frame where it is expressed. In fact, we can take a force ${}_{B}\mathbf{f}$ applied to the origin of frame B and expressed in the same frame, and change its coordinates to frame A with the following transformation:

$$ {}_{A}\mathbf{f} = {}_{A}\mathbf{X}^{B}\,{}_{B}\mathbf{f}. \tag{2.11} $$

The coordinate transformation of 6D forces, strictly related to the transformation of 6D velocities between the same frames, can be defined as:

$$ {}_A\mathbf{X}^B = {}^B\mathbf{X}_A^\top = \begin{bmatrix} {}^A\mathbf{R}_B & \mathbf{0}_3 \\ {}^A\mathbf{o}_B^\wedge \, {}^A\mathbf{R}_B & {}^A\mathbf{R}_B \end{bmatrix} \in \mathbb{R}^{6\times 6}. $$

From this relation, also ${}_A\mathbf{X}^B = {}^A\mathbf{X}_B^{-\top}$ follows.

Remark 2.4.1. The coordinate transformation of a 6D force can be expanded as follows:

$$ {}_A\mathbf{f} = \begin{bmatrix} {}_A\boldsymbol{f} \\ {}_A\boldsymbol{m} \end{bmatrix} = {}_A\mathbf{X}^B \, {}_B\mathbf{f} = \begin{bmatrix} {}^A\mathbf{R}_B \, {}_B\boldsymbol{f} \\ {}^A\mathbf{o}_B^\wedge \left({}^A\mathbf{R}_B \, {}_B\boldsymbol{f} \right) + {}^A\mathbf{R}_B \, {}_B\boldsymbol{m} \end{bmatrix}. $$

It can be notices that the application of ${}_B\mathbf{f}$ to a different frame A, beyond rotating its components $({}_B\boldsymbol{f}, {}_B\boldsymbol{m})$ with ${}^A\mathbf{R}_B$, requires the introduction of an additional angular term proportional to ${}_B\boldsymbol{f}$. This behaviour can be explained considering a simplified case of applying a pure linear force ${}_B\mathbf{f} = ({}_B\boldsymbol{f}, \mathbf{0}_3)$ to the origin of frame A. Changing the application point would produce a torque due to the moment arm between the origins of frames A and B. Since the transformation should not alter the physical effect (for example, a resulting acceleration of the frame), ${}_A\mathbf{X}^B$ introduces the additional term ${}^A\mathbf{o}_B^\wedge \left({}^A\mathbf{R}_B \, {}_B\boldsymbol{f} \right)$ that compensates the moment arm. □

The cross product on $\mathbb{R}^6$ for 6D forces can be obtained from the relation between the coordinate transformation of forces and velocities:

$$ {}_A\dot{\mathbf{X}}^B = {}_A\mathbf{X}^B \, {}^B\mathbf{v}_{A,B}\bar{\times}^*, $$

where the last term is the matrix representation of the cross-product in $\mathbb{R}^6$, defined as:

$$ {}^B\mathbf{v}_{A,B}\bar{\times}^* = \begin{bmatrix} {}^B\boldsymbol{\omega}_{A,B}^\wedge & \mathbf{0}_{3\times 3} \\ {}^B\boldsymbol{v}_{A,B}^\wedge & {}^B\boldsymbol{\omega}_{A,B}^\wedge \end{bmatrix}. $$

Note that the relation $({}^B\mathbf{v}_{A,B}\times)^{-\top} = {}^B\mathbf{v}_{A,B}\bar{\times}^*$ holds. The combination of the overline with the star marks mnemonically the $(\cdot)^{-\top}$ operator.

2.5 RIGID-BODY KINEMATICS

This section provides a mathematical description of the kinematics of a *rigid body*. We first provide the definition of a rigid body, and then define its position and velocity.

Definition 2.5.1 (Rigid Body)**.** A *Rigid Body* is a mathematical abstraction describing an arbitrary distribution of mass in the 3D space fixed with respect to a given frame B, called *body frame*. It is assumed not being subject to any internal deformation when external forces are applied. □

Definition 2.5.2 (Rigid Body Pose)**.** The *pose* of a rigid body associated with a frame B w.r.t. a generic frame A is defined by the transform ${}^{A}\mathbf{H}_{B} \in \mathrm{SE}(3)$. □

Definition 2.5.3 (Rigid Body Velocity)**.** The *velocity* of a rigid body associated with a frame B w.r.t. a generic frame A is denoted as $\mathbf{v}_{A,B} \in \mathbb{R}^6$. □

Remark 2.5.1 (Velocity Representations of a Rigid Body)**.** The velocity of a rigid body can be expressed in different representations, depending on the used trivialization as explained in Section 2.3. The terminology used for the velocity representations becomes straightforward when we consider $A = W$. In this case, the left-trivialized velocity would be ${}^{W}\mathbf{v}_{W,B}$, where it can be noticed that it is expressed in world (inertial) coordinates, from what the alternative *inertial-fixed* name derives. The right-trivialized velocity ${}^{B}\mathbf{v}_{W,B}$ follows a similar reasoning, with its alternative *body-fixed* name. Finally, the *mixed* velocity ${}^{B[W]}\mathbf{v}_{W,B}$ can be seen as related to the inertial-fixed representation for its linear part, and to the body-fixed representation for its angular part. □

Remark 2.5.2 (Terminology)**.** In this thesis, we also use the term *link* to refer to a rigid body, particularly when it is part of a multibody system. Furthermore, we often name the body with the letter of its corresponding frame, i.e. when we say body B we mean the body whose pose corresponds to frame B. □

2.6 RIGID-BODY DYNAMICS

2.6.1 Inertial parameters

In this section, we introduce all the inertial parameters necessary to describe the dynamics of a rigid body. Given a body B, in order to simplify the notation, we denote the coordinates of a point $\mathbf{p}_i$ belonging to the body and expressed in B as $\mathbf{r} = {}^{B}\mathbf{p}_i$.

- The *total mass* of the rigid body can be calculated by introducing the function $\rho(\cdot) : \mathbb{R}^3 \mapsto \mathbb{R}^+$ that maps each point of the body to its density, and integrating over the volume occupied by the body:

$$m = \iiint_{\mathbb{R}^3} \rho(\mathbf{r}) \, \mathrm{d}\mathbf{r} \in \mathbb{R}.$$

- The Center of Mass (CoM) of the body can be calculated as the average point of its density:

$$\mathbf{c} = {}^{B}\mathbf{p}_{CoM} = \frac{\iiint_{\mathbb{R}^3} \mathbf{r}\rho(\mathbf{r}) \, \mathrm{d}\mathbf{r}}{\iiint_{\mathbb{R}^3} \rho(\mathbf{r}) \, \mathrm{d}\mathbf{r}} = \frac{1}{m} \iiint_{\mathbb{R}^3} \mathbf{r}\rho(\mathbf{r}) \, \mathrm{d}\mathbf{r} \in \mathbb{R}^3. \tag{2.12}$$

- The *inertia tensor* of the body, describing all moments of inertia of a body rotating around a specific axis, can be computed as:

$$I = -\iiint_{\mathbb{R}^3} \rho(\mathbf{r}) \left(\mathbf{r}^{\wedge}\right)^2 \mathrm{d}\mathbf{r} \in \mathbb{R}^{3\times 3},$$

resulting from the following computation of the body *angular momentum* $\mathbf{h}^{\omega} \in \mathbb{R}^3$:

$$\begin{aligned}
\mathbf{h}^{\omega} = I\,\boldsymbol{\omega} &= \iiint_{\mathbb{R}^3} \rho(\mathbf{r}) \left(\mathbf{r} \times \boldsymbol{v}\right) \mathrm{d}\mathbf{r} = \iiint_{\mathbb{R}^3} \rho(\mathbf{r}) \left(\mathbf{r} \times \boldsymbol{\omega} \times \mathbf{r}\right) \mathrm{d}\mathbf{r} \\
&= \iiint_{\mathbb{R}^3} \rho(\mathbf{r}) \, \mathbf{r}^{\wedge} \left(\boldsymbol{\omega}^{\wedge}\mathbf{r}\right) \mathrm{d}\mathbf{r} = -\iiint_{\mathbb{R}^3} \rho(\mathbf{r}) \, \mathbf{r}^{\wedge} \left(\mathbf{r}^{\wedge}\,\boldsymbol{\omega}\right) \mathrm{d}\mathbf{r} \\
&= -\iiint_{\mathbb{R}^3} \rho(\mathbf{r}) \left(\mathbf{r}^{\wedge}\right)^2 \mathrm{d}\mathbf{r}\;\boldsymbol{\omega}.
\end{aligned}$$

- The *6D inertia matrix* of the body that unifies all the previous inertial properties can be defined as follows:

$$\mathbb{M} = \begin{bmatrix} m\mathbf{I}_3 & -(m\mathbf{c})^\wedge \\ (m\mathbf{c})^\wedge & I \end{bmatrix} \in \mathbb{R}^{6\times 6}.$$

Remark 2.6.1. When we need to denote inertia matrices of different bodies, we use subscripts $\mathbb{M}_{B1}, \mathbb{M}_{B2}$, etc. Note that the 6D inertia matrix definition is valid only in body frame, and we should have specified it with an additional prescript ${}_B\mathbb{M}_B$. In this thesis, we only need 6D inertia matrices expressed in body frames, therefore we will always omit the prescript. □

2.6.2 Equations of Motion

We have seen that the kinematics of a rigid body can be described with the position and velocity of its corresponding frame: ${}^W\mathbf{H}_B$ and ${}^W\dot{\mathbf{H}}_B$, respectively. The dynamics of the rigid body can be derived from the formulation of Lagrangian mechanics founded on the principle of least action [Bullo et al., 2004; Selig, 2005; Marsden, Jerrold E. et al., 2013; Maruskin, 2018].

Definition 2.6.1 (Lagrangian mechanics)**.** Lagrangian mechanics defines a *mechanical system* with a pair $(\mathcal{Q}, L)$ of a *configuration space* $\mathcal{Q}$ and a smooth function $L(\mathbf{q}, \dot{\mathbf{q}}) = K - U$ called *Lagrangian*. The Lagrangian takes as input the system configuration and the system velocity $(\mathbf{q}, \dot{\mathbf{q}}) \in \mathcal{Q} \times \mathcal{V}$, where $\mathcal{V}$ is the *tangent space* of $\mathcal{Q}$, and computes the difference between the *kinetic energy* K and *potential energy* U of the system. □

Definition 2.6.2 (Principle of Least Action)**.** The Principle of Least Action states that the trajectory $\mathbf{q}(t)$ of the system in the interval $t \in [0, T]$ is the stationary point that minimises the system's *action functional*:

$$\mathcal{S}[\mathbf{q}] = \int_0^T L(\mathbf{q}(t), \dot{\mathbf{q}}(t))\, \mathrm{d}t\,.$$

The variational principle, when applied to the action of a mechanical system, yields the system's Equations of Motion. □

Definition 2.6.3 (Euler-Lagrange equation)**.** The trajectory $\mathbf{q}(t)$ is the stationary point of the action functional $\mathcal{S}$ if and only if it satisfies the Euler-Lagrange equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{\mathbf{q}}} - \frac{\partial L}{\partial \mathbf{q}} = 0. \tag{2.13}$$

This equation holds in Euclidean space, when $\mathcal{Q} = \mathcal{V} = \mathbb{R}^n$. □

Depending on the system to be described, there may be many possible choices of the *generalised position* $\mathbf{q} \in \mathcal{Q}$ and its derivative $\dot{\mathbf{q}} \in \mathcal{V}$, that we will refer as *generalised velocity*. In the case of a rigid body, we can describe the system configuration with the kinematic quantities of its corresponding frame, i.e. using $\mathbf{q} = {}^{W}\mathbf{H}_{B} \in \mathrm{SE}(3)$ and $\dot{\mathbf{q}} = {}^{W}\dot{\mathbf{H}}_{B}$. With this choice of variables, the Lagrangian of the system is the following:

$$L\left({}^{W}\mathbf{H}_{B}, {}^{W}\dot{\mathbf{H}}_{B}\right) = K\left({}^{W}\mathbf{H}_{B}, {}^{W}\dot{\mathbf{H}}_{B}\right) - U\left({}^{W}\mathbf{H}_{B}\right).$$

Remark 2.6.2 (Extension to non-Euclidean spaces)**.** It can be noted that the choice of $(\mathbf{q}, \dot{\mathbf{q}}) = ({}^{W}\mathbf{H}_{B}, {}^{W}\dot{\mathbf{H}}_{B})$ implies that the configuration space is no longer Euclidean. In these circumstances, the Euler-Lagrange equation (2.13) does not hold. It can be shown that, introducing Lie theory, the Lagrangian formulation can be generalised to any smooth manifold, and the EoMs of the system can be obtained by applying the Euler-Poincaré equation [Marsden, Jerrold E. et al., 2013; Maruskin, 2018]. To this end, $\mathcal{Q}$ must belong to a group and $\mathcal{V}$ related to its Lie algebra (respectively, the $\mathrm{SE}(3)$ matrix Lie group and the $\mathfrak{se}(3)$ group in our case). In this background chapter, in order to help focus only on the most important theoretical results required to understand the topic discussed in this thesis, we will omit the mathematical details of differential geometry. The interested readers are recommended to refer to specific textbooks [Warner, 1983; Selig, 2005] for a rigorous derivation of the theory of Lie groups and differential manifolds. □

Traversaro [2017] shows that transforming the system's velocity helps reduce the complexity of the equations, and introduces the *left-trivialized Lagrangian*, that takes as inputs $({}^{W}\mathbf{H}_{B}, {}^{B}\mathbf{v}_{W,B})$, where the velocity is now trivialized in body coordinates with Equation (2.4). In the following sections, we will always

assume quantities expressed in body-fixed representation, and refer to the rigid body pose and body-fixed velocity with $\mathbf{H}$ and $\mathbf{v}$, respectively. With this notation, the *left-trivialized Lagrangian* is defined as:

$$\ell(\mathbf{H}, \mathbf{v}) = L(\mathbf{H}, \mathbf{H}\,\mathbf{v}^\wedge) = \kappa(\mathbf{v}) - U(\mathbf{H}), \tag{2.14}$$

where we used the relation $\dot{\mathbf{H}} = \mathbf{H}\,\mathbf{v}^\wedge$ of Equation (2.4), and introduced the *trivialized kinetic energy* $\kappa(\cdot)$ that can be shown to only depend on $\mathbf{v}$ [Traversaro, 2017, Remark 2.5]. The trivialized kinetic energy and the potential energy can be computed as:

$$\kappa(\mathbf{v}) = \frac{1}{2}\,\mathbf{v}^\top\,\mathbb{M}\,\mathbf{v}, \tag{2.15}$$

$$U(\mathbf{H}) = \begin{bmatrix}\mathbf{g}^\top & \mathbf{0}_{1\times 3}\end{bmatrix} m\,\mathbf{H}\begin{bmatrix}\mathbf{c}\\ 1\end{bmatrix}, \tag{2.16}$$

where $\mathbf{g} = {}^W\mathbf{g} \in \mathbb{R}^3$ is the gravitational acceleration vector, and $\mathbf{c}$ the displacement between of the CoM of the body and its frame, as defined in Equation (2.12).

We now note that the sets of the system's positions and velocities are connected by a relationship between ${}^W\mathbf{H}_B \in \mathrm{SE}(3)$ and $\mathbf{v}^\wedge_{W,B} \in \mathfrak{se}(3)$. This relation enables the application of the Euler-Poincaré equation to obtain the system's dynamics. As shown by Traversaro [2017, Section 2.6.2], the resulting EoMs of the rigid body are the following:

$$\begin{cases} \dot{\mathbf{H}} = \mathbf{H}\,\mathbf{v}^\wedge & \text{(2.17a)}\\[2ex] \mathbb{M}\,\dot{\mathbf{v}} + \mathbf{v}\bar{\times}^*\,\mathbb{M}\,\mathbf{v} = \mathbb{M}\begin{bmatrix}\mathbf{R}^\top\,\mathbf{g}\\ \mathbf{0}_{3\times 1}\end{bmatrix} + {}_B\mathbf{f}^{ext} & \text{(2.17b)}\end{cases}$$

which includes the influence of additional non-gravitational external 6D forces ${}_B\mathbf{f}^{ext}$ acting on the body [Bullo et al., 2004]. Equation (2.17b) is the Newton-Euler equation of the rigid body.

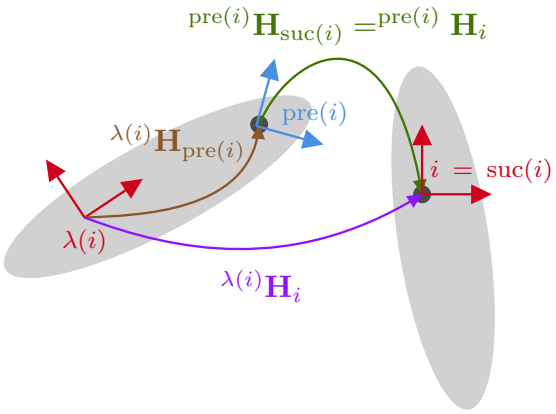


Figure 2.1: Illustration of the joint model. The frame of the parent link P is shown as $\lambda(i)$, and the frame of the child link C is shown as i. The i-th joint connects together the two links, enforcing a motion constraint. The operators $\text{pre}(\cdot)$ and $\text{suc}(\cdot)$ accept a joint number and return, respectively, its predecessor and successor frames. The successor frame of a joint matches with the frame i of the child link. The joint model aims to obtain the transform ${}^{\lambda(i)}\mathbf{H}_i(s)$ as a function of the joint position s.

Remark 2.6.3. Equation (2.17b), in its form with full notation, is expressed as follows:

$$\mathbb{M}_B \, {}^B\dot{\mathbf{v}}_{W,B} + {}^B\mathbf{v}_{W,B} \bar{\times}^* \, \mathbb{M}_B \, {}^B\mathbf{v}_{W,B} = \mathbb{M}_B \begin{bmatrix} {}^B\mathbf{R}_W \, {}^W\mathbf{g} \\ \mathbf{0}_{3\times 1} \end{bmatrix} + {}_B\mathbf{f}^{ext}.$$

If we bring the gravitational effect to the left hand side:

$$\mathbb{M}_B \left({}^B\dot{\mathbf{v}}_{W,B} - \begin{bmatrix} {}^B\mathbf{R}_W \, {}^W\mathbf{g} \\ \mathbf{0}_{3\times 1} \end{bmatrix} \right) + {}^B\mathbf{v}_{W,B} \bar{\times}^* \, \mathbb{M}_B \, {}^B\mathbf{v}_{W,B} = {}_B\mathbf{f}^{ext},$$

we can simplify this equation using the proper acceleration:

$$\mathbb{M}_B \, {}^B\bar{\mathbf{a}}_{W,B} + {}^B\mathbf{v}_{W,B} \bar{\times}^* \, \mathbb{M}_B \, {}^B\mathbf{v}_{W,B} = {}_B\mathbf{f}^{ext}. \tag{2.18}$$

This equation will be useful in the definition of recursive algorithms for rigid body dynamics.

2.7 JOINT MODEL

The second fundamental element for modelling a multibody system is the *joint*. All links part of the system are characterised by 6 DoF that, ignoring for the moment possible collisions that could occur, are free to evolve in space independently of each other. Joints can be used to connect links together and act as constraints that limit their relative motion. Each joint is characterised by its number of DoFs, which can range from 0 to 6 and, considering the relative position between two links as a local topological space, describes its dimension.

Assumption 2.7.1. This thesis only considers multibody systems modelled with 1 DoF joints. All the theories and algorithms proposed in the following chapters can be extended with minor modifications to other less common multi-DoF joint types [Featherstone, 2008]. □

The most common 1-DoF joints used in robotics are called *revolute* and *prismatic*. They impose motion constraints on 5 dimensions of the local space, therefore their configuration s is an element of $\mathbb{R}$. We model a joint as a time-varying transformation between the frames P and C of its *parent* and *child* links:

$$^{P}\mathbf{H}_{C}(s) : \mathbb{R} \to \mathrm{SE}(3).$$

As illustrated in Figure 2.1, we break down this parent-to-child transform in two different components: a constant transform $^{P}\mathbf{H}_{\mathrm{pre}(i)}$ that locates the *predecessor* joint frame from the parent link, and the time-varying joint transform $^{\mathrm{pre}(i)}\mathbf{H}_{\mathrm{suc}(i)}(s)$ that locates the *successor* frame of the joint depending on the joint configuration s. The entire parent-to-child transform is defined by positioning the frame of the child link C over the successor frame $^{\mathrm{suc}(i)}\mathbf{H}_{C} = \mathbf{I}_{3}$:

$$\begin{aligned} ^{P}\mathbf{H}_{C}(s) &= {}^{P}\mathbf{H}_{\mathrm{pre}(i)}\, {}^{\mathrm{pre}(i)}\mathbf{H}_{\mathrm{suc}(i)}(s)\, {}^{\mathrm{suc}(i)}\mathbf{H}_{C} \\ &= {}^{P}\mathbf{H}_{\mathrm{pre}(i)}\, {}^{\mathrm{pre}(i)}\mathbf{H}_{C}(s). \end{aligned} \tag{2.19}$$

In this thesis, we will refer to $^{P}\mathbf{H}_{\mathrm{pre}(i)}$ as *tree transform* of joint i, $^{\mathrm{pre}(i)}\mathbf{H}_{C}(s)$ as *joint transform*, and s as *joint generalised position* or just *joint position*.

Definition 2.7.1 (Joint axis of revolute and prismatic joints)**.** Revolute and prismatic joints can be defined by introducing a *joint axis* $\mathbf{a} \in \mathbb{R}^3$, whose coordinates are expressed in the predecessor frame. The joint position $s \in \mathbb{R}$ induces a transform corresponding to an angle-axis decomposition around $\mathbf{a}$ for revolute joints, and a translation along $\mathbf{a}$ for prismatic joints. □

The relative velocities between links P and C can be obtained by differentiating over time Equation 2.19:

$$\frac{\mathrm{d}\,{}^{P}\mathbf{H}_{C}(s)}{\mathrm{d}t} = \frac{\mathrm{d}\,{}^{P}\mathbf{H}_{C}(s)}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}t} = \frac{\mathrm{d}\,{}^{P}\mathbf{H}_{C}(s)}{\mathrm{d}s}\dot{s},$$

where $\dot{s} \in \mathbb{R}$ is the *joint velocity*. Also in this case, it will be convenient to express the relative velocity as a 6D velocity. If X is a placeholder that selects any of the velocity representations introduced in Section 2.3, we express the relative velocity between links P and C as follows:

$${}^{X}\mathbf{v}_{P,C} = {}^{X}\mathbf{S}_{P,C}(s)\dot{s}, \tag{2.20}$$

where we introduced the *joint motion subspace vector* ${}^{X}\mathbf{S}_{P,C}(s) \in \mathbb{R}^6$. In different velocity representations, it is defined as:

$$\begin{aligned}
{}^{C}\mathbf{S}_{P,C}(s) &= \left[{}^{C}\mathbf{H}_{P}(s)\frac{\mathrm{d}\,{}^{P}\mathbf{H}_{C}(s)}{\mathrm{d}s}\right]^{\vee}, \\
{}^{P}\mathbf{S}_{P,C}(s) &= \left[\frac{\mathrm{d}\,{}^{P}\mathbf{H}_{C}(s)}{\mathrm{d}s}\,{}^{C}\mathbf{H}_{P}(s)\right]^{\vee}, \\
{}^{C[P]}\mathbf{S}_{P,C}(s) &= \begin{bmatrix} \frac{\mathrm{d}\,{}^{P}\mathbf{o}_{C}(s)}{\mathrm{d}s} \\ \left(\frac{\mathrm{d}\,{}^{P}\mathbf{R}_{C(s)}}{\mathrm{d}s}\,{}^{C}\mathbf{R}_{P}(s)\right)^{\vee} \end{bmatrix}^{\vee}.
\end{aligned} \tag{2.21}$$

Assumption 2.7.2. In this thesis, we assume that motion subspaces are independent from the joint configuration, i.e. $\frac{\mathrm{d}\,{}^{X}\mathbf{S}_{P,C}}{\mathrm{d}s} = \mathbf{0}_6$, from which ${}^{X}\mathbf{S}_{P,C}(s) = {}^{X}\mathbf{S}_{P,C}$ follows. □

Remark 2.7.1. From the Equations (2.21), it can be shown that the motion subspaces of revolute and prismatic joints, reported in Table 2.1, are independent of the velocity representation. For this reason, in the continuation of this thesis, we drop all its superscripts and subscripts, and denote the motion subspace as just $\mathbf{S}$.

Table 2.1: List of motion subspaces for the supported 1 DoF joints.

Joint type	Motion subspace
Revolute	$\mathbf{S} = \begin{bmatrix} \mathbf{0}_3 \\ \mathbf{a} \end{bmatrix}$
Prismatic	$\mathbf{S} = \begin{bmatrix} \mathbf{a} \\ \mathbf{0}_3 \end{bmatrix}$

The relative acceleration between links P and C can be obtained by differentiating Equation (2.20). Considering Assumption 2.7.2 and Remark 2.7.1, it can be shown that the following resulting relation holds:

$$ {}^{X}\dot{\mathbf{v}}_{P,C} = \frac{\mathrm{d}\mathbf{S}}{\mathrm{d}t}\dot{s} + \mathbf{S}\ddot{s} = \mathbf{S}\ddot{s}, \tag{2.22} $$

where $\ddot{s} \in \mathbb{R}$ is the joint acceleration.

Kinematics and dynamics propagation

Many common RBDAs need to propagate quantities in both directions of a kinematic tree representing a multibody system: child-to-parent and parent-to-child.

Let's consider a pair of links (P, C), connected together with a 1-DoF joint characterised by a position, velocity, and acceleration $s, \dot{s}, \ddot{s} \in \mathbb{R}$. If link P is the joint's parent, and link C its child, we can compute their relative pose ${}^{C}\mathbf{H}_{P}$ with Equation (2.19). From Equation (2.6), we can also compute the related coordinate transformation ${}^{C}\mathbf{X}_{P}$ for 6D velocities.

Definition 2.7.2 (Propagation of 6D Velocities)**.** Given the 6D velocity of the parent link P, and the joint velocity, we want to calculate the velocity of the child link C. In C coordinates, if ${}^{P}\mathbf{v}_{W,P}$ is the body-fixed velocity of the parent link, and ${}^{C}\mathbf{v}_{P,C}$ is the 6D velocity induced by the joint motion, we can write the following relation:

$$ {}^{C}\mathbf{v}_{W,C} = {}^{C}\mathbf{v}_{W,P} + {}^{C}\mathbf{v}_{P,C} = {}^{C}\mathbf{X}_{P}\,{}^{P}\mathbf{v}_{W,P} + \mathbf{S}\dot{s}, $$

where we used Equation (2.20) and Remark 2.7.1 to express the joint component using its velocity. □

Definition 2.7.3 (Propagation of 6D Accelerations)**.** Given the 6D acceleration of the parent link P, and the joint acceleration, we want to calculate the acceleration of the child link C. In C coordinates, if ${}^{P}\dot{\mathbf{v}}_{W,P}$ is the body-fixed apparent acceleration of the parent link, and ${}^{C}\dot{\mathbf{v}}_{P,C}$ is the 6D apparent acceleration induced by the joint motion, we can write the following relation for the intrinsic acceleration:

$$
{}^{C}\mathbf{a}_{W,C} = {}^{C}\dot{\mathbf{v}}_{W,C} = {}^{C}\dot{\mathbf{v}}_{W,P} + {}^{C}\dot{\mathbf{v}}_{P,C} = {}^{C}\dot{\mathbf{v}}_{W,P} + \mathbf{S}\ddot{s},
$$

where we used Equation (2.22) to express the joint component using its acceleration. In this case, we also want to expand the right-hand side to have ${}^{P}\mathbf{a}_{W,P}$, so that the propagation of the accelerations can be performed iteratively. Expanding ${}^{C}\dot{\mathbf{v}}_{W,P}$ using Equation (2.9), and exploiting properties of the cross-product in $\mathbb{R}^6$, we obtain the following expression:

$$
\begin{aligned}
{}^{C}\mathbf{a}_{W,C} &= {}^{C}\dot{\mathbf{v}}_{W,P} + \mathbf{S}\ddot{s} \\
&= {}^{C}\mathbf{X}_{P}\left({}^{P}\dot{\mathbf{v}}_{W,P} + {}^{P}\mathbf{v}_{C,P}\times {}^{P}\mathbf{v}_{W,P}\right) + \mathbf{S}\ddot{s} \\
&= {}^{C}\mathbf{X}_{P}\,{}^{P}\dot{\mathbf{v}}_{W,P} + {}^{C}\mathbf{X}_{P}\,{}^{P}\mathbf{v}_{C,P}\times {}^{P}\mathbf{v}_{W,P} + \mathbf{S}\ddot{s} \\
&= {}^{C}\mathbf{X}_{P}\,{}^{P}\mathbf{a}_{W,P} + {}^{C}\mathbf{X}_{P}\,{}^{P}\mathbf{v}_{W,P}\times {}^{P}\mathbf{v}_{P,C} + \mathbf{S}\ddot{s} \\
&= {}^{C}\mathbf{X}_{P}\,{}^{P}\mathbf{a}_{W,P} + {}^{C}\mathbf{X}_{P}\,{}^{P}\mathbf{v}_{W,P}\times \mathbf{S}\dot{s} + \mathbf{S}\ddot{s},
\end{aligned}
$$

where we used the relation ${}^{P}\dot{\mathbf{v}}_{W,P} = {}^{P}\mathbf{a}_{W,P}$, and Equation (2.22) to express ${}^{P}\mathbf{v}_{P,C} = \mathbf{S}\dot{s}$. In this form, we can propagate the acceleration from parent to child having only the knowledge of the parent 6D velocity and acceleration expressed in its own frame, and the joint quantities. □

Definition 2.7.4 (Propagation of 6D forces)**.** As illustrated in Figure 2.2, given a link L, we want to compute the effects of the propagation of exchanged 6D forces on its dynamics. In this case, we consider a single parent link P, and multiple child links $C_1, C_2, \ldots$. The parent link P applies a 6D force $\mathbf{f}_{J_P}$ to L through the joint connecting them. Similarly, each joint connecting L to its child links C_j receives a 6D force $\mathbf{f}_{J_{C_j}}$ from link L. Furthermore, to model the most generic setting, we assume that an external force $\mathbf{f}_E$ coming from the environment is applied to link L.

In this setting, the dynamics of the link L expressed with the Newton-Euler

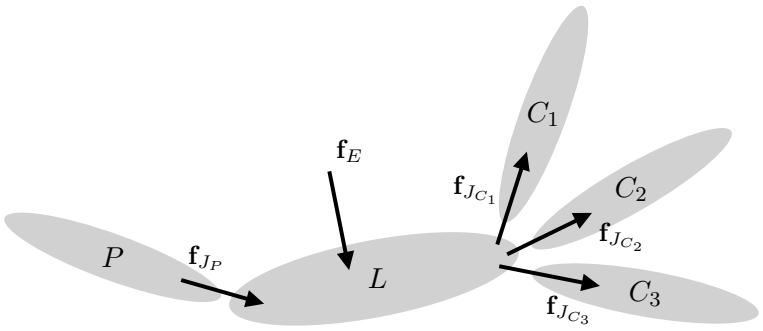


Figure 2.2: Illustration of 6D force propagation through a set of links connected by joints. The link L receives a force $\mathbf{f}_{J_P}$ from its parent P transmitted through the joint J_P, and an external force $\mathbf{f}_E$. The link L transmits to each children C_i a force $\mathbf{f}_{J_{C_i}}$ through their connecting joint J_{C_i}.

equation (2.17b) in its own frame, and ignoring the gravitational effects, is the following:

$$\begin{cases} {}_L\mathbf{f}^{ext} = {}_L\mathbf{f}_{J_P} + {}_L\mathbf{f}_E - \sum_{C_j} {}_L\mathbf{f}_{J_{C_j}} \\ {}_L\mathbf{f}_{J_P} = \mathbb{M}_L \, {}^L\dot{\mathbf{v}}_{W,L} + {}^L\mathbf{v}_{W,L} \bar{\times}^* \mathbb{M}_L \, {}^L\mathbf{v}_{W,L} + \sum_{C_j} {}_L\mathbf{f}_{J_{C_J}} - {}_L\mathbf{f}_E \end{cases}.$$

In contrast to the propagation of velocity and accelerations, in this case we will use this relation to propagate forces from the children (and the environment) to the parent. As we will see, gravitational effects can be considered by propagating through the kinematic tree of a multibody system an equivalent acceleration applied on the base link. □

2.8 FREE-FLOATING MECHANICAL SYSTEMS

In the previous sections, we introduced how individual links and joints can be described, and presented their properties. In this section, we present the mathematical description of a free-floating mechanical system composed of a set of links connected by a set of joints. The free-floating terminology means that no link of the system is rigidly attached to the world frame. Also in this case, we present the system modelling based on the free-floating equations that originate from the Lagrangian formalism as proposed by Traversaro [2017, Chapter 3].

2.8.1 Topology

Definition 2.8.1 (Multibody System)**.** A *multibody system* composed of n_L rigid bodies (also called links) interconnected with n_J joints, can be represented by a *undirected graph*. The links, grouped in the set $\mathcal{L}$, form the graph nodes, while the joints, grouped in the set $\mathcal{J}$, form its edges. □

Definition 2.8.2 (Kinematic graph)**.** The undirected graph with n_L nodes and n_J edges representing a multibody system will also be referred to as *kinematic graph*. □

Definition 2.8.3 (Path)**.** The path $\pi_B(E) = \{B, \dots, E\}$ between link B and link E is the ordered sequence of links part of the kinematic graph that connects B to E. □

Definition 2.8.4 (Base link)**.** We select one of the links part of $\mathcal{L}$ and call it *base link* B. The base link is the root of the kinematic graph. □

Assumption 2.8.1 (Link frames)**.** Each link belonging to $\mathcal{L}$ is associated with a frame rigidly attached to it, called *link frame*. □

Assumption 2.8.2 (Acyclic graph)**.** We assume the kinematic graph to be acyclic, i.e. considering any pair of links $C, D \in \mathcal{L}$, their connecting path $\pi_C(D)$ is unique. □

Definition 2.8.5 (Parent Link)**.** For each link $L \in \mathcal{L}$, if B is the base link, the parent function $\lambda_B : \{\mathcal{L}/B\} \mapsto \mathcal{L}$ maps each link to its parent, with the exclusion of the base link since it is the graph's root. In contexts where B is clearly specified, we omit the subscript. □

Definition 2.8.6 (Link index)**.** For each $L \in \mathcal{L}$, the index function $\mathrm{idx} : \mathcal{L} \mapsto 0 \cup \mathbb{N}$ returns its index. If B is the base link, we assign indices such that $\mathrm{idx}(B) = 0$ and, for the remaining $L \in \{\mathcal{L}/B\}$, we enforce $\mathrm{idx}(L) > \mathrm{idx}(\lambda(L))$. Therefore, links are numbered from 0 to $n_L - 1$. □

Definition 2.8.7 (Joint index)**.** For each $L \in \{\mathcal{L}/B\}$, we assign to the joint $J \in \mathcal{J}$ connecting the link pair $(\lambda(L), L)$ the index $\mathrm{idx}(L)$, that is the index of its child link. Therefore, joints are numbered starting from 1 to n_J. □

2.8.2 Generalised position and velocity

The configuration of a free-floating mechanical system can be modelled as the set formed by the poses of all links. However, the existence of the joints that induce motion constraints enables to determine the system configuration as a pair composed of the pose of a *base link* and the generalised joints positions. These two modelling choices are known, respectively, as *maximal coordinates* and *reduced coordinates*. In this thesis, we focus on the case of reduced coordinates, since it enables the application of efficient iterative algorithms [Featherstone, 2008] to operate on the system's kinematics and dynamics. Furthermore, interesting properties of the mathematical model that can be computed in reduced coordinates can be exploited for designing control systems.

In reduced coordinates, we can formalise the generalised position and the generalised velocity of the floating-base multibody system as follows:

$$\begin{cases} \mathbf{q} = \left({}^{W}\mathbf{H}_{B}, \mathbf{s}\right) \in \mathcal{Q} = \mathrm{SE}(3) \times \mathbb{R}^{n} \\ \dot{\mathbf{q}} = \left({}^{W}\dot{\mathbf{H}}_{B}, \dot{\mathbf{s}}\right) \in \mathcal{V} \end{cases} \tag{2.23}$$

where we introduced the *joint positions* $\mathbf{s} \in \mathbb{R}^n$, also called *shape*. Under the assumption of having only 1-DoF joints, n is the overall number of internal Degrees of Freedom of the system, matching the number of joints n_J. Note that this limitation can be removed. A more general formulation can be found in [Featherstone, 2008].

Similarly to what we observed for a single rigid body, it can be more convenient to represent the system's velocity as a column vector:

$${}^{X}\boldsymbol{\nu} = \begin{bmatrix} {}^{X}\mathbf{v}_{W,B} \\ \dot{\mathbf{s}} \end{bmatrix} \in \mathbb{R}^{6+n}, \tag{2.24}$$

where ${}^{X}\mathbf{v}_{W,B}$ is the velocity of the base link, $\dot{\mathbf{s}} \in \mathbb{R}^n$ are the *joint velocities*, and the generic frame X is a placeholder to select one among the *body-fixed* $X = B$, *inertial-fixed* $X = W$, or *mixed* $X = B[W]$ representations.

2.8.3 Kinematics

In this section, we describe how we can relate the pose ${}^{W}\mathbf{H}_{E}$ and the velocity $\mathbf{v}_{W,E}$ between the world frame W and a generic link E of the multibody mechanical structure with the generalised position $\mathbf{q}$ and generalised velocity $\boldsymbol{\nu}$ of the system. The link E can be thought of as the *end-effector* frame, even if it applies to any generic link $L \in \mathcal{L}$ of the model and, more generally, any frame rigidly attached to any link.

Link pose

The pose of a link E with respect to the world frame uniquely depends on the generalised position $\mathbf{q}$. We can denote the pose as a function ${}^{W}\mathbf{H}_{E}(\mathbf{q}) : \mathcal{Q} \mapsto \mathrm{SE}(3)$, defined as follows:

$$
{}^{W}\mathbf{H}_{E}(\mathbf{q}) = {}^{W}\mathbf{H}_{B}\,{}^{B}\mathbf{H}_{E}(\mathbf{s}) = \begin{bmatrix} {}^{W}\mathbf{R}_{B} & {}^{W}\mathbf{o}_{B} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} {}^{B}\mathbf{H}_{E}(\mathbf{s}). \tag{2.25}
$$

The transform ${}^{B}\mathbf{H}_{E}(\mathbf{s})$ defines the *relative forward kinematics* between link B and link E, and it depends on the sequence of parent-to-child transforms ${}^{\lambda(i)}\mathbf{H}_{i}$ of all the adjacent links belonging to the path $\pi_{B}(E)$:

$$
\begin{aligned}
{}^{B}\mathbf{H}_{E}(\mathbf{s}) &= {}^{B}\mathbf{H}_{\lambda(\lambda\cdots(E))} \cdots {}^{\lambda(\lambda(E))}\mathbf{H}_{\lambda(E)}\,{}^{\lambda(E)}\mathbf{H}_{E} \\
&= \prod_{L_i \in \{\pi_{B}(E)/B\}} {}^{\lambda(L_i)}\mathbf{H}_{L_i}(s_i).
\end{aligned}
$$

Each entry ${}^{\lambda(L)}\mathbf{H}_{L} = {}^{\lambda(L)}\mathbf{H}_{L}(s)$ of the product is given by the joint model of Equation (2.19) that defines the transform between two links connected by a joint.

Link velocity

For what concerns the velocity between the world frame W and link E, we want to find an expression in the following generic form:

$$
{}^{Y}\mathbf{v}_{W,E} = {}^{Y}J_{W,E/X}(\mathbf{q})\,{}^{X}\boldsymbol{\nu},
$$

where we introduced $J_{W,E} \in \mathbb{R}^{6\times(6+n)}$ as the *floating-base Jacobian* of link E. We can notice that two different velocity representations characterise this relation, denoted by the X and Y placeholder frames: X is related to the input system velocity ${}^{X}\boldsymbol{\nu}$, and Y is related to the output link velocity ${}^{Y}\mathbf{v}_{W,E}$. We will show the derivation of the *left-trivialized Jacobian* ${}^{Y}J_{W,Y/X}$, and then introduce the appropriate transformations to change the representations of X and Y.

The velocity of link E w.r.t. the world frame can be computed differentiating Equation (2.25). Instead of proceeding with this calculation, we follow the equivalent approach of decomposing the velocity $\mathbf{v}_{W,E}$ as the sum of the base velocity and the velocity between the base B and the link E:

$$
{}^{E}\mathbf{v}_{W,E} = {}^{E}\mathbf{v}_{W,B} + {}^{E}\mathbf{v}_{B,E}.
$$

We can express $\mathbf{v}_{B,E}$ as the sum of the velocities between adjacent links in the link path $\pi_B(E)$ between link B and E:

$$
\begin{aligned}
{}^{E}\mathbf{v}_{W,E} &= {}^{E}\mathbf{v}_{W,B} + \sum_{L_i \in \{\pi_B(E)/B\}} {}^{E}\mathbf{v}_{\lambda(L_i),L_i} \\
&= {}^{E}\mathbf{v}_{W,B} + \sum_{L_i \in \{\pi_B(E)/B\}} {}^{E}\mathbf{X}_{L_i}\, {}^{L_i}\mathbf{v}_{\lambda(L_i),L_i} \\
&= {}^{E}\mathbf{v}_{W,B} + \sum_{L_i \in \{\pi_B(E)/B\}} {}^{E}\mathbf{X}_{L_i}\, {}^{L_i}\mathbf{S}_{\lambda(L_i),L_i}(s_i)\dot{s}_i,
\end{aligned}
$$

where we used the expression of the relative velocity between two adjacent links ${}^{L}\mathbf{v}_{\lambda(L),L}$ introduced in Equation (2.20). Expressing the obtained relation in matrix form, we reach the expression of the desired left-trivialized Jacobian, where X is a placeholder that depends on the representation of the system's velocity:

$$
{}^{E}\mathbf{v}_{W,E} = \begin{bmatrix} {}^{E}\mathbf{X}_{X} & {}^{E}S_{B,E}(\mathbf{s}) \end{bmatrix} \begin{bmatrix} {}^{X}\mathbf{v}_{W,B} \\ \dot{\mathbf{s}} \end{bmatrix} = {}^{E}J_{W,E/X}\, {}^{X}\boldsymbol{\nu}. \tag{2.26}
$$

We introduced the matrix $S_{B,E}(\mathbf{s}) \in \mathbb{R}^{6\times n}$ for the joint part, where its i-th column is defined as:

$$ {}^{E}S_{B,E}^{(:,i)}(s) = \begin{cases} {}^{E}\mathbf{X}_{L}\, {}^{L}\mathbf{S}_{\lambda(L),L}(s) & \text{if } L \in \{\pi_B(E)/B\}, \\ \mathbf{0}_6 & \text{otherwise.} \end{cases} $$

Definition 2.8.8 (Link Jacobian)**.** The generic form of the floating-base Jacobian of link E is, therefore:

$$ {}^{Y}J_{W,E/X}(\mathbf{q}) = \begin{bmatrix} {}^{Y}\mathbf{X}_{X} & {}^{Y}S_{B,E}(\mathbf{s}) \end{bmatrix}. \tag{2.27} $$

□

Finally, the input and output representations can be changed from the pair (X, Y) to the pair (D, F) by either left or right multiplication:

$$ {}^{D}J_{W,E/F}(\mathbf{q}) = {}^{D}\mathbf{X}_{Y}\, {}^{Y}J_{W,E/X}(\mathbf{q})\, \mathrm{diag}\left({}^{X}\mathbf{X}_{F}, \mathbf{I}_n\right). \tag{2.28} $$

Remark 2.8.1 (Floating-base Jacobian and 6D forces)**.** The duality between 6D velocities and 6D forces also propagates to the definition of the floating-base Jacobian. If we consider a 6D force ${}_{C_i}\mathbf{f}_i$, that could be for example the force applied to the frame $C_i = (\mathbf{o}_{C_i}, [W])$ associated to the contact point i of link L which pose is defined by a transform ${}^{L}\mathbf{H}_{C_i}$, we can use Equation (2.27) to compute its projection to the floating-base configuration space as ${}^{C_i}J^{\top}_{W,L/X}(\mathbf{q})\, {}_{C_i}\mathbf{f}_i = ({}^{C_i}\mathbf{X}_{L}\, {}^{L}J_{W,L/X}(\mathbf{q}))^{\top}\, {}_{C_i}\mathbf{f}_i \in \mathbb{R}^{6+n}$. This verbose notation results particularly useful in this case, because in the example of a rolling contact, the frame C_i is time-varying, and the most appropriate contact Jacobian is ${}^{C_i}J_{W,L}$ instead of ${}^{C_i}J_{W,C_i}$, regardless of the system's velocity representation X.

2.8.4 Dynamics

The EoMs of the free-floating mechanical system, similar to what we showed for a single rigid body in Section 2.6.2, can be derived from Lagrangian mechanics. Also in this case, we utilise the left-trivialized Lagrangian taking as input the pair $({}^{W}\mathbf{H}_{B}, {}^{B}\boldsymbol{\nu})$, and drop the B superscript by assuming that all quantities are derived in body-fixed representation. In the simplified setting of maximal coordinates, the Lagrangian of the overall system can be obtained as the combination of the left-trivialized Lagrangian of all its links:

$$
\begin{aligned}
\ell(\mathbf{q}, \boldsymbol{\nu}) &= k(\mathbf{q}, \boldsymbol{\nu}) - U(\mathbf{q}), \\
\kappa(\mathbf{q}, \boldsymbol{\nu}) &= \frac{1}{2} \sum_{L \in \mathcal{L}} {}^{L}\mathbf{v}_{W,L}^{\top} \, \mathbb{M}_{L} \, {}^{L}\mathbf{v}_{W,L}, \\
U(\mathbf{q}) &= - \sum_{L \in \mathcal{L}} \begin{bmatrix} {}^{W}\mathbf{g}^{\top} & \mathbf{0}_{1\times 3} \end{bmatrix} m_{L} \, {}^{W}\mathbf{H}_{L} \begin{bmatrix} {}^{L}\mathbf{c} \\ 1 \end{bmatrix}.
\end{aligned}
$$

In reduced coordinates, we can obtain an alternative and more compact expression of the energies:

$$
\begin{aligned}
\kappa(\mathbf{q}, \boldsymbol{\nu}) &= \frac{1}{2} \, \boldsymbol{\nu}^{\top} M(\mathbf{q}) \, \boldsymbol{\nu}, \\
U(\mathbf{q}) &= - \begin{bmatrix} \mathbf{g}^{\top} & \mathbf{0}_{1\times 3} \end{bmatrix} m \, {}^{W}\mathbf{H}_{B} \begin{bmatrix} {}^{B}\mathbf{c}(\mathbf{s}) \\ 1 \end{bmatrix},
\end{aligned}
$$

where $M(\mathbf{q}) \in \mathbb{R}^{(6+n)\times(6+n)}$ is the system's *mass matrix*, defined as:

$$
M(\mathbf{q}) = \sum_{L \in \mathcal{L}} J_{L}^{\top}(\mathbf{q}) \, \mathbb{M}_{L} J_{L}(\mathbf{q}), \tag{2.29}
$$

$m \in \mathbb{R}$ is the total mass of the mechanical system:

$$
m = \sum_{L \in \mathcal{L}} m_{L},
$$

and ${}^{B}\mathbf{c}(\mathbf{s}) \in \mathbb{R}^{3}$ is its CoM expressed in the coordinates of the base frame B:

$$
\begin{bmatrix} {}^{B}\mathbf{c}(\mathbf{s}) \\ 1 \end{bmatrix} = \frac{1}{m} \sum_{L \in \mathcal{L}} m_{L} \, {}^{B}\mathbf{H}_{L} \begin{bmatrix} {}^{L}\mathbf{c}_{L} \\ 1 \end{bmatrix}.
$$

We introduced $J_L(\mathbf{q}) = {}^L J_{W,L/B}$ as the floating-base left-trivialized Jacobian of link L for left-trivialized generalised velocities ${}^B\boldsymbol{\nu}$, as defined in Equation (2.26).

The Equations of Motion of the multibody system, considering that its configuration $\mathbf{q}$ is an element of $\mathrm{SE}(3) \times \mathbb{R}^n$, can be obtained by applying the Hamel equations [Marsden, Jerrold E. et al., 2013; Maruskin, 2018], that can be seen as the combination of the Euler-Poincarè equation for the base variables in $\mathrm{SE}(3)$ and the classical Euler-Lagrange equation for the joint variables in $\mathbb{R}^n$. The left-trivialized Lagrangian plugged into the Hamel equations gives the EoMs of the multibody system [Traversaro, 2017, Appendix A.4]:

$$\begin{cases} \dot{\mathbf{q}} = \left({}^W\dot{\mathbf{H}}_B, \dot{\mathbf{s}} \right) \\ M(\mathbf{q})\,\dot{\boldsymbol{\nu}} + C(\mathbf{q}, \boldsymbol{\nu})\,\boldsymbol{\nu} + g(\mathbf{q}) = B\boldsymbol{\tau} + \sum_{L \in \mathcal{L}} J_L^\top(\mathbf{q})\,\mathbf{f}_L^{ext} \end{cases} \tag{2.30}$$

where we used the mass matrix $M(\mathbf{q})$ defined in Equation (2.29), the actuation selector $B = (\mathbf{0}_{6\times n}; \mathbf{I}_n) \in \mathbb{R}^{(6+n)\times n}$, and:

$$C(\mathbf{q}, \boldsymbol{\nu}) = \sum_{L \in \mathcal{L}} J_L^\top \left[(\mathbf{v}_L \bar{\times}^* \mathbb{M}_L + \mathbb{M}_L\, \mathbf{v}_L \times) J_L + \mathbb{M}_L \dot{J}_L \right],$$

$$g(\mathbf{q}) = -M(\mathbf{q}) \begin{bmatrix} {}^W\mathbf{R}_B^\top\, {}^W\mathbf{g} \\ \mathbf{0}_{3\times 1} \\ \mathbf{0}_{n\times 1} \end{bmatrix}.$$

When the decomposition of gravity and Coriolis effects is not important, we will use the compact form of the EoM:

$$M(\mathbf{q})\,\dot{\boldsymbol{\nu}} + h(\mathbf{q}, \boldsymbol{\nu}) = B\boldsymbol{\tau} + \sum_{L \in \mathcal{L}} J_L^\top(\mathbf{q})\,\mathbf{f}_L^{ext}, \tag{2.31}$$

where we introduced the vector of *bias forces* $h(\mathbf{q}, \boldsymbol{\nu}) \in \mathbb{R}^{6+n}$.

Remark 2.8.2. We introduced for each link an *external force* $\mathbf{f}_L^{ext}$ expressed in link frame L. If multiple forces are applied to different locations of the link where a local frame C_i is positioned, they can either be projected individually to the configuration space as described in Remark 2.8.1, or summed all together to a single 6D force $\mathbf{f}_L^{ext} = \sum_i {}_L\mathbf{X}^{C_i}\, {}_{C_i}\mathbf{f}_i$ expressed in L and projected as done in Equation (2.30). □

Remark 2.8.3 (Structure of the mass matrix)**.** The mass matrix of Equation (2.30), considering the usage of the body-fixed representation for velocities and inertias, should be denoted as $M(\mathbf{q}) = M_B(\mathbf{s})$. In fact, this is the only representation in which the mass matrix depends only on the shape and not on the base pose. In this representation, the mass matrix can be factorised as follows:

$$M_B(\mathbf{s}) = \begin{bmatrix} {}_B\mathbb{M}(\mathbf{s}) & F(\mathbf{s}) \\ F^\top(\mathbf{s}) & H(\mathbf{s}) \end{bmatrix},$$

where $F(\mathbf{s}) \in \mathbb{R}^{6\times n}$, $H(\mathbf{s}) \in \mathbb{R}^{n\times n}$ is the *joint mass matrix*, and ${}_B\mathbb{M}(\mathbf{s}) \in \mathbb{R}^{6\times 6}$ is the *locked 6D rigid body inertia* of the multibody system. □

2.8.5 Change of base variables

In the previous section, we derived the EoMs of a multibody system assuming the base link B being the root of the kinematic graph, and using the body-fixed representation for all velocities and forces. As shown by Traversaro [2017, Section 3.6], it's possible to apply a change of variables to the EoMs of Equation (2.30) and express the dynamics either in a different velocity representation or with a different base link.

Starting from a multibody system having the pair $(\mathbf{q}, \boldsymbol{\nu})$ as generalised position and velocity, we want to find a change of variables such that the new pair is $(\tilde{\mathbf{q}}, \tilde{\boldsymbol{\nu}})$, defined as follows:

$$\begin{aligned} \tilde{\mathbf{q}} &= (\tilde{\mathbf{H}}, \mathbf{s}), \\ \tilde{\mathbf{H}} &= \mathbf{H}_T(\mathbf{s})\, \mathbf{H} \in \mathrm{SE}(3), \\ \tilde{\boldsymbol{\nu}} &= T(\mathbf{q})\, \boldsymbol{\nu}, \\ T(\mathbf{q}) &= \begin{bmatrix} T_{bb}(\mathbf{q}) & T_{bs}(\mathbf{q}) \\ \mathbf{0}_{n\times 6} & \mathbf{1}_n \end{bmatrix} \in \mathbb{R}^{(6+n)\times(6+n)}. \end{aligned}$$

We introduced the linear transformation $T : \mathrm{SE}(3) \times \mathbb{R}^n \mapsto \mathbb{R}^{(6+n)\times(6+n)}$ assuming that it is a smooth function and $\forall \mathbf{q} \in \mathrm{SE}(3) \times \mathbb{R}^n$, $\det[T(\mathbf{q})] \neq 0$. Note that T does not alter the joint equations of the dynamics.

It can be shown that the EoMs (2.30) are transformed as follows:

$$\begin{cases} \dot{\tilde{\mathbf{q}}} = \left(\dot{\mathbf{H}}(\mathbf{q}, \boldsymbol{\nu})\, \mathbf{H}_T(\mathbf{s}) + \mathbf{H}(\mathbf{q})\dot{\mathbf{H}}_T(\mathbf{s}, \dot{\mathbf{s}}), \dot{\mathbf{s}}\right) \\ \tilde{M}(\tilde{\mathbf{q}})\dot{\tilde{\boldsymbol{\nu}}} + \tilde{C}(\tilde{\mathbf{q}}, \tilde{\boldsymbol{\nu}})\tilde{\boldsymbol{\nu}} + \tilde{g}(\tilde{\mathbf{q}}) = B\boldsymbol{\tau} + \sum_{\mathcal{L}} \tilde{J}_L(\mathbf{q})\, {}_L\mathbf{f}_L^{ext} \end{cases},$$

where we have introduced the transformed quantities:

$$\begin{aligned} \tilde{M} &= T^{-\top} M T^{-1}, \\ \tilde{C} &= T^{-\top}\left(M\frac{\mathrm{d}}{\mathrm{d}t}(T^{-1}) + CT^{-1}\right), \\ \tilde{g} &= T^{-\top} g, \\ \tilde{J}_L &= J_L T, \end{aligned} \tag{2.32}$$

and omitted their dependencies to improve readability.

Remark 2.8.4. For what regards the link Jacobian $J_L(\mathbf{q}) = {}^L J_{W,L/B}(\mathbf{q})$, beyond depending on the transformation $T(\mathbf{q})$ related to the change of variables, it also depends on the reference frame in which the 6D forces $\mathbf{f}_L^{ext}$ are expressed. Until now, we always expressed the link forces in the link frame, i.e. ${}_L\mathbf{f}_L^{ext}$. However, if link forces are expressed in a different frame, we can use Equation (2.28) to modify the output representation of the Jacobian.

In this thesis, we only need the definition of the EoMs in different velocity representations. It can be shown that the transform $\mathbf{H}_T$ is used only if the base link changes from B to any other frame belonging to the multibody system. For this reason, starting from the obtained left-trivialized EoMs, we will present the following change of variables assuming the base link always being B, and therefore $\mathbf{H}_T = \mathbf{I}_4$. The interested reader could refer to [Traversaro, 2017] for the transform necessary to change the base link.

Left-trivialized

The left-trivialized EoMs (2.30) of the multibody system considering B as base link, using a complete and non-ambiguous notation, are the following:

$$M_B(\mathbf{s})\, {}^B\dot{\boldsymbol{\nu}} + C_B(\mathbf{q}, {}^B\boldsymbol{\nu})\, {}^B\boldsymbol{\nu} + g_B(\mathbf{q}) = B\boldsymbol{\tau} + \sum_{L\in\mathcal{L}} {}^L J_{W,L/B}^{\top}(\mathbf{q})\, {}_L\mathbf{f}_L^{ext}$$

where:

$$
\begin{aligned}
M_B(\mathbf{s}) &= \sum_{L\in\mathcal{L}} {}^{L}J_{W,L/B}^{\top}\, {}_{L}\mathbb{M}_{L}\, {}^{L}J_{W,L/B}, \\
C_B(\mathbf{q}, {}^{B}\boldsymbol{\nu}) &= \sum_{L\in\mathcal{L}} {}^{L}J_{W,L/B}^{\top}\Big[({}^{L}\mathbf{v}_{W,L}\bar{\times}^{*}\, {}_{L}\mathbb{M}_{L} + {}_{L}\mathbb{M}_{L}\, {}^{L}\mathbf{v}_{W,L}\times)\, {}^{L}J_{W,L/B} + \\
&\qquad + {}_{L}\mathbb{M}_{L}\, {}^{L}\dot{J}_{W,L/B}\Big], \\
g(\mathbf{q}) &= -M_B(\mathbf{q}) \begin{bmatrix} {}^{W}\mathbf{R}_{B}^{\top}\, {}^{W}\mathbf{g} \\ \mathbf{0}_{3\times 1} \\ \mathbf{0}_{n\times 1} \end{bmatrix}, \\
{}^{L}J_{W,L/B}(\mathbf{q}) &= \begin{bmatrix} {}^{L}\mathbf{X}_{B} & {}^{L}\mathbf{S}_{B,L}(\mathbf{s}) \end{bmatrix}.
\end{aligned}
$$

Right-trivialized

If we change the velocity representation of the base to inertial-fixed, we can apply the change of variables (2.32) using the following transformation matrix:

$$
{}^{W}T_{B} = \begin{bmatrix} {}^{W}\mathbf{X}_{B} & \mathbf{0}_{6\times n} \\ \mathbf{0}_{n\times 6} & \mathbf{I}_{n} \end{bmatrix}
$$

Furthermore, if we also want to use external 6D forces expressed in the world frame ${}_{W}\mathbf{f}_{L}^{ext}$, we can obtain the right-trivialized Jacobian by applying T, and then update its output representation as follows:

$$
{}^{W}J_{W,L/W} = {}^{W}\mathbf{X}_{L}\, {}^{L}J_{W,L/B}\, {}^{B}T_{W}.
$$

Mixed

Similarly, the transformation to obtain the EoMs in mixed representation is the following:

$$
{}^{B[W]}T_{B} = \begin{bmatrix} {}^{B[W]}\mathbf{X}_{B} & \mathbf{0}_{6\times n} \\ \mathbf{0}_{n\times 6} & \mathbf{I}_{n} \end{bmatrix}.
$$

In life, unlike chess, the game continues after checkmate.

— Isaac Asimov

3 BASICS OF REINFORCEMENT LEARNING

In the field of Artificial Intelligence (AI), it has been hypothesised that intelligence can be understood as subserving the maximisation of reward [Silver et al., 2021]. It was suggested that intelligent abilities and behaviours could be attained by agents that learn from trial-and-error by receiving feedback on their performance: the reward. RL is one among the possible *generic* formulations aiming to train agents that solve the problem of maximising the reward.

Reinforcement Learning operates on a unified setting decoupled as two systems interacting sequentially over time, illustrated in Figure 3.1. The *environment* is the world the *agent* interacts with. Unlike other ML domains, the learned RL policy generates actions that may affect not only the current instant (immediate reward), but also the new configuration in which the environment transitions and its corresponding reward. The trial-and-error nature together with delayed rewards, giving rise to the *credit assignment problem*, are two among the essential features that characterise RL [Sutton et al., 2018].

This chapter, based on the theory and notation from Achiam [2018] and Dong et al. [2020], describes in greater detail the Reinforcement Learning setting, introduces the terminology and the mathematical framework used throughout the thesis, and presents algorithms to solve the corresponding reward maximisation problem. Despite being a generic setting from a high-level perspective, all the element parts of Figure 3.1 may have different properties altering the formulation of the specific problem at hand. This thesis focuses on the application of RL to the robotics domain, constraining the nature of environment modelling, the structure of the policies that can be learned, and the family of algorithms that can be employed. In particular, this chapter is aimed at introducing the theory of operating on environments modelled with unknown stochastic continuous dynamics, providing continuous rewards and receiving continuous actions generated by stochastic policies implemented as neural networks. The setting adopting this family of policies is known as Deep Reinforcement Learning (DRL). We will use throughout the thesis the RL and DRL terminology interchangeably.

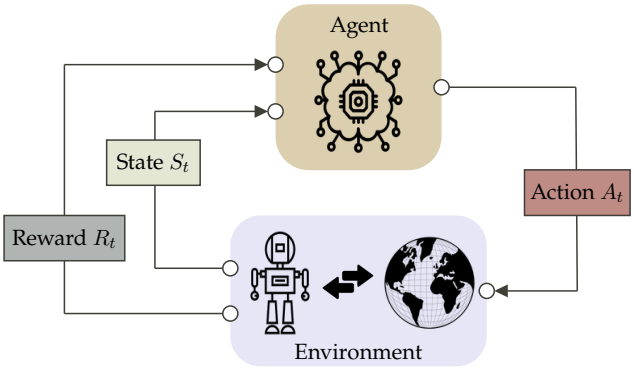


Figure 3.1: The Reinforcement Learning setting.

3.1 KEY CONCEPTS

This section provides the basic terminology and definitions of RL used throughout the following chapters, mainly borrowed from [Sutton et al., 2018; Achiam, 2018; Dong et al., 2020].

3.1.1 Environment

The environment is the – rather abstract – entity of the RL setting that determines the evolution of a system as a consequence of an action, and the generation of the reward signal.

Let's assume we can encode the overall configuration of the environment at time t in a *state* $s_t \in \mathcal{S}$. If $a_t \in \mathcal{A}$ is the action generated by the agent at time t, we define as *state transition* the map $(s_t, a_t) \mapsto s_{t+1}$. We model the state such that the transition to a next state at time t depends only on s_t and the selected action. Therefore, the state's definition is enough to encode all the information about the past evolution of the environment.

State transition maps can assume different forms depending on the nature of the environment. Formally, if the environment is *deterministic*, state transitions can be expressed with a *state-transition function* $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$; instead, if the environment is *stochastic*, state transitions can be expressed with the *state-transition probability density function* $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Pr[\mathcal{S}]$:

$$\begin{aligned} s_{t+1} &= f(s_t, a_t) \qquad \text{if deterministic,} \\ s_{t+1} &\sim \mathcal{P}(\cdot|s_t, a_t) \qquad \text{if stochastic.} \end{aligned} \tag{3.1}$$

The environment is also responsible for generating the *immediate reward* $r_t \in \mathbb{R}$. In its most generic form, the *reward function* can be modelled as a function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

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

Reinforcement Learning, contrary to other approaches that find optimal policies in a comparable setting, assumes that both the state-transition function $\mathcal{P}$ and the reward function $\mathcal{R}$ are unknown. It focuses on methods to learn how to act optimally only by sampling sequences of states-actions-rewards without assuming any knowledge of the process generating their data.

The objective(s) of the learning process are known as *tasks*. In the context of RL, the reward function $\mathcal{R}$ is responsible to guide the agent towards the fulfilment of the considered task(s).

Remark 3.1.1 (Episodic and Continuing tasks)**.** The iterative nature of RL requires the environment to be sampled continuously over different *episodes*. An *episode* is defined as the environment evolution from an initial state s_0 to its termination. We can identify two episode termination strategies, corresponding to the underlying *task* – the objective the reward is maximising. *Episodic tasks* are characterised by a –possibly variable– finite length T, i.e. they do not continue after reaching a terminal state s_T. *Continuing tasks*, instead, are characterised by an infinite horizon and they never terminate. On some occasions, it is useful considering episodic tasks as continuing tasks by assuming that in time steps $t > T$, the last state s_T becomes a special *absorbing state*, that is a state that can only transition to itself regardless of the action generating only rewards of zero. On other occasions, instead, it is useful to truncate the evolution of a continuing task at a certain time step [Pardo et al., 2022]. □

Remark 3.1.2 (States and observations)**.** It's common in the RL literature to use the terms *state* and *observation* interchangeably, depending on the context. However, practically speaking, the agent does not always receive the same state s that encodes the complete environment configuration, whose space $\mathcal{S}$ could also be non-Euclidean. When necessary, we introduce the function $O : \mathcal{S} \to \mathcal{O}$ that computes the observation $o \in \mathcal{O}$ from the state s. Oftentimes, we consider the observation o as the input to the agent, and it is convenient to

consider $\mathcal{O} = \mathbb{R}^n$. To keep a clean notation and reduce confusion, in the rest of this background chapter we just use the notation s. We only differentiate between s and o when they are explicitly different quantities. Furthermore, we always assume *fully observable* problems, meaning that the full state is always available to the agent [Lovejoy, 1991; Jaakkola et al., 1994]. □

3.1.2 Agent

The agent is the – rather abstract – entity of the RL setting that learns and interacts with the environment with the objective of maximising the received reward signal. In our setting, depending on the RL algorithm selected for training, agents are at least composed of a function approximator used to generate the action, i.e. a *policy*, and a method for optimising this function given the observed states and received rewards.

3.1.3 Policy

The policy encodes the strategy followed by the agent in order to select its control actions. Policies can be either *deterministic*, i.e. given a state s, the agent always takes the same action:

$$a_t(s_t) = \mu(s_t) \quad \text{, with } \mu(s_t) : \mathcal{S} \rightarrow \mathcal{A},$$

or *stochastic*, i.e. modelled as a probability distribution, from which the action is sampled:

$$a_t \sim \pi(\cdot|s_t) \quad \text{, with } \pi(\cdot|s_t) : \mathcal{S} \rightarrow \Pr(\mathcal{A}).$$

In most DRL applications, policies assume the form of neural networks, whose parameters $\boldsymbol{\theta}$ are updated during the training procedure by an optimisation algorithm. We often make this parameterization explicit by using subscripts, i.e. $\mu_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}}$. The most common policy chosen in our target setting is described in the following example.

Example 3.1.1 (Diagonal Gaussian Policy)**.** The general form for the Probability Density Function (PDF) of the univariate normal distribution is the following Gaussian function:

$$f_{\mathcal{N}}(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right). \tag{3.2}$$

We denote sampling a single real-valued random variable $x \in \mathbb{R}$ from this distribution as $x \sim \mathcal{N}(\mu, \sigma^2)$, where μ and σ^2 are the *mean* and the *variance* of the distribution, respectively. We can use the same distribution also in the multivariate case, in which $x, \mu \in \mathbb{R}^k$ and $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots) \in \mathbb{R}^{k \times k}$, where we assumed a diagonal covariance matrix parameterized with diagonal values denoted as $\sigma \in \mathbb{R}^k$. A stochastic policy could be implemented as a parameterized multivariate diagonal Gaussian distribution. During the training phase, exploration is achieved by sampling from the distribution. During the evaluation phase, instead, the policy's exploitation is achieved by always selecting the mean of the distribution. We can decide to condition the action a_t on the state s_t by modelling the distribution with a NN parameterized by $\boldsymbol{\theta}$. We define the policy being the corresponding parameterized density, denoted as:

$$\pi_{\boldsymbol{\theta}}(a_t \mid s_t) = f(a_t \mid \boldsymbol{\theta}(s_t)) = f_{\mathcal{N}}(a_t \mid \mu_{\boldsymbol{\theta}}(s_t), \sigma_{\boldsymbol{\theta}}).$$

A common choice is to compute only $\mu_{\boldsymbol{\theta}} = \mu_{\boldsymbol{\theta}}(s_t)$ from the forward pass of the NN, which takes the state s_t as input. The standard deviation is often computed by standalone parameters independent of the NN and consequently is state-independent. Furthermore, to prevent generating negative values of $\sigma_{\boldsymbol{\theta}}$, it's common practice to let these standalone parameters provide $\log \sigma_{\boldsymbol{\theta}}$, so that they could be optimised in $\mathbb{R}$ and, upon exponentiation, return a standard deviation in $\mathbb{R}^+$.

In this setting, actions could be sampled as $a_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid s_t)$. Implementations that exploit AD frameworks modify the sampling strategy using the *reparameterization trick*, allowing to apply backpropagation on stochastic problems. In practice, actions are sampled as $a_t = \mu_{\boldsymbol{\theta}}(s_t) + \sigma_{\boldsymbol{\theta}} \cdot \boldsymbol{z}$, with $\boldsymbol{z} \sim \mathcal{N}(0, 1)$. □

Remark 3.1.3 (Probability Density Function and likelihood)**.** The Gaussian function of Equation (3.2) could be used for two scopes depending on which

of its parameters is considered as fixed. When considered as $x \mapsto f(x \mid \mu, \sigma)$, it defines the PDF of the distribution. Instead, when considered as $(\mu, \sigma) \mapsto f(x \mid \mu, \sigma)$, it defines the *likelihood function* of the distribution. The former calculates the probability of x to fall within a particular range of values considering constant distribution parameters. The latter, instead, also denoted as $\mathcal{L}(\mu, \sigma \mid x)$, describes the probability of the observed data x as a function of varying distribution parameters. As we will analyse later, calculations are often simpler when considering the *log-likelihood* $\boldsymbol{\ell} = \log \mathcal{L}$ instead of the plain likelihood. From a state s, assuming a diagonal multivariate Gaussian policy with mean $\mu = \mu_{\boldsymbol{\theta}}(s)$ and standard deviation $\sigma = \sigma_{\boldsymbol{\theta}}$, it can be shown that the log-likelihood of an action $a \in \mathbb{R}^k$ is the following [Bishop, 2006]:

$$\boldsymbol{\ell}(\mu, \sigma \mid a) = \log \pi_{\boldsymbol{\theta}}(a \mid s) = -\frac{1}{2} k \log 2\pi - \sum_{i=1}^{k} \left(\log \sigma_i + \frac{(a_i - \mu_i)^2}{2\sigma_i^2} \right). \quad (3.3)$$

□

3.1.4 Trajectory

In the RL setting illustrated in Figure 3.1, the sequential interaction between the agent and the environment generates a *trajectory* τ of states and actions:

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T).$$

The first state s_0, i.e. the starting point of the trajectory, is randomly sampled from the *initial state distribution* $\rho_0(\cdot)$ as $s_0 \sim \rho_0(\cdot)$. Afterwards, the state evolves accordingly to one of the state transition maps defined in Equation (3.1), until a terminal state s_T is reached[12]. The process generating trajectory data is illustrated in Figure 3.2.

Remark 3.1.4 (Reward trajectory)**.** The literature is not uniform on the time subscript of the immediate reward $r_{(\cdot)}$. In this thesis, we consider the data-generating process illustrated in Figure 3.2. The transition $t \to t+1$ corresponds

12 This is always true for episodic tasks, not always for the continuing tasks that characterise continuous control. When a continuous control trajectory terminates on a terminal state s_T, there is no difference with an episodic task. However, practically speaking, truncating a continuous control trajectory is common after a given number of steps. It is essential to distinguish these two cases because the propagation of the reward backward in time differs significantly.

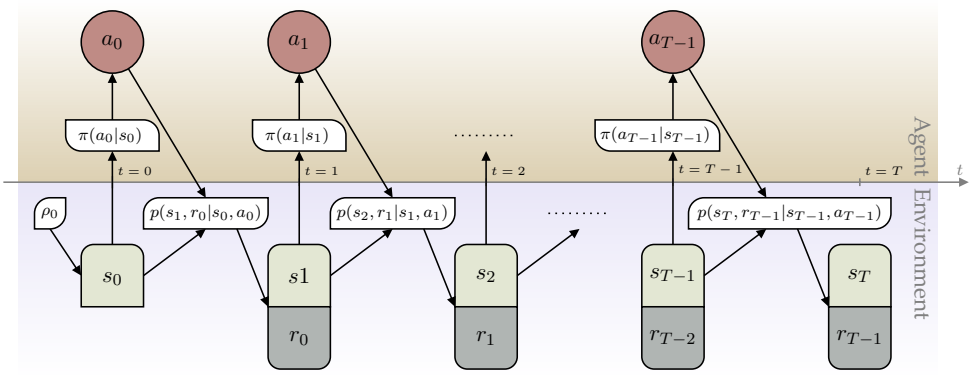


Figure 3.2: Illustration of the process generating trajectory data.

to the data $(s_t, a_t) \mapsto (r_t, s_{t+1})$. Note that the subscript of the reward is associated to the time t of the tuple (s_t, a_t) that generated it. While this might seem odd at a first glance, it keeps the notation consistent when building the dataset $\mathcal{D}$ of a trajectory τ. In practice, it's common to store the experience in the form $\mathcal{D} = \{(s_0, a_0, r_0), (s_1, a_1, r_1), \dots, (s_{T-1}, a_{T-1}, r_{T-1}), (s_T, \cdot, 0)\}$. □

3.1.5 Return

The rewards $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$ returned at every step by the environment have been defined as *immediate*, which means generated from and related only to the transition $t \to t+1$. From the description of the RL setup provided in 3.1, it could be intuitive to conclude that the reward maximisation process should allow, in some way, to consider possibly delayed rewards that could occur over a trajectory. In other words, an action taken at time t in state s_t may not immediately produce a high reward r_t, but it could lead to future states associated with high rewards. With this intuition in mind, we can define the *return* at time t as the sum of all the immediate rewards received until termination:

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_T = \sum_{k=0}^{T-t} r_{t+k}.$$

This equation defines the *finite-horizon undiscounted return*, which suits episodic tasks well. However, if applied to continuing tasks, the sum could not converge

to a limit value because the final time step is $T = \infty$. In this setting, we can introduce the *infinite-horizon discounted return*:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}.$$

The term $\gamma \in [0, 1]$ is called *discount factor*, and allows to obtain a bounded return. The discount factor is one of the most critical hyper-parameters to tune since it controls how farsighted the agent could be once trained.

Considering a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T)$, we denote as $R(\tau)$ its discounted return, defined as:

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) = \sum_{t=0}^{T-1} \gamma^t r_t.$$

Remark 3.1.5 (Variance in the bias-variance trade-off)**.** During the training phase, many RL algorithms try to estimate the expected return of each state of the environment. If we take multiple trajectories starting from the same state but evolving differently, the computed return of that state for each trajectory may differ considerably. In fact, the return sums – and optionally discounts – the collected future rewards. A possible estimation of a state's return could be obtained by averaging the computed return of all trajectories. The variance of the estimated return constitutes one element of the *bias-variance* trade-off, that characterises many policy learning methods. □

3.1.6 The Reinforcement Learning Problem

Combining all these concepts allows defining the RL problem more pragmatically. In our setting, it is defined as follows:

> The maximisation of the infinite-horizon discounted return $R(\tau)$ of a trajectory τ sampled following a stochastic policy $\pi(\cdot|s_t)$ from an uncertain environment modelled with an unknown state transition probability $\mathcal{P}(\cdot|s_t, a_t)$ and unknown reward function $\mathcal{R}(s_t, a_t, s_{t+1})$.

In this setting, we can compute the probability of a trajectory τ [Sutton et al., 2018; Achiam, 2018] of an arbitrary length T as[13]:

$$P(\tau \,|\, \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} \mathcal{P}(s_{t+1} \,|\, s_t, a_t)\pi(a_t \,|\, s_t) \tag{3.4}$$

We can now formulate the reward maximisation objective as an optimisation problem by introducing the following *performance function*:

$$J(\pi) = \int_{\tau} P(\tau \,|\, \pi)R(\tau)\,\mathrm{d}\tau = \underset{\tau\sim\pi}{\mathbb{E}}\left[R(\tau)\right]. \tag{3.5}$$

Finally, the RL problem can be mathematically defined as follows:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, J(\pi),$$

where π^* is the *optimal policy* yielding the maximum return from each visited state along the trajectory.

13 For continuing tasks, we can truncate the trajectory after T steps and handle the return of the last state properly.

3.2 REINFORCEMENT LEARNING FORMALISM

The previous section provided an informal introduction to the Reinforcement Learning setting. In this section, we consolidate and formalise the notions into a structured framework composed of two key ingredients: Markov Decision Processes and the Bellman equation.

3.2.1 Markov Decision Processes

Markov Decision Processes [Puterman, 2005; Sutton et al., 2018] are one of the classical formulations of sequential decision-making, which introduce the mathematical framework of choice for the discrete-time stochastic setting described in Section 3.1.

With the notation introduced in the previous section:

- The set of all valid states $\mathcal{S}$,
- The set of all valid actions $\mathcal{A}$,
- The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$,
- The state-transition probability function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Pr[\mathcal{S}]$,
- The initial state distribution ρ_0,

we define as Markov Decision Process (MDP) the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \rho_0 \rangle$. The two key features of the iterative RL settings are correctly modelled by a MDP. Firstly, MDPs satisfy the *Markov Property*.

Definition 3.2.1 (Markov Property)**.** A stochastic process satisfies the Markov property if the conditional probability distribution of future states of the process depends only upon the present state. Given the present state, the future does not depend on the past ones. A process with this property is said to be Markovian or a Markov process. □

It follows that the dynamics of an MDP is uniquely modelled by $\mathcal{P}$ [Sutton et al., 2018]. Given a state-action pair $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, the probability of transitioning to the next state s_{t+1} is $\mathcal{P}(s_{t+1} \mid s_t, a_t)$. Concerning the support of delayed rewards to solve the credit assignment problem, which represents the second key feature of iterative RL, MDPs exploit *value functions*.

3.2.2 Value functions

Given a Markov Decision Process $\mathcal{M}$, we can associate each state s – or state-action pair (s, a) – to a scalar value representing how rewarding it is for an agent to generate a trajectory passing through it. Considering the objective of reward maximisation, a natural choice for this score is the *expected return*. We introduce the *state-value function for policy* π as the following quantity:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau) \,|\, s_0 = s\right] = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r_t \,\middle|\, s_0 = s\right]. \tag{3.6}$$

It provides the expected return of a trajectory starting from s and always acting following policy π. This is also the reason why we need to specify the active policy with the superscript.

Another important value function to introduce is the *action-value function for policy* π:

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau) \,|\, s_0 = s, a_0 = a\right] = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r_t \,\middle|\, s_0 = s, a_0 = a\right]. \tag{3.7}$$

It provides the expected return of a trajectory starting from s, taking an action a – not necessarily generated by the same policy π –, and then always acting following policy π. In this case, we assign a scalar value to each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$.

The definitions of these value functions now give us a helpful metric to define the performance of a policy π. In fact, we can consider policy π_A better than policy π_B if $V^{\pi_A}(s) > V^{\pi_B}(s)$ for all states $s \in \mathcal{S}$. An MDP $\mathcal{M}$ always has at least one policy that performs better than all the others, and this is the optimal policy π_*. The optimal state-value function and the optimal action-value function of the policy π_* (or policies, if the optimal policy is not unique) are:

$$\begin{aligned} V^*(s) &= \max_{\pi} V^{\pi}(s) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau) \,|\, s_0 = s\right] \\ Q^*(s, a) &= \max_{\pi} Q^{\pi}(s, a) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[R(\tau) \,|\, s_0 = s, a_0 = a\right]. \end{aligned}$$

It is often not needed to compute both value functions. Two connections between the value functions are the following [Sutton et al., 2018]:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a\sim\pi}\left[Q^{\pi}(s,a)\right] \tag{3.8}$$

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{s_{t+1}\sim\mathcal{P}(\cdot\,|\,s_t,a_t)}\left[r_t + \gamma V^{\pi}(s_{t+1}) \,|\, s_t = s, a_t = a\right]. \tag{3.9}$$

Remark 3.2.1 (Bias in the bias-variance trade-off)**.** Many RL algorithms operating on continuous spaces, during the training phase, try to fit a function to estimate the optimal value function (or functions). Before reaching an acceptable convergence, the effect of using a wrong value function estimate that could affect training performance is called *bias*, and it is the other element of the previously introduced *bias-variance* trade-off that characterises many policy learning methods (Remark 3.1.5). □

A commonly-used byproduct of the state-value and action-value functions is the *advantage function*:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{3.10}$$

It plays an important role when we need to describe the quality of an action in a relative sense. In fact, we can think of $V(s_t)$ as a function providing the expected return of state s_t averaged over all the possible actions a_t that can be taken in this state[14], and $Q(s_t, a_t)$ as providing the expected return of state s_t considering that the action taken was a_t. If this action a_t performs better than average, expressed mathematically as $Q(s_t, a_t) > V(s_t) \implies A(s_t, a_t) > 0$, we could use this information to reinforce the choice of a_t the next time the trajectory evolves through s_t.

14 This can be clearly seen from Equation (3.8).

3.2.3 Bellman Equation

An optimisation problem in discrete time like the RL problem can be structured in a recursive form, i.e. expressing the relationship between a quantity in one step and the same quantity in the next one. If a problem can be structured in this form, the equation expressing the update rule between the two periods is known as *Bellman equation*.

The value functions for policy π introduced in Section 3.2.2 can be transformed into a recursive form by noticing that we can express the return as follows:

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = r_t + \gamma \left(r_{t+1} + \gamma r_{t+2} + \cdots \right) \\ &= r_t + \gamma R_{t+1} = \mathcal{R}(s_t, a_t, s_{t+1}) + \gamma R_{t+1}. \end{aligned}$$

Replacing this relation in Equation (3.6) and Equation (3.7) leads to the Bellman equations of the value functions for policy π:

$$\begin{aligned} V^\pi(s_t) &= \mathop{\mathbb{E}}_{\substack{a_t \sim \pi \\ s_{t+1} \sim \mathcal{P}}} \left[\mathcal{R}(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1}) \right], \\ Q^\pi(s_t, a_t) &= \mathop{\mathbb{E}}_{s_{t+1} \sim \mathcal{P}} \left[\mathcal{R}(s_t, a_t, s_{t+1}) + \gamma \mathop{\mathbb{E}}_{a_{t+1} \sim \pi} \left[Q^\pi(s_{t+1}, a_{t+1}) \right] \right]. \end{aligned}$$

Once the optimal policy π^* has been found, the Bellman equations for this policy are the same except from the action selection that, instead of sampling it as $a \sim \pi(\cdot|s)$, deterministically selects the action yielding the highest value.

One of the possible solutions to problems structured in this recursive form is proceeding by *backward induction*, i.e. considering what is the optimal action to take in the last state of a sequence, and then propagating backward in time. This process has a closed form under specific assumptions. Our setting, characterised by continuous action and state spaces, unknown $\mathcal{P}$ and $\mathcal{R}$, and the usage of function approximation for the value functions and the policy, does not have any closed-form solution. Therefore, at best, we can use iterative approaches.

3.3 ALGORITHMS

After the emergence of Deep Reinforcement Learning, the late 2010s have seen an extensive research effort that led to a great variety of algorithms. In this section, we provide a bird's-eye view of different families of algorithms that can iteratively solve the RL problem. We highlight their main properties that motivate the choice of methods used for the setting studied in this thesis.

3.3.1 Model-free and Model-based

The formulation of the RL problem provided in Section 3.1.6 outlines that the agent has no knowledge of environment details. The state-transition probability density function $\mathcal{P}(\cdot|s_t, a_t)$ and the reward function $\mathcal{R}(s_t, a_t, s_{t+1})$ are usually unknown. The agent needs to explore the environment through trial-and-error while trying to learn how to maximise the collected immediate rewards. There are, however, algorithms that assume a –possibly partial– knowledge of environment details that can be advantageous in different circumstances.

The first major categorisation separates the algorithms in *model-free* and *model-based* methods, depending on whether the agent has access to a model of the environment. Model-free methods aim to maximise the reward directly from observed data. Instead, model-based methods exploit the knowledge of the environment model to perform planning, enabling to anticipate the direction in which the trajectory will evolve and use this information to improve the action selection. The environment model could either be given, or learned from the observed trajectories. In fact, the agent has access to the environment trajectories, and under the assumption of collecting enough data, it may try to learn both the environment dynamics and reward function.

Having access to the model could seem a desired improvement in most scenarios. In settings where model learning is successful, or the model is given, model-based methods show excellent sample efficiency, one of the major downsides of model-free methods. Still, the reality is that in the case the model is learned from data, it remains highly challenging to obtain a description of a complex environment accurate enough to be exploited by the agent. In addition, the prediction inaccuracy introduces a strong bias in the learning process that can be exploited by the agent, resulting in sub-optimal behaviour in the actual environment.

3.3.2 Value-based and Policy-based

The second major categorisation separates the algorithms in *value-based* and *policy-based* methods. Value-based methods aim to learn the value functions introduced in Section 3.2.2, usually $Q^\pi(s, a)$, from which a policy can be implicitly generated. Instead, policy-based methods do not rely on any value function. They introduce a function approximator whose parameters $\boldsymbol{\theta}$ can be iteratively optimised to maximise a performance function $J(\boldsymbol{\theta})$.

Value-based methods learn a parameterized action-value function $Q_{\boldsymbol{\theta}}(s, a)$ usually applying updates based on the Bellman equations introduced in Section 3.2.3. From the action-value function, actions can be deterministically computed as $a^*(s) \in \operatorname{argmax}_a Q_{\boldsymbol{\theta}}(s, a)$. This equation, however, shows both the limitations of this family of methods. First, due to the action selection based on the argmax, the computation could be expensive in high-dimensional discrete action spaces. Also, they are incompatible with continuous action spaces since the maximisation expects a finite set of actions. Second, they can only learn deterministic policies, introducing the need to be combined with proper exploration strategies that could be less effective than those implicitly implemented with stochastic policies. On the other hand, value-based methods can reuse most of the collected data with high sampling efficiency, and the maximisation strategy allows to improve faster and with a lower variance to the optimal policy.

Policy-based methods, instead of resorting to a value function, perform an optimisation that directly targets the final aim of reward maximisation. They represent a policy with a parameterized function, typically a state-conditioned probability distribution, directly optimised from collected trajectories. Depending on the choice of the policy, these methods are suitable for continuous and high-dimensional action spaces. In practice, they present better convergence properties by applying small incremental changes at every iteration. Although usually slower, less efficient, and more prone to getting stuck and converging to a local optimum, learning performance could be more stable.

The separation between value-based and policy-based methods has become less and less defined in the past few years. Many new algorithms conceptually close to the policy-based methods, do learn and take advantage of value

functions. This family of methods is known as *actor-critic* methods. They combine the broader policy support and better convergence properties of policy-based methods by using a parameterized policy, called actor, with a lower variance obtained from learning and exploiting a value function, called critic. The learning process interleaves optimisations of the actor and the critic so that both converge towards the optimal policy and optimal value function, respectively.

3.3.3 On-policy and off-policy

The third major categorisation separates the algorithms between *on-policy* and *off-policy* methods. The difference between these two methods is whether their policy is used both as *behaviour policy*, used for exploring the environment, and as *target policy*, used as the actual output of the optimisation problem.

Off-policy methods are capable of learning an optimal policy from experience sampled by any exploration strategy, under the assumption of visiting enough times all environment states. This feature makes off-policy methods particularly sample efficient since the sampled experience during training always remains valid and, therefore, can be used multiple times. Most of the algorithms belonging to this family are also value-based, thus they inherit better convergence properties albeit being more unstable.

On-policy methods usually learn a stochastic policy and use it both as behaviour and target. These methods are mainly either policy-based or actor-critic. Under the assumption that the policy's stochasticity is sufficient for environment exploration, on-policy methods share the properties of policy-based methods. Contrarily to off-policy methods, during training, they expect data to be generated from the same policy, preventing the usage of old data acquired in previous optimisation epochs and therefore showing a lower sampling efficiency. However, the most popular algorithms in this family implement techniques based on *importance sampling*, enabling multiple optimisation steps per on-policy batch, mitigating the need for newly-sampled trajectories.

3.4 POLICY OPTIMIZATION

Most of the Reinforcement Learning algorithms used in robotics belong to the family of *policy gradients* methods, i.e. model-free policy-based on-policy methods. In this section, we first derive how we can compute the gradient of the policy performance function $J(\pi_{\boldsymbol{\theta}})$, already introduced in Equation (3.5), w.r.t. its parameterization $\boldsymbol{\theta}$ directly from the trajectories τ. Then, we introduce Proximal Policy Optimization (PPO), a widely used algorithm that exploits a local approximation of this gradient to let $\pi_{\boldsymbol{\theta}}$ converge towards π^*.

3.4.1 Policy Gradient

Let's consider a policy $\pi_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^p$. From Equation (3.5), its performance function can be defined as:

$$J(\pi_{\boldsymbol{\theta}}) = \mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}}[R(\tau)] = \int_{\tau} P(\tau|\boldsymbol{\theta}) R(\tau) \, \mathrm{d}\tau \,. \tag{3.11}$$

We want to derive the equation of the gradient of this performance function w.r.t. $\boldsymbol{\theta}$ (or, at least, its stochastic estimate) so that we can maximise the return by optimising $\boldsymbol{\theta}$ through gradient ascent with the following update rule:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) \Big|_{\boldsymbol{\theta}_k}, \tag{3.12}$$

where $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \left[\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \frac{\partial J}{\partial \theta_3}, \cdots \right]^\top \in \mathbb{R}^p$ is the *policy gradient* term. It can be expanded as:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) &= \nabla_{\boldsymbol{\theta}} \int_{\tau} R(\tau) P(\tau \,|\, \boldsymbol{\theta}) \, \mathrm{d}\tau = \int_{\tau} R(\tau) \nabla_{\boldsymbol{\theta}} P(\tau \,|\, \boldsymbol{\theta}) \, \mathrm{d}\tau \\
&= \int_{\tau} R(\tau) P(\tau \,|\, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log P(\tau \,|\, \boldsymbol{\theta}) \, \mathrm{d}\tau \\
&= \mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[R(\tau) \nabla_{\boldsymbol{\theta}} \log P(\tau \,|\, \boldsymbol{\theta}) \right].
\end{aligned}$$

Note that we used the Leibniz integral rule for differentiation under the integral sign to move the gradient, and the *log-derivative trick* on $\nabla_{\boldsymbol{\theta}} P(\tau|\boldsymbol{\theta})$ to express it as an expectation.

Theorem 3.4.1 (Log-derivative trick)**.** Given a function $f(\boldsymbol{x})$, we can express its gradient in the following form:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = f(\boldsymbol{x}) \nabla_{\boldsymbol{x}} \log f(\boldsymbol{x}). \tag{3.13}$$

Although it seems to involve more terms, this form can be a convenient expression since it can be easier to differentiate the logarithm of a function rather than the function itself. This could occur when f is a product of many terms, which is transformed into a sum by taking the logarithm.

Proof. Given a function $f(\boldsymbol{x})$, from the chain rule it follows:

$$\nabla_{\boldsymbol{x}} \log f(\boldsymbol{x}) = \frac{1}{f(\boldsymbol{x})} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}).$$

Rearranging the terms of this expression leads to Equation (3.13). □

The application of the log-derivative trick has another important consequence. The probability of a trajectory τ was already defined in Equation (3.4) as:

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

If we take its logarithm, we obtain:

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

We note that it depends on $\boldsymbol{\theta}$ only through $\pi_{\boldsymbol{\theta}}(a_t \,|\, s_t)$, therefore we can ignore the other terms and simplify the policy gradient to its final form:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) &= \mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[R(\tau) \nabla_{\boldsymbol{\theta}} \log P(\tau \,|\, \boldsymbol{\theta}) \right] \\ &= \mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[R(\tau) \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \,|\, s_t) \right]. \end{aligned} \tag{3.14}$$

Being an expectation, we can obtain a gradient estimate by collecting at time k a dataset of on-policy trajectories $\mathcal{D}_k$ and computing the *empirical average*, denoted by $\hat{\mathbb{E}}$, over the finite batch of samples:

$$\hat{g}_k = \hat{\mathbb{E}}_t \left[R(\tau) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t) \right] = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} R(\tau) \sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t) \Big|_{\boldsymbol{\theta}_k}. \tag{3.15}$$

Remark 3.4.1 (Gradient variance reduction exploiting causality)**.** The return $R(\tau)$ of Equation (3.14) can be thought of as the weight of the log-likelihoods computed along the trajectory. It can be noted that log-likelihoods at $t > 0$ are weighted by the return computed from $t = 0$, which can be seen as reinforcing an action using a quantity that includes information from the past. Instead, we should consider only the consequences of an action. We can update the policy gradient with this intuition obtaining the *reward-to-go policy gradient*:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t) \sum_{t'=t}^{T-1} \mathcal{R}(s_{t'}, a_{t'}, s_{t'+1}) \right]. \tag{3.16}$$

Now, log-likelihoods are weighted with the causal return, often called *reward-to-go*:

$$\hat{R}_t = \sum_{t'=t}^{T-1} \mathcal{R}(s_{t'}, a_{t'}, s_{t'+1}).$$

□

3.4.2 Generalized Advantage Estimation

Policy gradient methods are not uniquely defined by the final forms of Equation (3.14) and Equation (3.16). They are just two specific cases of a more general formulation expressed in the following form:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{\infty} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t) \Psi_t \right]. \tag{3.17}$$

Among all possible choices of Ψ_t, the one yielding the lowest variance [Schulman et al., 2018] is the *advantage function*, already introduced in Equation (3.10):

$$\Psi_t = A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t).$$

Using the advantage as log-likelihood's weight can be intuitively understood from its properties to describe the relative quality of actions. In fact, if $A^\pi(s, a) > 0$, the policy gradient $\hat{g}$ pushes the parameters $\boldsymbol{\theta}$ towards an increased action likelihood $\pi_{\boldsymbol{\theta}}$, with the consequence of reinforcing the probability to choose again a_t over other actions. The main reason for the favourable properties regarding the variance reduction introduced by using the advantage function originates from integrating a reinforcement effect based on a quantity that expresses relative feedback instead of an absolute value that could be noisy.

In practice, the advantage function is unknown, and we have to find a way to obtain a valid *advantage estimator* $\hat{A}(s, a)$. There are many methods to obtain $\hat{A}_t$, that can be determined by how the return is estimated: high-variance Monte-Carlo on one end, high-bias one-step TD on the other:

$$\begin{aligned} \hat{A}_t^{(1)} &= -V(s_t) + r_t + \gamma V(s_{t+1}) && \text{one-step TD,} \\ \hat{A}_t^{(\infty)} &= -V(s_t) + \sum_{t=0}^{\infty} \gamma^t r_t && \text{Monte-Carlo.} \end{aligned}$$

Note that $\hat{A}_t^{(1)}$ is also called *TD error*, denoted by δ_t^V. As done by TD methods, the process of approximating future rewards with an estimated return is also known as *boostrapping* and, similarly as we discussed in Remark 3.2.1, it introduces bias.

One may notice that we can blend the two methods by truncating the Monte-Carlo estimate after k steps, and approximate the future rewards from the return corresponding to the k-th state. This approach, known as k-step TD, allows balancing the bias-variance trade-off by interpolating between the two extremes:

$$\hat{A}_t^{(k)} = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}).$$

Think of k as an additional hyperparameter to tune. Intuitively, the variance is reduced by considering a smaller number of noisy sampled rewards, and the bias is mitigated by the high discount applied to the bootstrapped value.

While k-step TD already helps trade off bias and variance, it selects just one horizon. More sophisticated methods compute $\hat{A}_t^{(k)}$ on multiple horizons and then combine the estimates, for example, by averaging them. The Generalized Advantage Estimator (GAE) [Schulman et al., 2018] is defined as an exponentially-weighted average of k-step TD estimates:

$$\begin{aligned}\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} &= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots\right) \\ &= \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V.\end{aligned}$$

We can recognise $\mathrm{GAE}(\gamma, 0) = \hat{A}_t^{(1)}$ and $\mathrm{GAE}(\gamma, 1) = \hat{A}_t^{(\infty)}$. The hyperparameter $0 < \lambda < 1$ balances between these two extremes.

3.4.3 Proximal Policy Optimization

In the previous sections, we obtained the generic Equation (3.17) of the policy gradient, whose advantage-based form can be empirically estimated over a finite batch of samples as follows:

$$\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_t \mid s_t)\hat{A}_t\right].$$

In practice, implementations of Policy Gradient (PG) use Automatic Differentiation frameworks to calculate a loss function and, from it, extract the gradient. The loss function typically used is the following:

$$L^{PG}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t\left[\log\pi_{\boldsymbol{\theta}}(a_t \mid s_t)\hat{A}_t\right]. \tag{3.18}$$

Given a dataset of trajectories $\mathcal{D}$, one might think to perform multiple steps of gradient ascent. Often, this process generates steps too large in the policy's parameters space, leading to updates that destroy the previously learned behaviour.

Authors of [Schulman et al., 2017a] have proposed a modification of Equation (3.18) that guarantees the monotonic improvement of a stochastic policy. The Trust Region Policy Optimization (TRPO) algorithm exploits importance sampling for correcting the probabilities of trajectories produced by different policies, and limits how much these policies can change between two iterations using a hard constraint on their Kullback–Leibler (KL) divergence:

$$\begin{aligned} &\underset{\boldsymbol{\theta}}{\text{maximize}} \;\; \hat{\mathbb{E}}_t \left[\frac{\pi_{\boldsymbol{\theta}}(a_t \mid s_t)}{\pi_{\boldsymbol{\theta}_{old}}(a_t \mid s_t)} \hat{A}_t \right] \\ &\text{subject to} \;\; \hat{\mathbb{E}}_t \left[D_{KL} \left[\pi_{\boldsymbol{\theta}}(\cdot|s_t) || \pi_{\boldsymbol{\theta}_{old}}(\cdot|s_t) \right] \right] \leq \delta. \end{aligned}$$

This algorithm, in practice, relies on the computation of a linear approximation of the objective, and a quadratic approximation of the constraint.

PPO [Schulman et al., 2017b] builds upon similar intuitions, replacing the complicated computation of the approximated constrained problem with techniques that, although less rigorous, are much simpler and effective in practice. Authors provide two different variants of the algorithm: *clipped surrogate objective* and *adaptive KL penalty coefficient*. The two approaches can be either considered as standalone methods or combined.

The clipped variant of the algorithm removes the hard constraint of TRPO. Denoting the likelihood ratio as $r_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(a_t \mid s_t)}{\pi_{\boldsymbol{\theta}_{old}}(a_t \mid s_t)}$, this variant modifies the loss function as follows:

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

While optimising over a dataset $\mathcal{D}$, after the optimisation of the first batch, $\boldsymbol{\theta} \neq \boldsymbol{\theta}_{old}$. Clipping the likelihood ratio in the interval $1 \pm \epsilon$ has the effect of dampening potentially large steps towards the reinforcement direction ($r_t(\boldsymbol{\theta}) \gg 1$ if $\hat{A}_t > 0$, and $r_t(\boldsymbol{\theta}) \ll 1$ if $\hat{A}_t < 0$).

The penalty variant of the algorithm, instead, transforms the hard constraint of TRPO into a soft constraint:

$$L^{KLPEN}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[r_t(\boldsymbol{\theta}) \hat{A}_t - \beta D_{KL} \left[\pi_{\boldsymbol{\theta}_{old}}(\cdot|s_t) \,||\, \pi_{\boldsymbol{\theta}}(\cdot|s_t) \right] \right].$$

While selecting a constant β is a possibility, the authors of PPO proposed an adaptive parameter update. After each policy update, the KL divergence of the

target policy $\pi_{\boldsymbol{\theta}}$ from the behaviour policy $\pi_{\boldsymbol{\theta}_{old}}$ can be computed (or, if the parameterized policy has no closed-form expression, estimated) as:

$$d = D_{KL}\left[\pi_{\boldsymbol{\theta}_{old}}(\cdot|s_t) \,||\, \pi_{\boldsymbol{\theta}}(\cdot|s_t)\right].$$

Then, the new β parameter used in the following policy is adjusted as follows:

$$\beta \leftarrow \beta/2, \quad \text{if } d < d_{targ}/1.5$$
$$\beta \leftarrow \beta \times 2, \quad \text{if } d > d_{targ} \times 1.5.$$

The constant d_{targ} is the desired value of the KL divergence.

Intelligence is what you use when you don't know what to do.

— Jean Piaget

4 STATE-OF-THE-ART AND THESIS CONTENT

This chapter concludes the introductory part of this thesis. With the basic knowledge about robot simulators, robot modelling, and RL provided by the previous chapters, we attempt to draw the state-of-the-art of domains covered by this thesis. Then, from the picture of the current research status, we identify what we believe are problems still open, and outline how this thesis aims to solve them by providing the contributions to knowledge of this work. We start by reviewing the research on RL for robot locomotion, which should provide a bird-eye view of how this methodology evolved and has been applied over the past three decades to the specific domain of interest of this thesis. Then, we review the technological evolution of robot simulators, focusing on robot learning applications. Finally, to provide the necessary context to motivate one of our contributions, we review the domain of push-recovery strategies applied to bipedal robots.

4.1 REVIEW OF REINFORCEMENT LEARNING FOR ROBOT LOCOMOTION

The origin of Reinforcement Learning is often attributed to the development of Q-Learning [Watkins, 1989] in the early 1990s, unifying the three existing threads of learning by trial-and-error, optimal control, and temporal differences. In the same period, major developments such as $\mathrm{TD}(\lambda)$ [Sutton, 1988], REINFORCE [Williams, 1992], SARSA [Rummery et al., 1994], etc. triggered an increasing interest around methodologies based on reinforcement, gaining popularity with the first system able to challenge human ability exploiting NNs: TD-Gammon [Tesauro, 1994]. The first successful attempts to apply Re-

inforcement Learning to robotics, combined with usage of NNs as function approximators, appeared in the same decade [Lin, 1993; Gullapalli et al., 1994; Benbrahim et al., 1997].

In the following period, in the late 1990s and the entire 2000s, the body of research mainly focused on trajectory optimisation, particularly targeting learning from demonstration [Schaal, 1996; Atkeson et al., 1997; Schaal, 1999]. In order to limit the challenges posed by the computational constraints in conjunction with the available hardware, researchers widely exploited parameterized movement primitives [Schaal, 2006] and policy gradients methods [Peters et al., 2003; Peters et al., 2006; Kober et al., 2008], also attempting to bridge the latter with stochastic optimal control [E. A. Theodorou et al., 2010]. Restricting the domain to robot locomotion, this decade was mainly characterised by the usage of quadrupeds [Kohl et al., 2004; Honglak Lee et al., 2006; Kolter et al., 2007; E. Theodorou et al., 2010]. Kober et al. [2013] provide a well-structured and extensive survey on the state and challenges of research in RL applied to robotics characterising this period.

In the early 2010s, after the advent and initial success of DL [Hinton et al., 2006], hardware advancements enabled significant progress in speech recognition and computer vision research. The transfer of similar methodologies to RL, whose performance was previously limited by computational constraints, at that point, was natural, and the interest in their combination, DRL, exploded. Mnih et al. [2015] showed that DRL was suitable for becoming a generic and effective method to learn end-to-end policies also in high-dimensional problems. Similar ideas have been transferred to the control of simulated robots by Lillicrap et al. [2016], showing that this category of algorithms can train end-to-end policies with performance comparable to those obtained by methods that can access the complete system's dynamics. In the same years, the RL community has been very prolific and produced a large number of algorithms, for example, TRPO [Schulman et al., 2017a], PPO [Schulman et al., 2017b], and Soft Actor-Critic (SAC) [Haarnoja et al., 2018], to mention a few among those that have experienced a wide adoption.

The emergence of DRL, together with new algorithms, provided a strong thrust also in their transfer to the robotics domain. Limiting the scope to *quadrupedal locomotion*, many parallel research directions emerged over time.

Researchers succeeded to transfer policies from simulation to real-world robots [Tan et al., 2018; Jemin Hwangbo et al., 2019] introducing models of the actuation dynamics in the training process. Other studies, exploiting imitation learning from motion data, proposed techniques to learn policies for simulated robots [Peng et al., 2020] that were later adapted to their real-world counterparts [Smith et al., 2021]. Other methods integrated high-level planners and low-level controllers within the learning pipeline [Tsounis et al., 2020], used the learning component as corrective action [Gangapurwala et al., 2021], performed control in Cartesian space [Bellegarda et al., 2021], or exploited hardware accelerators to speed-up learning [Rudin et al., 2021].

For what concerns research on *bipedal locomotion*, instead, the literature is more sparse. The work of Heess et al. [2017] posed one of the first milestones in end-to-end learning of locomotion behaviours from simple reward signals, exploiting a distributed version of PPO and a curriculum learning strategy that makes the learned task progressively more challenging. A second relevant work comes from the computer graphics community, which nowadays shares multiple research directions with robot learning. Peng et al. [2018a] proposed a method that, utilising motion capture data of highly-dynamic movements, produces a policy that can execute and adapt the collected trajectories on a simulated character. Regarding the transfer of policies to real bipeds, instead, Xie et al. [2019] proposed to learn locomotion policies leveraging data from generated from pre-existing controllers. Castillo et al. [2021] developed a cascade control architecture to train a trajectory planning policy that is deployed on a real robot, and Li et al. [2021] were able to track high-level velocity commands on their bipedal robot with a policy learned in simulation with an extensive use of domain randomization. Finally, Rodriguez et al. [2021] proposed a methodology based on curriculum learning to train a single policy capable of controlling a humanoid robot for omnidirectional walking, and Bloesch et al. [2022] introduced an end-to-end method to learn walking gaits without relying on simulated data.

4.2 REVIEW OF SIMULATORS FOR ROBOT LEARNING

All works discussed in the previous section proposed algorithms, architectures, and learning pipelines that, during the training phase of a policy, require a constant flow of new data sampled from the controlled robot. In most cases, obtaining the necessary amount of data from physical robots would either take too long or be dangerous due to the high probability of damaging the system and its surroundings as a consequence of the inherent trial-and-error nature of RL. Many of the contributions in the robot learning domain exploit rigid-body simulators and robot models to generate enough state transitions for the learning process to converge to a satisfying solution. Due to the unavoidable introduction of approximations, the evolution of rigid-body simulations will differ from the evolution of the real system. This mismatch, for example, could originate from the estimation error of the inertial parameters of the simulated robot, from the wrong assumption of a perfectly rigid body, from the simplistic modelling of the actuation dynamics, from the approximations of the contact dynamics and contact parameters, from the mismatch between simulated and real sensors noise, etc. Agents trained in environments characterised by such approximations, which could be referred all together as *reality gap*, could learn to exploit modelling approximations to reach unrealistic performance that could never be transferred elsewhere. The most popular method to mitigate the occurrence of this behaviour is *domain randomization* [Peng et al., 2018b; Muratore et al., 2022], widely studied in sim-to-real research [Zhao et al., 2020], which aims to regularise the simulation with different methods to prevent overfitting during training. In the rest of this section, we bypass similar methodologies that could be applied to any simulation, and focus instead on providing a review of common simulators that could be used for robot learning and, particularly, locomotion applications.

The process of describing the evolution of a dynamical system is deeply rooted in control and systems theory. As regards physical robots, despite the theory behind the evolution of their rigid-body description has been known for centuries, the advent of general-purpose simulators did not occur until the early 2000s. The most established simulator that nowadays is still widely used is Gazebo [Koenig et al., 2004]. It interfaces with multiple physics

engines, using ODE[15] as a default, and supports importing models and world descriptions from the SDF and the URDF files. A second very popular simulator, mainly for robot learning, is Mujoco [E. Todorov et al., 2012]. It is specifically designed for model-based optimisation with a particular focus on contact dynamics. Another common simulator supporting multiple physics engines and common descriptions is CoppeliaSim [Rohmer et al., 2013], formerly known as V-Rep. It features Bullet [Coumans et al., 2016] as the default physics engine. More recently, Nvidia released its general-purpose toolkit Nvidia Isaac[16] that integrates with their PhysX[17] physics engine.

Beyond general-purpose simulators, many standalone physics engines can be used to simulate multibody systems. PyBullet [Coumans et al., 2016], despite its origins in videogame development, is another common option in robot learning research. DART [Jeongseok Lee et al., 2018], also available in Gazebo, interfaces with several collision detectors and constraint solvers, and was recently used as the basis of Nimble Physics [Werling et al., 2021] which adds differentiable physics support.

Differentiable physics originates from differentiable programming [Innes et al., 2019] and scientific machine learning [Rackauckas et al., 2021], in which physics laws are implemented with AD [Baydin et al., 2018] frameworks that allow to propagate the gradients through their calculation. In the past few years, it has become a popular research direction that has yielded outstanding results in many fields. In the domain of robotics, differentiability is still under scrutiny [Suh et al., 2022] and research is active [Gillen et al., 2022]. On the one hand, common RBDAs have been studied to compute analytical gradients [Carpentier et al., 2018; Belbute-Peres et al., 2018; Singh et al., 2022]. On the other hand, the entire computational flow was implemented with AD frameworks [Freeman et al., 2021; Howell et al., 2022].

On a similar note, simulations implemented entirely with AD frameworks could be executed directly on hardware accelerators like GPUs or TPUs, as an alternative to CPUs either on a single machine or a cluster. Even though CPU simulations could be optimised to run fast [J. Hwangbo et al., 2018] under given circumstances, hardware accelerators can provide a degree of

15 https://www.ode.org/
16 https://developer.nvidia.com/isaac-sdk
17 https://developer.nvidia.com/physx-sdk

scalability that outperforms any CPU-based solution. Thanks to their hardware, Nvidia pioneered this domain with PhysX. More recent development showed remarkable learning speed with their Isaac Gym framework [J. Liang et al., 2018; Makoviychuk et al., 2021; Rudin et al., 2021]. In this realm, the past few years have seen several other attempts to develop comparable solutions [Heiden et al., 2021; Qiao et al., 2021; Freeman et al., 2021].

Regardless of the selected simulator, in RL research it is common to abstract environments and agents, allowing to develop them independently from each other. The most common environment abstraction is provided by OpenAI Gym [Brockman et al., 2016] in Python. Environments employing the `gym.Env` interface can be implemented with any of the simulators discussed in this section, and they can be employed by any RL agent included in the available frameworks supporting this interface.

Considering the increasing interest in simulations and its corresponding speed of advancements, the review provided in this section is far from complete. We conclude by pointing the interested reader to the surveys on the topic, from the early ones [Ivaldi et al., 2014; Erez et al., 2015] to the most recent [Collins et al., 2021; T. Kim et al., 2021; Körber et al., 2021], acknowledging that either they became outdated quickly, or use metrics that are valid only for a few selected use cases.

4.3 REVIEW OF PUSH-RECOVERY STRATEGIES

Locomotion is among the most fundamental capabilities that legged robots need to master in order to become useful in the real world. In the past ten years, quadrupedal locomotion research achieved remarkable results, and nowadays, quadrupeds are able to autonomously navigate hazardous environments with great agility [Joonho Lee et al., 2020; Miki et al., 2022]. Despite decades of research, the situation of bipedal locomotion is quite different, especially when we compare the agility of most of the existing humanoid robots with human capabilities. Many fundamental methods, techniques, and control architectures widely adopted by bipedal locomotion research have been first studied in the simplified case of push recovery. In fact, the ability to react

appropriately to external perturbations is paramount for achieving robust locomotion capabilities, and often advances in push recovery research are preparatory for advances in locomotion research [Jeong et al., 2019].

Humans use various strategies to maintain balance, including *ankle*, *hip*, and *stepping* strategies [Nashner et al., 1985; Maki et al., 1997; Stephens, 2007]. The adoption of these strategies follows an activation proportional to the magnitude of external disturbances. The effectiveness of human capabilities mainly stems from how different strategies are combined into a continuous motion [McGreavy et al., 2020]. The applicability of these principles for the generation of control actions applied to robots has traditionally relied on *simplified models* approximating their dynamics, such as the Linear Inverted Pendulum (LIP) model [Kajita et al., 2001] and the Capture Point (CP) [Pratt et al., 2006]. Together with the formulation of the Zero Moment Point (ZMP) [Vukobratovic et al., 1969; Vukobratović et al., 2004] widely adopted as a stability criterion, simplified models became very popular and still used nowadays. Modern applications alternatively rely on the Divergent Component of Motion (DCM), that can be seen as an extension of the CP theory [Shafiee et al., 2019].

The structure of control algorithms utilizing any of these models, however, typically considers only the CoM of the robot. The generation of the actual joint strategy is usually achieved through either hierarchical [Feng et al., 2014] or predictive [Wieber, 2006; Aftab et al., 2012] architectures. Implementing an effective and robust blending of all the discrete strategies (ankle, hip, stepping, etc.) has been considered challenging and prone to failures, even with careful tuning [McGreavy et al., 2020]. Nonetheless, their usage still represents an active research area that can achieve promising results [Jeong et al., 2019].

In the past few years, the robotics community attempted to develop methods based on robot learning and, in particular, RL to generate a unified control action that automatically blends the discrete strategies [Yang et al., 2018]. Early results have been demonstrated capable of controlling the lower limbs of a simulated humanoid robot [H. Kim et al., 2019] and, more recently, a real exoskeleton [Duburcq et al., 2022].

4.4 THESIS CONTENT

The previous sections outlined the history, the overall status, and the most recent breakthroughs in the domains of reinforcement learning for robot locomotion, rigid-body simulations for robot learning, and push-recovery strategies for humanoid robots. In view of the subject of this thesis regarding how modern technology can help us generate synthetic data for humanoid robot planning and control, and considering the three reviewed research domains, we conclude this chapter by identifying problems still open and detailing how the contributions to knowledge provided in the next Part ii aim to solve them.

4.4.1 Chapter 5: Reinforcement Learning Environments for Robotics

OPEN PROBLEM The number of frameworks for RL research is constantly increasing. While the `gym.Env` interface became the de-facto abstraction layer to isolate agents and environments, an appropriate structure for environments has never been properly outlined. In robot learning applications, the implementation of the decision-making logic related to the task to learn is often intertwined with the setting in which it is executed, that could be either in simulation or in the real world. Therefore, usually environments cannot be transferred between different settings without significant refactoring. Furthermore, in simulated environments, the choice of the simulator and how it communicates with the environment could undermine the reproducibility of the simulation, resulting in different outcomes at each execution.

CONTEXT OF CONTRIBUTION We present a framework for developing robotic environments for Reinforcement Learning research. The framework is composed of two main components. At the lower level, the Scenario C++ abstraction layer provides different interfaces of entities part of a scene like World and Model, with additional Link and Joint interfaces to simplify the interaction. Scenario is not strictly related to robot learning, but it provides a unified interface that could be implemented to communicate either with

simulated robots running in any simulator, or with real robots, passing through, e.g., a robotic middleware. We currently provide a complete implementation for simulated robots interfacing with the new Gazebo Sim[18] simulator, supporting the DART and Bullet physics engines. At the high level, the *Gym-Ignition* Python package[19] provides abstraction layers of the different components that typically form a robotic environment: the Task and the Runtime. On the one hand, the Task provides the necessary components to develop agnostic decision-making logic with the generic Scenario APIs, and on the other hand, the Runtime provides the actual interfacing either with a simulator or a real robot. This whole architecture the user to only develop the Tasks only once, and use them for training, executing, or refining a policy in any of the supported Runtimes. The selected Gazebo Sim simulator has multiple advantages over alternative options for generating synthetic data. Its plugin-based architecture allows third-party developers to integrate custom physics engines and custom rendering engines, enabling them to develop agnostic environments that select the desired engines during runtime. For robot learning, if needed, it enables seamlessly switching engines, opening the possibility to add them as a whole in the domain randomization set. Furthermore, considering the wide adoption of Gazebo within the robotics community, it enables roboticists to create environments using familiar tools, guaranteeing that their execution remains reproducible thanks to a single-process architecture not possible with the previous generations of the simulator.

CONTRIBUTION OUTPUTS A short form of the contribution to knowledge described in this chapter was first presented in 2019 at the following workshop:

> **Gym- Ignition: Reproducible Robotic Simulations for Reinforcement Learning**
>
> Diego Ferigo, Silvio Traversaro, Daniele Pucci
>
> *Robotics: Science and Systems (RSS) - Workshop on Closing the Reality Gap in Sim2real Transfer for Robotic Manipulation*, 2019

18 `https://gazebosim.org/`

19 In its earlier releases, the new Gazebo Sim simulator was called Ignition Gazebo, from what derives the name of our Python package.

CRediT **DF:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - Original Draft, Writing - Review & Editing, Visualization; **ST:** Supervision; **DP:** Resources, Supervision, Funding acquisition.

Its extended version has been later presented at the following IEEE conference:

Gym- Ignition: Reproducible Robotic Simulations for Reinforcement Learning
Diego Ferigo, Silvio Traversaro, Giorgio Metta, Daniele Pucci
International Symposium on System Integration (SII), 2020

CRediT **DF:** Conceptualization, Methodology, Software, Validation, Investigation, Writing - Original Draft, Writing - Review & Editing, Visualization; **ST:** Supervision; **GM:** Funding acquisition; **DP:** Resources, Supervision, Funding acquisition.

The two components of the presented framework, Scenario and Gym-Ignition, have been open-sourced and are publicly available at the following link: `https://github.com/robotology/gym-ignition`.

4.4.2 Chapter 6: Learning from scratch exploiting robot models

For validating the framework proposed in the previous contribution, we identify a challenging problem affecting humanoid robots and try to find a solution by framing it as a RL problem.

OPEN PROBLEM Generating the appropriate control action for highly-dynamic movements has always been challenging in robotics, especially when the controlled system is redundant and under-actuated like a humanoid robot. Traditional methods based on control theory and optimal control either rely on accurate descriptions of the dynamics, or exploit approximations in the form of either simplified or reduced models. Methods based on control theory strongly rely on the accuracy of the dynamic model, to the point of failure in the presence of a mismatch sufficiently large that controller's robustness is unable to compensate for it adequately. Instead, those based on optimal control

often constrain the space of individual control actions and, when dealing with multiple options, their automatic selection might require careful and often manual tuning. Furthermore, with the increase in the controlled DoFs and the number of optimisation constraints, this family of methods is still facing computational challenges in real-time settings. An alternative direction for this category of problems is RL, providing a unified learning framework that, given a meaningful reward signal, does not require the knowledge of the controlled system's dynamics. However, regardless of its accuracy, the dynamic model can provide interesting priors that could benefit learning. Without leaving the model-free RL setting, these priors could be exploited in the reward design process, also known as *reward shaping*.

CONTEXT OF CONTRIBUTION We consider the problem of synthesising a control action capable of employing different balancing strategies for push recovery on a simulated humanoid robot. The control architecture consists of a high-level policy trained with model-free DRL generating joint velocity references, actuated by low-level PID controllers. The controller operates on a model of the iCub humanoid robot, controlling 23 DoFs of its legs, torso, and arms. By applying external forces to the pelvis of the robot during training, we reward the agent utilising specific terms depending on the obtained configuration: steady-state when balancing is successful, and transient during the recovery phase. In addition, similarly to common practice in optimal control, we also include regularisation reward terms to smoothen the control action. Instead of learning a model at the agent level as is done in model-based RL, we shape the reward function with multiple components computed from the robot description, that slightly differs from the simulated model after the application of domain randomization over some of its parameters. We show that the emerged balancing behaviour blends together different balancing strategies showing the usage of ankles, hips, stepping, and momentum, obtaining as a result a single whole-body push-recovery strategy. This work aims to validate the architecture proposed in the previous chapter, since the training pipeline implements the decision-making task with the Gazebo backend of Scenario and Gym-Ignition, exposing the environment to a framework providing a PPO agent.

CONTRIBUTION OUTPUTS The contribution described in this chapter was published in the following journal and presented in 2021 at the IEEE International Conference on Humanoid Robots:

> **On the Emergence of Whole-body Strategies from Humanoid Robot Push-recovery Learning**
> Diego Ferigo, Raffaello Camoriano, Paolo Maria Viceconte, Daniele Calandriello, Silvio Traversaro, Lorenzo Rosasco, Daniele Pucci
> *Robotics and Automation Letters*, 2021
>
> CRediT **DF:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization; **RC:** Conceptualization, Methodology, Formal analysis, Investigation, Writing - Original Draft, Writing - Review & Editing, Visualization, Project administration; **PMV:** Methodology, Software, Formal analysis, Investigation, Writing - Original Draft, Writing - Review & Editing, Visualization; **DC:** Supervision, Writing - Review & Editing; **ST:** Supervision; **LR:** Resources, Funding acquisition; **DP:** Resources, Supervision, Funding acquisition.

This work was nominated by the conference committee among the finalists for the Best Paper Award in the Interactive Session.

4.4.3 Chapter 7: Contact-aware Multibody Dynamics

Software architectures typically used to train RL policies for application in robotics, in most cases rely on general-purpose rigid-body simulators from which experience is sampled. As experienced in the experiment of the previous contribution, often the bottleneck that limits the performance of such architectures does not reside in the optimisation problem that utilises the data, but rather in the process of data generation.

OPEN PROBLEM Most of the physics engines included in general-purpose simulators, besides computing the evolution of a multibody system considering its law of motion, also need to implement routines for detecting and solving collisions. In the overall computation, these routines often become the real bottleneck, limiting the speed of the entire simulation. Applications having high sampling throughput as their main target might strongly benefit from less general but more optimised execution. Robot locomotion applications may not need many of the features provided by general-purpose simulators, opening the possibility of sacrificing some of them in exchange for higher sampling efficiency.

CONTEXT OF CONTRIBUTION We describe the multibody dynamics in reduced coordinates as a dynamical system in state-space capable of modelling free-floating robots. With locomotion applications in mind, we propose a dynamical system augmenting the multibody EoMs with contacts in presence of uneven smooth terrain, formulating a soft-contacts model capable to handle both sticking and slipping states without approximating the corresponding friction cone. To this end, we extend existing soft-contact models developed for sphere-plane surface to a more generic point-surface setting. When combined with the multibody EoMs, we obtain a continuous, albeit stiff, state-space representation of the dynamical system whose trajectories can be computed by any numerical integration scheme.

CONTRIBUTION OUTPUTS The activities leading to the publication of the contribution to knowledge are currently ongoing.

> CRediT **Diego Ferigo:** Conceptualization, Methodology, Validation, Formal analysis, Investigation; **Silvio Traversaro:** Supervision; **Daniele Pucci:** Resources, Supervision, Funding acquisition.

4.4.4 Chapter 8: Scaling Rigid Body Simulations

Towards the aim of maximising the performance of sampling synthetic data for robot locomotion started in the previous contribution, we combine the contact-aware state-space representation of free-floating robots dynamics with state-of-the-art RBDAs to create a novel physics engine that can exploit modern hardware accelerators.

OPEN PROBLEM Training policies in a simulated setting is still the predominant choice in RL research. Although sim-to-real methodologies are progressing rapidly, and the community is actively researching on either learning directly from real-world robots or from offline transitions, simulations remain a central component in this domain. In the setting of robot locomotion, the training process involves sampling simulated trajectories of a multibody system endowed with a considerable number of DoFs interacting, at least, with the terrain surface. Most of the general-purpose simulators currently available perform computations entirely on CPUs. Despite the most advanced frameworks providing RL agents support the parallel execution of multiple environments either on a single machine or on a cluster, sampling trajectories represents the main bottleneck of the entire learning pipeline. It is not uncommon for a single training experiment to last multiple days. Especially when visual perception is not necessary, and function approximators are not excessively large models, most of the training time is spent sampling new simulated data rather than optimising the policy. In the era of big data, simulators for robotics are not yet fast enough [Choi et al., 2021].

CONTEXT OF CONTRIBUTION Inspired by the results shown by Freeman et al. [2021], we propose JAXsim, a new physics engine in reduced coordinates capable of simulating multibody systems on CPUs, GPUs, and TPUs. The key to the transparent execution on different devices is the exploitation of the contact-aware multibody dynamics introduced in Chapter 7. It enables the development of efficient routines, not relying on any dynamic allocation that can be deployed on hardware accelerators. The possibility to execute the simulation entirely on hardware accelerators represents an essential feature for

applications requiring the generation of a massive amount of data like those belonging to the robot learning domain. We also describe state-of-the-art RBDAs proposed by Featherstone [2008] with the notation introduced in Chapter 2 that can be executed on this hardware, and provide an updated version of the Recursive Newton-Euler Algorithm (RNEA) compatible with floating-base robots, that exactly corresponds to the inverse of forward dynamics algorithms like Articulated Body Algorithm (ABA).

CONTRIBUTION OUTPUTS The activities leading to the publication of the contribution to knowledge are currently ongoing.

> CREDIT **Diego Ferigo:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data Curation, Visualization; **Silvio Traversaro:** Supervision; **Daniele Pucci:** Resources, Supervision, Funding acquisition.

The software components described in this chapter have been open-sourced and are publicly available at the following link: `https://github.com/ami-iit/jaxsim`.

Part II

CONTRIBUTION

The real danger is not that computers will begin to think like men,
but that men will begin to think like computers.
— Sydney J. Harris

5 REINFORCEMENT LEARNING ENVIRONMENTS FOR ROBOTICS

Training a RL agent necessarily passes through a process that intertwines sampling data according to the underlying MDP and optimising either one or multiple function approximators. In this thesis, we focus on the sampling side of these training architectures, belonging to the environment block of the RL problem illustrated in Figure 3.1. We consider environments for robotic applications with their domain-specific needs and limitations.

In this chapter, we study frameworks to create simulated robotic environments from which synthetic data is sampled and used to train RL policies. With the aim of transferring the obtained policies to real robots, we further specialise our analysis in software architectures that allow us to bridge simulation and real world. We first identify high-level properties we consider important in this setting. Then, based on these properties, we proceed by selecting, describing, and categorising the major frameworks providing robotic environments accessible by the research community. Finally, we propose a new framework for developing robotic environments, starting with a description of the design goals, and continuing with the implemented software architecture. This new framework will be validated in the next chapter, where it is used to learn a push-recovery policy for balancing a simulated humanoid robot.

5.1 FRAMEWORKS PROVIDING ROBOTIC ENVIRONMENTS

5.1.1 Properties

In this section, we define the properties characterising frameworks that provide robotic RL environments. We start from the properties already identified for robot simulators in Section 1.1.2, framing their application for robot learning. We also introduce new properties more specific to the context of RL, that includes and expands the traditional applications of robot control. The properties of this section will be used in the following to draw a comparison of existing frameworks providing robotic environments.

REPRODUCIBILITY A simulation is reproducible if different executions of the same scene under the same control actions yield the same simulated trajectories. Environments that interface with simulators implementing a client-server architecture based on network transport could not become reproducible due to the operating system's possibility to preempt the network sockets' processes, particularly when the load of the system increases. Furthermore, complex architectures exposing environments might require generating random data from different components. Reproducibility, in this case, can be enforced only with careful handling and propagation of the seed to all the Random Number Generators (RNGs) used by the framework.

MODULARITY Reinforcement Learning is one of the most generic learning frameworks. Its structure is composed of a learning agent interacting with an environment, as illustrated in Figure 3.1. While most of the related software architectures already map their component based on this high-level structure, specific implementations for robotics might benefit from a more fine-grained abstraction, particularly regarding the environment. For instance, we might want to achieve the same learning goal on robots with different mechanical structures. In order to promote code reuse and minimise system integration effort, a modular design that abstracts the robot and the logic of the learned task is a valuable addition.

REAL–TIME COMPATIBILITY The main reason to rely on simulations for training an agent is the low cost of synthetic samples. However, the final goal should be deploying the policy on the real robotic platform. Frameworks should allow to either execute the resulting policy or continue its training on the

real robot with minimal changes. An open problem, though, is how to reset a real-world episode. For instance, in the case of floating-base robots, this operation may require moving the robot back to the initial position in the operating area.

PARALLEL SIMULATION Modern computers are nowadays endowed with multiple computational cores. The independence between simulated instances makes executing parallel environments trivial, maximising the efficient use of local computational resources. On a higher level, the same applies when scaling to multiple machines. Typically, job distribution is performed by frameworks that provide the algorithms. Environment providers should only ensure instances' independence for multithread and multiprocess execution.

ACCELERATED SIMULATION Based on their complexity, simulations can run either faster or slower than real-time. The ratio between real and simulated time is known as RTF. A RTF greater than 1 indicates that the simulation is running in an accelerated state, i.e. faster than real-time. In order to speed up experience collection, environments should be able to run in accelerated mode.

MULTIPLE PHYSICS ENGINES The physics engine is the simulator component that integrates physical equations, evaluates collisions, and solves contact constraints. Classic techniques in domain randomization [Peng et al., 2018b; Ramos et al., 2019] operate on parameters of the physics engine. Supporting multiple back-ends and being able to switch between them on the fly would permit the randomization of the entire physics engine, bringing domain randomization to a higher level while preventing the learning agent from overfitting possible subtleties of an individual implementation.

PHOTOREALISTIC RENDERING Visuomotor control is one of the leading research directions in the field of reinforcement learning applied to robotics. The need for photorealistic rendering is a crucial component in this use case. Modern GPUs are becoming more capable of efficiently computing extremely complex light interactions in the simulated environment, and technologies such as ray tracing are becoming suitable for real-time usage.

5.1.2 Existing frameworks

OPENAI ROBOTIC ENVIRONMENTS [20] are part of the official OpenAI environments, which became the standard solution commonly used to benchmark algorithms. They are simulated with the Mujoco simulator [E. Todorov et al., 2012], which became one of the most common solutions for continuous control tasks. The simulator used to be proprietary, a constraint greatly limiting its use. At the time of writing, however, open-sourcing activities are ongoing.

PYBULLET ENVIRONMENTS [Coumans et al., 2016] are part of the Bullet3 project and use Bullet as a physics engine. Given the project's active development and open-source nature, a big community revolves around this physics engine. Simulations are reliable and fast, but the default rendering capabilities are not photorealistic. The provided robotic environments are complete, even if their documentation and modularity can be improved.

UNITY ML AGENT [Juliani et al., 2018] is another promising toolkit for creating environments based on the Unity platform. It supports Nvidia PhysX out-of-the-box, and plugins exist for Bullet and Mujoco. Being based on a gaming engine, rendering is very photorealistic. Despite agent code and physics engine residing on different processes, the selected gRPC communication protocol in its synchronous variant ensures determinism. However, custom actions and observations require the ad-hoc development of the data serialisation between client and server.

RAISIM [J. Hwangbo et al., 2018] is a recent simulator specific for robotics. Its main advantage is an efficient contact solver that greatly speeds up the simulation. Due to its very recent release, there are not many examples available. Like other frameworks, its closed-source nature might limit applications. It includes the Python framework RaisimGymTorch[21] that allows creating RL environment.

20 https://openai.com/blog/ingredients-for-robotics-research
21 https://raisim.com/sections/RaisimGymTorch.html

PYROBOLEARN [Delhaisse et al., 2019] is another framework containing both robotic environments and RL algorithms. It focuses on modularity and flexibility to promote code reuse. It currently supports only PyBullet, though it already has a physics engine abstraction layer in Python that will simplify adding other back-ends. A current limitation is missing support to transfer code from simulation to real robots.

JIMINY [22] is a simulator for poly-articulated systems based on the Pinocchio library. It supports advanced simulation features like motor inertia, sensor noise, and delays. The simulator includes Gym-Jiminy, a Python package offering convenience tools for learning, and a GUI based on the Meshcat library that does not provide photorealistic rendering.

PYREP [James et al., 2019] is a toolkit for robot learning research based on Coppelia Sim, formerly known as V-Rep. The toolkit is able to efficiently run parallel simulations thanks to custom modifications that replaced inter-thread communications between instances. Different renderers are available, whose frame rate depends on the desired photorealism.

ROBO-GYM [Lucchi et al., 2020] is an open-source toolkit for distributed reinforcement learning on real and simulated robots. It provides a collection of RL environments involving robotic tasks applicable to both simulation and real-world. Additionally, it provides the tools to facilitate the creation of new environments featuring different robots and sensors. The architecture is based on the ROS middleware, therefore all simulators implementing the ROS interfaces can be integrated seamlessly. As a drawback, the usage of network transport does not guarantee reproducibility.

NVIDIA ISAAC GYM [Makoviychuk et al., 2021] is the RL component of Isaac, the new Nvidia toolbox for AI applications in robotics. Simulations can be executed in their PhysX engine and they provide state-of-the-art photorealistic rendering. Nvidia Isaac[23] is one of the most promising projects that will provide a unified framework for robotics and AI, but

22 https://github.com/duburcqa/jiminy
23 https://developer.nvidia.com/isaac-sdk

unfortunately its closed source nature might limit the possibility of extending and customising it.

BRAX [Freeman et al., 2021] is a differentiable physics engine that simulates rigid bodies in maximal coordinates. It is written in JAX and is designed for use on acceleration hardware, enabling massively parallel simulations on multiple devices. The physics engine, beyond the drawbacks in joint constraints enforcement due to the maximal coordinates, neglects some dynamical effects and therefore the simulation is approximate. Rendering capabilities are limited and the execution of environments provided in the framework is constrained to a simulated setting.

Table 5.1: Comparison of frameworks that provide robotic environments compatible with OpenAI Gym.

Software	Reproducible	Multiple Physics Engines	Photorealistic Rendering	Accelerated	Parallel	Real-Time Compatible	Modular	Open Source
OpenAI Robotic Envs				✓	✓		✓	✓
Bullet3 Environments	✓	✓		✓	✓		∼	✓
Unity ML-Agents	✓	✓	✓	✓	✓			✓
RaiSim	✓		✓	✓	✓			
Jiminy	✓			✓	✓			✓
PyRep	✓	✓	✓	✓	✓			✓
robo-gym		✓		✓		✓		✓
Nvidia Isaac Gym	✓	✓	✓	✓	✓	✓		
Brax	✓			✓	✓			✓
Gym-Ignition	✓	✓	∼	✓	✓	✓	✓	✓

5.2 DESIGN GOALS

In this section, we outline the design goals we want to achieve with the proposed framework, acknowledging this phase as one of the most fundamental in software design.

SELECTABLE RUNTIME The major goal of the framework is the possibility to develop agnostic decision-making logic that can run in different settings. The software architecture component that defines the setting and, therefore, how the decision-making logic communicates with the controlled robots is called `Runtime`. The runtime could either implement the stepping logic of any simulator or a routine that enforces the environment to be executed with soft real-time guarantees.

UNIFIED SCENE INTERFACES In the low level, the framework should provide an abstraction layer of robotics scenes so that the decision-making logic can operate seamlessly on unified APIs agnostic from the runtime setting. A commonly used pattern is to provide specialised interfaces such as `World`, `Model`, `Link`, `Joint`, etc. such that models can be gathered from the world, and links and joints from the model, providing specialised functionalities.

IMPLEMENTATION-AGNOSTIC DECISION-MAKING TASKS Another primary goal of the framework is preventing code duplication of decision-making logic. We refer to this logic as `Task`, which includes the operations to perform when the environment is initialised and stepped, and when actions are taken. The task should be implemented with the unified scene interface to become agnostic of the runtime. Therefore, it should be able to run on any implementation, whether it is a simulated or real robot. This architecture enables settings where policies are first trained in simulation and then executed on physical robots, without excluding the use case where the real-world setting is also initially used to sample real data to perform a refinement step of the policy.

ROBOTICS TOOLING The development of robotic environments often requires the computation of kinematics and dynamics model-based quantities,

such as Jacobians, inverse kinematics, total momentum, etc. The framework should integrate resources for their computation accessible from the decision-making logic.

GYM COMPATIBILITY The resulting environment should expose the `gym.Env` interface to be seamlessly compatible with the majority of the frameworks providing RL agents.

Having specified the design goals and the related considerations, we structured the framework in two different components: *Scenario* and *Gym-Ignition*. Considering that most of the robotics libraries and simulators are available in C++, we designed the low-level unified scene interfaces in this language. Scenario (SCENe interfAces for Robot Input/Output) defines the `World`, `Model`, `Link`, and `Joint` abstractions, and allows to implement them either in C++ or, through a set of bindings, in Python. Instead, the most popular language for the RL logic is Python. Gym-Ignition is a pure Python package that provides the `Runtime` and `Task` interfaces, together with high-level helpers to compute model-based quantities based on the iDynTree [Nori et al., 2015] library.

5.3 SCENARIO: SCENE INTERFACES FOR ROBOT INPUT/OUTPUT

Scenario[24] is a C++ library acting as a Hardware Abstraction Layer (HAL) over either simulated or real robots. The abstraction of the scene is structured in different interfaces:

`World` The world interface is the entry point of the entire scene. It is returned directly from the active `Runtime` and allows to query, insert, get, and remove objects part of the scene, including robots.

`Model` A model is an entity part of the scene. It could be, for example, a static object or a robot interacting with the scene. The `Model` interface operates on the entire multibody system. It allows to inspect link and joint properties, get and set base link data, and perform vectorised calls

24 https://github.com/robotology/gym-ignition/tree/master/scenario

of operations specific to individual joints and links. In order to simplify fine-grained operations, it returns `Link` and `Joint` objects.

`Link` This interface returns all the inertial and kinematic properties of the rigid body forming the link. It also returns the pose and, in different representations, the velocity and the accelerations of the link frame. Furthermore, it allows querying the location of active contact points with their corresponding 6D contact force.

`Joint` This interface returns all the parameters of the joint, including the number of DoFs, the type, the friction parameters, the position, velocity, force limits, etc. It is also the entry point to get joint variables like position, velocity, and acceleration, and set the corresponding targets used for motion control. The joint interface also controls the actuation type, exposing either a position or velocity PID controller with its parameters.

Beyond these scene interfaces, Scenario also includes the `Controller` interfaces, allowing development of runtime-agnostic whole-body controllers that can be enabled at the `Model` level. They can be used as a replacement for the default low-level PID controllers associated with each of the model's joints.

The abstraction layer provided by Scenario enables to develop either C++ or Python code agnostic of the actual setting where the scene and its robots operate. For both robot control and RL research, we implemented the Scenario interfaces to communicate with the Gazebo Sim simulator, obtaining the *Scenario Gazebo* library. Future development will also bring a real-time implementation for direct applicability to physical robots mediated by a robotic middleware. As we will discuss, Gazebo Sim already provides an abstraction layer over different physics engines. We can take advantage of this existing abstraction because our Scenario Gazebo backend benefits without any additional implementation effort of any new physics engine that will be included in Gazebo Sim in the future. Finally, the aim of the Scenario layer is to provide an additional abstraction to enable the same high-level code to run also on real robots and other simulators. In the robot learning setting, this can be particularly beneficial for *sim-to-real* and *sim-to-sim* applications.

5.3.1 Scenario Gazebo

Scenario Gazebo is a simulation-based backend of the Scenario interfaces. It communicates with the new *Gazebo Sim* simulator, also known in its earlier releases as Ignition Gazebo.

Gazebo Sim is the new generation of the widely used Gazebo Classic simulator, developed by Open Robotics[25]. It was used in the new DARPA SubT Challenge[26] for both local and cloud simulations. The monolithic architecture of Gazebo Classic has been split into a suite of multiple libraries, with Gazebo Sim being only one among them, and refactored with a more pervasive plugin-based architecture. For our target applications, we selected Gazebo Sim as our main simulation backend due to the following two key features:

PHYSICS ENGINE PLUGINS In Gazebo Sim, physics engines are plugin libraries loaded during runtime. Compared to the previous monolithic architecture, the usage of plugins enables the independent development of the physics library. It allows third-party developers to implement or integrate new engines relatively easily, interfacing them with the rest of the robotics suite without additional effort.

REPRODUCIBILITY Gazebo Sim offers high-level C++ APIs to initialise and control the simulation, contrary to its previous generation and many other robotics simulators. This feature allows running the simulation in the same process of the caller, without relying on any network transport to exchange data. Being able to read, write, and step the simulator programmatically ensures the complete reproducibility of the execution, regardless of the system's load and other variable factors.

These two features are particularly beneficial in the robot learning setting. The plugin-based architecture of physics engines allows running simulated environments transparently in any of the supported physics backends. It enables to add in the domain randomization set not only physics parameters, but also the entire physics engine, preventing policy overfitting to subtle implementation details. Furthermore, ensuring reproducibility already at the

25 https://www.openrobotics.org/
26 https://www.subtchallenge.com/

simulation level is a necessary condition to inherit this feature by simulated RL environments. Other key features of Gazebo Sim are the following:

- Simulator developed for robotics;
- Architecture enabling the simulator-as-a-library usage;
- Plugin-based abstraction also of rendering engines;
- Support of many among the most used sensors like RGB, depth, and segmentation cameras, IMUs, force-torque sensors, lidars, etc.;
- Modular software architecture of the entire suite;
- Support of the standardised and actively developed SDF description for defining models and scenes;
- Well maintained, packaged, and widely tested;
- Integration with Fuel[27], a new large database of simulated objects, robots, and worlds ready to be downloaded and used;
- Long-term support provided.

Beyond implementing the Scenario interfaces, Scenario Gazebo also provides a `GazeboSimulator` resource to initialize the simulator, prepare the simulated scene, and retrieve the `World`. Furthermore, the controllers implemented with the Scenario interfaces can be executed in the inner simulation loop, enabling a more efficient stepping strategy where a single step from the high-level application triggers a sequence of temporally fine-grained steps composed by a closed-loop controller over the physics system. One can compare Scenario Gazebo to other popular alternatives that provide programmatic APIs like pybullet [Coumans et al., 2016] and mujoco-py[28].

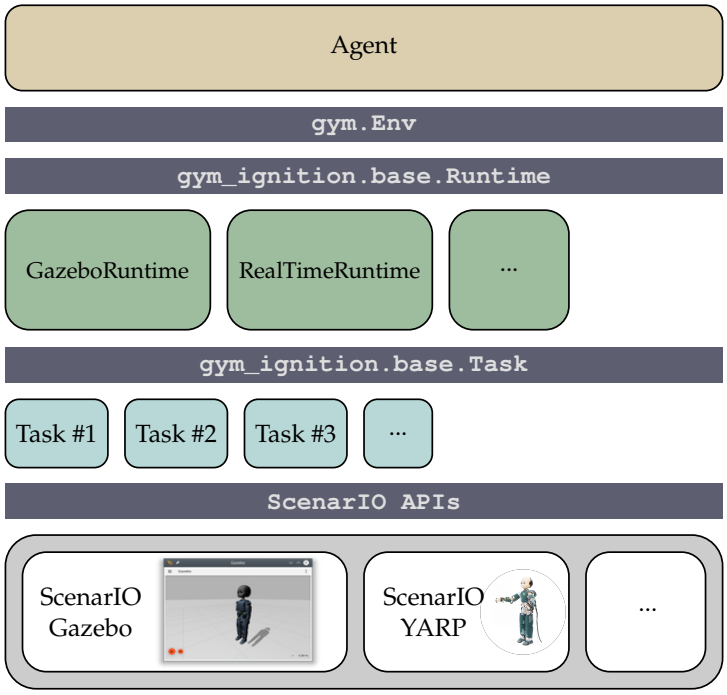


Figure 5.1: Architecture of Scenario and Gym-Ignition. Users of the overall framework just need to provide the URDF or SDF description of their robot and implement the `Task` interface with the desired decision-making logic. The framework, following a top-down approach, exposes to the `Agent` algorithms the unified `gym.Env` interface. The provided `Runtime` classes either instantiate the simulator, or handle soft real-time logic for real-world robots. The runtimes are generic and can operate on any decision-making logic that exposes the `Task` interface. Finally, `Task` implementations use the Scenario APIs to interact with the robots part of the environment.
A typical data flow starts with the agent setting the action with `gym.Env.step`. The processing of the action is a combination of logic inside the active runtime and the active task. In particular, the runtime receives the action and directly forwards it to the task for being processed. The task, by operating transparently over the Scenario APIs, applies the action to the robot, and then waits the runtime to perform the time stepping. After this phase, the task computes the reward, packs the observation, detects if the environment reached the terminal state, and returns all this data back to the agent passing through the `gym.Env` APIs.

5.4 GYM-IGNITION

Gym-Ignition[29] is a Python package providing resources to develop robotic environments quickly. It provides boilerplate code that helps minimising the duplication that often occurs in this domain, allowing environment developers to focus on the decision-making logic rather than glue code. The main components of its architecture are the following, also illustrated in Figure 5.1:

`Runtime` This interface inherits directly from `gym.Env`, therefore concrete `Runtime` implementations are the actual environments passed to the agents. In the proposed architecture, our framework provides implemented runtimes for all the supported backends, and allows third-party developers to add their own. The logic included in the runtimes is part of the boilerplate code (e.g. the stepping logic in simulated environments) always present in all environments sharing the same backend. All runtimes operate on a generic `Task` abstraction that, when implemented by the environment developer, defines the desired decision-making logic.

`Task` This interface defines the decision-making logic of the environment. It determines how to process the action received from the agent, how to create the observation sample, and how to calculate the scalar reward. It also evaluates if the environment reached its terminal state and, when it occurs, it includes the logic to obtain the initial state of the new episode. The task has access to the complete state of the scene by operating directly on the `World`, therefore, in partially observable problems, it is also responsible for exposing only the desired information to the agent. The architecture also supports the multi-agent setting, since individual tasks do not trigger the time evolution of the scene, but can read its state and set actions processed by a unique runtime.

RANDOMIZERS During the training process, when the environment is undergoing a reset, the resetting logic could be different whether the runtime is simulated or operating on a real-world scene. Examples of different

27 `https://app.ignitionrobotics.org/fuel`

28 `https://github.com/openai/mujoco-py`. At the time of writing, the Mujoco simulator is in the process of being open-sourced. Future releases of the simulator will include official Python bindings.

29 `https://github.com/robotology/gym-ignition`

behaviours could originate from the domain randomization process, the selection of the initial distribution, mismatch between training and evaluation, etc. In most cases, this logic is not strictly related to the task nor the runtime. For this reason, our framework also introduces the possibility to optionally define *environment wrappers* called `Randomizers` to specify this custom logic.

MULTIBODY ALGORITHMS In most cases, the development of an environment requires the computation of accessory kinematics and dynamics quantities not directly exposed through ScenarIO. In order to facilitate the development, we interface with the iDynTree [Nori et al., 2015] library that provides a large amount of multibody dynamics algorithms supporting floating-based robots.

In complex software architectures, and particularly regarding simulated environments, ensuring the reproducibility of results is always a delicate matter. In a RL pipeline, the main sources of randomness are either the operations that we allow isolating into the `Randomizer`, or additive noise applied by the `Task` to the received action and produced observation. The generation of random quantities can be controlled by passing through a RNG, producing a deterministic pseudo-random sequence of values from a given seed number. Our framework also helps enforce reproducible results in presence of randomness by storing a single generator in the `Runtime`, shared with all `Tasks` and `Randomizers`. Under the assumption that the environment logic's execution order does not change, multiple experiments sharing the same seed number produce the same sampled trajectories.

5.5 CONCLUSIONS

This chapter presented a novel framework to create reproducible robotic environments for Reinforcement Learning applications. At the low-level, we introduced Scenario, a C++ abstraction layer of a scene where robots can operate and interact. Applications of robot control and RL operating on the Scenario APIs can seamlessly switch between the existing runtime implementations with little effort. We currently provide a simulated backend called Scenario Gazebo, that interfaces with the Gazebo Sim simulator. At the high level, we introduced Gym-Ignition, a Python framework compatible with `gym.Env` to develop robotic environments. It provides another set of abstractions that, complementing those included in Scenario, enable the development of RL environments agnostic of the setting where they are executed. Gym-Ignition helps isolate the generic decision-making logic from the runtime that controls the setting where it runs.

The complete framework aims to narrow the gap between RL and robotic research, allowing roboticists to structure their environments with familiar tools while guaranteeing the reproducibility of simulated results. In the next chapter, we validate the framework proposing a scheme to train a push-recovery policy for balancing the humanoid robot iCub in a simulated setting.

The main lesson of thirty-five years of AI research
is that the hard problems are easy and the easy problems are hard.
— Steven Pinker

6 LEARNING FROM SCRATCH EXPLOITING ROBOT MODELS

In the previous chapter, we proposed a unified framework to develop robotic environments for RL research. The range of possible decision-making tasks that can be applied to robots is broad, from manipulation to locomotion. This chapter considers the task of balancing the humanoid robot iCub in the presence of external disturbances in a simulated setting. Framing the control objective as a RL problem, we aim to train with the PPO algorithm a policy encoded as a NN capable of synthesising the appropriate instantaneous control signals of 23 DoFs of the iCub humanoid robot. The resulting control action, simultaneously operating on both the upper- and lower-body joints, encodes multiple whole-body push-recovery strategies involving the usage of ankles, hips, momentum, and stepping strategies. The policy automatically selects and blends all these different strategies upon need.

The presented architecture adopts a *reward shaping* methodology that exploits as prior information quantities computed from the robot description, such as whole-body momentum and data related to the robot's CoM. This approach allows utilising the widely used family of model-free RL algorithms while exploiting priors without changing the overall learning framework. The priors act only as exploration hints, therefore, the model description does not need to be excessively accurate. We also apply domain randomization over the model description used in simulation, resulting in differences between the simulated robot's dynamics and the model's dynamic from which the reward-shaping priors are computed.

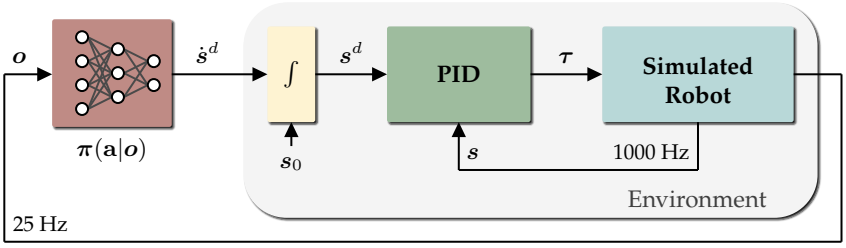


Figure 6.1: The proposed control system.

6.1 TRAINING ENVIRONMENT

The decision-making logic is structured as a continuous control task with early termination conditions. Its dynamics runs in the Gazebo Sim simulator embedded into the Gym-Ignition framework presented in Section 5.4, exposing a RL environment compatible with `gym.Env` [Brockman et al., 2016]. The enabled physics engine is DART [Jeongseok Lee et al., 2018]. We selected iDynTree [Nori et al., 2015] for calculating rigid-body dynamics quantities that model the floating-base multibody system as described in Section 2.8.4, whose dynamics is described by the following equation:

$$M(\mathbf{q})\,\dot{\boldsymbol{\nu}} + h(\mathbf{q}, \boldsymbol{\nu}) = B\boldsymbol{\tau} + \sum_{L \in \mathcal{L}} J_{W,L}^{\top}(\mathbf{q})\,\mathbf{f}_L^{ext}. \tag{6.1}$$

The iDynTree library allows to load a model description encoded in URDF and provides all the terms forming this Lagrangian representation of the floating-base EoMs together with additional quantities that will be included in the reward computation described in Section 6.3.

Figure 6.1 reports a high-level overview of the proposed control system. The environment receives actions from the agent, here represented as its underlying NN, and produces observations and rewards at 25 Hz. The physics and the low-level PID controllers run at 1000 Hz. During training, some properties of the environment are randomised (see Sec. 6.1.3).

Table 6.1: Observation components.

Name	Value	Set	Range
Joint positions	$\mathbf{o}_s = \boldsymbol{s}$	$\mathbb{R}^n$	$[\boldsymbol{s}_{lb}, \boldsymbol{s}_{ub}]$
Joint velocities	$\mathbf{o}_{\dot{s}} = \dot{\boldsymbol{s}}$	$\mathbb{R}^n$	$[-\pi, \pi]$
Base height	$\mathbf{o}_h = \boldsymbol{p}_B^z$	$\mathbb{R}$	$[0, 0.78]$
Base orientation	$\mathbf{o}_R = (\rho, \phi)_B$	$\mathbb{R}^2$	$[-2\pi, 2\pi]$
Contact configuration	$\mathbf{o}_c = (c_L, c_R)$	$\{0, 1\}^2$	-
CoP forces	$\mathbf{o}_f = (f_L^{CoP}, f_R^{CoP})$	$\mathbb{R}^2$	$[0, mg]$
Feet positions	$\mathbf{o}_F = ({}^B\boldsymbol{p}_L, {}^B\boldsymbol{p}_R)$	$\mathbb{R}^6$	$[0, 0.78]$
CoM velocity	$\mathbf{o}_v = {}^G\boldsymbol{v}_{CoM}$	$\mathbb{R}^3$	$[0, 3]$

6.1.1 Action

The separation between agent and environment is defined by the action selection. In our nested structure, the policy generates an action $\mathbf{a} \in \mathbb{R}^{23}$ composed of the reference velocities for a large subset of the robot joints (controlled joints), which are then integrated and fed to the corresponding PID position controllers. The controlled joints belong to the legs, torso, and arms. Hands, wrists, and neck, which arguably play a minor role in balancing, are locked in their natural positions. The policy computes target joint velocities bounded in $[-180, 180]$ deg/s at 25 Hz. Commanding joint velocities rather than joint positions prevents target joint positions from being too distant from each other in consecutive steps. Especially at training onset, this would lead to jumpy references that the PID controllers cannot track, affecting the discovery of the relation between the states $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$. The integration process, instead, enables to use a policy that generates discontinuous actions while maintaining continuous PID inputs with no need for additional filters.

6.1.2 State

Since no perception is involved, the state of the MDP contains information about the robot's kinematics and dynamics. It is defined as the tuple $\mathbf{x} := \langle \mathbf{q}, \boldsymbol{\nu}, \mathbf{f}_L, \mathbf{f}_R \rangle \in \mathcal{X}$, where $(\mathbf{f}_L, \mathbf{f}_R)$ are the 6D forces exchanged between the feet and the terrain. The observation, computed from the state $\mathbf{x}$ and the robot model $\mathcal{M}$, is defined as the tuple

$$\boldsymbol{o} = \boldsymbol{o}(\mathbf{x}, \mathcal{M}) := \langle \mathbf{o}_s, \mathbf{o}_{\dot{s}}, \mathbf{o}_h, \mathbf{o}_R, \mathbf{o}_c, \mathbf{o}_f, \mathbf{o}_F, \mathbf{o}_v \rangle \in \mathcal{O}$$

where $\mathcal{O} := \mathbb{R}^{62}$. The observation consists of the following terms, each of them re-scaled[30] into a given interval for better properties when used as NN inputs:

- $\mathbf{o}_s$ are the controlled joints angles in radians, normalised with the hard limits defined in the model description;
- $\mathbf{o}_{\dot{s}}$ are the velocities of the controlled joints, normalised in $[-\pi, \pi]$ rad/s;
- $\mathbf{o}_h$ is the height of the base frame, normalised in $[0, 0.78]$ m;
- $\mathbf{o}_R$ is a tuple containing the roll and pitch angles of the base frame w.r.t. the world frame, normalised in $[-2\pi, 2\pi]$ rad;
- $\mathbf{o}_c$ is a tuple defining whether the feet are in contact with the ground;
- $\mathbf{o}_f$ is a tuple containing the vertical forces applied to the local CoP of the feet (see Appendix A for its definition), normalised in $[0, 330]$ N, i.e. the nominal weight force of the robot;
- $\mathbf{o}_F$ is a tuple containing the positions of the feet w.r.t. the base frame, normalised in $[0, 0.78]$ m;
- $\mathbf{o}_v$ is the linear velocity of the CoM expressed in the CoM frame $G = (\mathbf{p}_{CoM}, [B])$, normalised in $[0, 3]$ m/s.

The exact definition of all the observation terms is reported in Table 6.1.

Although the agent is trained in simulation, we design it for real-time execution on actual robots. We carefully select state components that can be either measured or estimated on-board [Nori et al., 2015]. To promote policy transfer, we avoid measurements from noisy sensors and values that cannot be estimated with sufficient accuracy. In fact, any significant mismatch between simulated and real data would hinder transfer, increasing the reliance on policy robustness. We select minimal state components encoding the environment dynamics without affecting learning performance.

30 We also call the re-scaling operation of a bounded variable as *normalization*.

6.1.3 Other specifications

INITIAL STATE DISTRIBUTION The initial state distribution $\rho(\mathbf{x}_0) : \mathcal{X} \to \mathcal{O}$ defines the value of the observation in which the agent begins each episode. Sampling the initial state from a distribution with small variance, particularly regarding joint positions and velocities, positively affects exploration without degrading the learning performance. At the beginning of each episode, for each joint $j \in \mathcal{J}$ we sample its position $s_{j,0}$ from $\mathcal{N}(\mu = s_0, \sigma = 10\,\text{deg})$, where s_0 represents the fixed initial reference, and its velocity $\dot{s}_{j,0}$ from $\mathcal{N}(\mu = 0, \sigma = 90\,\text{deg/s})$. As a result, the robot may or may not start with the feet in contact with the ground, encouraging the agent to learn how to land and deal with impacts.

EXPLORATION To promote exploration beyond the initial state distribution and favour the emergence of push-recovery strategies, we apply external perturbations in the form of a 3D force to the robot's base frame located over its pelvis. The applied force vector has a fixed magnitude of 200 N and is applied for 200 ms. Considering the weight of the iCub, approximately 33 kg, the normalised impulse sums up to 1.21 Ns/Kg. We sample the direction of the applied force from a uniform spherical distribution. The frequency of the application is defined as average applications per second, again sampling from a uniform distribution. We apply a force on average every 5 simulated seconds.

EARLY TERMINATION The balancing and push-recovery objectives for a continuous-control task are characterised by an infinite-horizon discounted MDP. During training, however, episodes should stop as soon as the state reaches a subspace from which either it is impossible to recover or uninteresting to explore, following an early-termination criterion. The state space interesting for our work is where the robot is – almost – standing on its feet, therefore we terminate the episodes as soon as it falls to the ground. We detect the falling condition when any link but the feet touches the ground plane.

DOMAIN RANDOMISATION During the training process, the environment performs a domain randomisation step at the beginning of each new episode.

Table 6.2: PPO, policy, and training parameters.

Parameter	Value
Discount rate γ	0.95
Clip parameter ϵ	0.3
Learning rate α	0.0001
GAE parameter λ	1.0
Batch size	10000
Minibatch size	512
Number of SGD epochs	32
Number or parallel workers	32
Value function clip parameter	1000

The masses of the robot's links are sampled from a normal distribution $\mathcal{N}(\mu = m_0, \sigma = 0.2m_0)$, where m_0 is the nominal mass of the link defined in the model description. To avoid making assumptions about the feet's and ground's material properties, we randomise the Coulomb friction μ_c of the feet by sampling it from $\mathcal{U}(0.5, 3)$. Finally, since the simulation does not include the real dynamics of the actuators, to increase robustness, we apply a delay to the position references fed to the PID controllers, sampled from $\mathcal{U}(0, 20)$ ms, and kept constant during the entire episode (until termination).

6.2 AGENT

The agent receives the observation $\boldsymbol{o}$ from the environment and returns the action $\mathbf{a}$ defining the reference velocities of the controlled joints. The agent parameters are reported in Table 6.2 and further explained below.

LEARNING ALGORITHM We select PPO as the candidate learning algorithm, in the variant that includes both the clipped surrogate and KL penalty objectives introduced in Section 3.4.3. We selected this algorithm since it provides a simple but effective implementation of policy optimization, widely used in comparable studies[31]. A practical benefit of PPO is its small number of parameters (only

31 This chapter is mainly focused on the reward shaping process, not on the specific algorithm used to train the agent.

the clip parameter ϵ if the KL penalty coefficient is adjusted dynamically) that does not excessively overload the parameter tuning process.

POLICY AND VALUE FUNCTION The policy, given an observation $\boldsymbol{o}_t$, samples the action $\boldsymbol{a}_t$ to take from the stochastic distribution $\pi(\cdot \mid \boldsymbol{o}_t)$. The value function $\hat{V}(\boldsymbol{o}_t)$, instead, estimates the average return when starting from the observation $\boldsymbol{o}_t$ and then following the policy for the next steps. We represent both the policy and the value function with two different neural networks composed of two fully connected layers, with 512 and 128 units each, followed by a linear output layer. The hidden units use a ReLU activation function. The networks do not share any layer.

DISTRIBUTED SETUP The chosen PPO algorithm scales gracefully to a setup where the batch samples are collected from multiple workers in parallel. A single trainer and 32 workers with an independent copy of the environment form our training setup. After collecting a batch of 10000 on-policy transitions, we train the neural networks with stochastic gradient descent. The optimiser uses mini-batches containing 512 samples and performs 32 epochs per batch. The learning rate is $\lambda = 0.0001$. Each trial is stopped once it reaches 20 M agent steps, roughly equivalent to 7 days of experience on a real robot. Worker nodes run only on CPU resources, while the trainer has access to the GPU for accelerating the optimisation process. We use the RLlib [E. Liang et al., 2018] framework, OpenAI Gym, and distributed training.

6.3 REWARD SHAPING

6.3.1 RBF Kernel

Radial Basis Function (RBF) kernels are widely employed functions in machine learning, defined as

$$K(\mathbf{x}, \mathbf{x}^*) = \exp\left(-\tilde{\gamma}||\mathbf{x} - \mathbf{x}^*||^2\right) \quad \in [0, 1],$$

where $\tilde{\gamma}$ is the kernel bandwidth hyperparameter. The RBF kernel measures similarities between input vectors. This can be useful for defining scaled reward components. In particular, if $\mathbf{x}$ is the current measurement and $\mathbf{x}^*$ is the target, the kernel provides a normalised estimate of their similarity. The variable $\tilde{\gamma}$ can be used to tune the bandwidth of the kernel, i.e. its sensitivity. In particular, we use $\tilde{\gamma}$ to select the threshold from which the kernel tails begin to grow. Introducing the pair (x_c, ϵ), with $x_c, \epsilon \in \mathbb{R}^+$ and $|\epsilon| \ll 1$, we can parameterise $\tilde{\gamma} = -\ln(\epsilon)/x_c^2$. This formulation results in the following properties:

1. $K(\mathbf{x}^*, \mathbf{x}^*) = 1$, i.e. when the measurement reaches the target, the kernel outputs 1;
2. Given a measurement $\mathbf{x}_m$ such that $||\mathbf{x}_m - \mathbf{x}^*|| = x_c$, the kernel outputs $K(\mathbf{x}_m, \mathbf{x}^*) = \epsilon$.

In practice, ϵ can be kept constant for each reward component. The sensitivity of individual components is tuned by adjusting x_c. We refer to x_c as *cutoff* value of the kernel, since each norm of the distance in the input space bigger than x_c yields output values smaller than ϵ. This formulation eases the composition of the total reward r_t when its components are calculated from measurements of different dimensionalities and scales. In fact, once the sensitivities have been properly tuned for each component, they can simply be weighted differently as:

$$r_t = \sum_i w_i K\left(\mathbf{x}_i(t), \mathbf{x}_i^*(t)\right) \in \mathbb{R}, \tag{6.2}$$

where $\mathbf{x}_i(t)$ is the i-th measurement sampled at time t, and $w_i \in \mathbb{R}$ the weight corresponding to the i-th reward component.

6.3.2 Reward

This section describes all reward terms forming the instantaneous reward generated by the environment. Given their number, we divided the terms in three categories: *regularisers* are terms often used in optimal control for minimising the control action and the joint torques; *steady-state terms* help obtain the balancing behaviour in the absence of external perturbations, and are active only in Double Support (DS); *transient terms* favour the emergence of push-recovery whole-body strategies.. The logic of the task computes all reward terms at each environment step. Then, all terms are processed with a RBF kernel, weighted, and summed together using Equation (6.2), obtaining the final instantaneous reward r_t returned to the agent for its maximisation during the training phase.

The situation in which all individual reward terms r_i share the same range thanks to the filtering effect of the applied kernel simplifies the process of parameters tuning. Each reward term introduces two parameters: its weight ω_i and the kernel cutoff x_c, where we assumed to keep the ϵ kernel parameter fixed. Instead of applying grid-like search methods, that would require an excessively large number of permutations, we proceeded with an heuristic tuning by observing after each training the learning curve of each reward term independently. Firstly, we tuned the cutoff parameter (i.e. the sensitivity associated to the reward term) so that the agent could receive a perceivable increased reward to promote the desired exploration direction. This process could be thought as tuning the variance of a Gaussian curve having mean over the target value. Secondly, the reward terms with their tuned kernels are composed together by a weighted sum. We started the tuning process from a weight of 1.0 for all reward terms, and increased individual weights heuristically in case we wanted to adjust their relative importance. Table 6.3 reports all the parameters of the reward function.

Regularisers

JOINT TORQUES (r_τ) We compute the norm of all the joint torque references synthesised by the PID controllers from the velocity references provided by the policy, and penalise this value. The environment runs at 25 Hz and the low-level controllers at 1000 Hz. Therefore, for each of the 23 joints, 40 torques are actuated between two consecutive environment steps. We collect all these torques in a single vector $\boldsymbol{\tau}_{step} \in \mathbb{R}^{23 \cdot 40}$ and penalise its norm.

JOINT VELOCITIES ($r_{\dot{s}}$) Our control scheme ensures that joint position references are continuous. However, PPO explores the action space of joint velocities following the active distributions. To promote smoother trajectories, we penalise the norm of the latest action. It can be seen as the minimisation of the control effort.

Steady-state

POSTURAL (r_s) Whole-body humanoid control schemes apply different weights to various control objectives. The postural is notably one of the most used [Nava et al., 2016], although it is usually assigned a low priority. A postural reward term helps to reach a target posture during balancing instead of relying on local minima found in the learning process. This component penalises the mismatch between the sampled joint configuration and the reference configuration shown in Figure 6.4a.

COM PROJECTION (r_G) Statically balanced robots, in order to maintain stability, keep the CoM within the Support Polygon (SP), defined as the Convex Hull (CH) of their contact points with the ground. With the same aim, we introduce a Boolean component rewarding the agent if its CoM ground projection is within the SP induced by the feet. For additional safety, we shrink the SP by a 2.5 cm margin all along its perimeter.

HORIZONTAL COM VELOCITY (r_v^{xy}) We define a target horizontal velocity for the CoM as a vector pointing from the CoM projection to the center of the SP $\bar{\boldsymbol{p}}_{hull}^{xy}$. In order to promote faster motions if the CoM is relatively close to the ground, the magnitude of the target is amplified by a factor $w_0 = \sqrt{g / \boldsymbol{p}_G^z}$ derived from the LIP model [Kajita et al., 2001], where g is the standard gravity. This component encourages the motion of the CoM projection towards the center of the SP.

Transient

FEET IN CONTACT (r_c) The feet are encouraged to stay on the ground. In order to promote steps and increase movement freedom, we add a Boolean term marking whether any foot is in contact with the ground.

LINKS IN CONTACT (r_l) If any link excluding feet is in contact with the ground, the episode terminates with a negative reward of -10 for the terminal state.

WHOLE-BODY MOMENTUM (r_h) Our policy also controls joints belonging to the torso and the arms. Therefore, the momentum generated by the upper body can be exploited for balancing and push recovery. This term minimises the sum of the norms of the linear and angular components of the robot's total centroidal momentum ${}_G\mathbf{h}$ [Traversaro et al., 2017].

FEET CONTACT FORCES (r_f) This reward term pushes the transient towards a steady-state pose in which the vertical forces at feet's CoPs (f_L^{CoP}, f_R^{CoP}) (see Appendix A) assume the value of half of the robot's weight, distributing it equally on the two feet.

FEET COP (r_p) Beyond the force at the feet CoPs, we also promote their positions to be located at the center of the corresponding sole $\bar{\boldsymbol{p}}^{xy}_{foot,hull}$.

VERTICAL COM VELOCITY (r_v^z) This reward component discourages vertical motion of the CoM of the base link, promoting the usage of the horizontal component instead.

FEET ORIENTATION (r_o) In early experiments, the policy was converging towards feet tipping behaviours, i.e. the feet were not in full contact with the ground. Since the terrain is flat by assumption, we discourage tipping by promoting a foot's orientation such that its sole is parallel to the ground. If ${}^W R_{foot} = [\mathbf{r}^{(x)}, \mathbf{r}^{(y)}, \mathbf{r}^{(z)}]$ is the rotation between the foot frame and the world, this term promotes the alignment of its third column with the world frame.

Table 6.3: Reward function details. Terms with a defined cutoff are processed by the RBF kernel.

Name	Symbol(s)	Weight	Value x	Target $\mathbf{x}^*$	Cutoff x_c		SS	DS
Joint torques	r_τ	5	$\|\boldsymbol{\tau}_{step}\|$	$\mathbf{0}_n$	10.0	Nm	✓	✓
Joint velocities	$r_{\dot{s}}$	2	$\boldsymbol{a}$	$\mathbf{0}_n$	1.0	rad/s	✓	✓
Postural	r_s	10	$\boldsymbol{s}$	$\boldsymbol{s}_0$	7.5	deg		✓
CoM z velocity	r_v^z	2	$\boldsymbol{v}_G^{xy}$	0	1.0	m/s	✓	✓
CoM xy velocity	r_v^{xy}	2	$\boldsymbol{v}_G^z$	$\omega_0(\boldsymbol{p}_G^{xy} - \bar{\boldsymbol{p}}_{hull}^{xy})$	0.5	m/s		✓
Feet contact forces	$\{r_f^L, r_f^R\}$	4	$\{f_L^{CoP}, f_R^{CoP}\}$	$mg/2$	$mg/2$	N	✓	✓
Centroidal momentum	r_h	1	$\|{}_G\mathbf{h}_l\|^2 + \|{}_G\mathbf{h}_\omega\|^2$	0	50.0	kg m^2/s	✓	✓
Feet CoPs	$\{r_p^L, r_p^R\}$	20	$\{\boldsymbol{p}_{L,CoP}, \boldsymbol{p}_{R,CoP}\}$	$\{\bar{\boldsymbol{p}}_{L,hull}^{xy}, \bar{\boldsymbol{p}}_{R,hull}^{xy}\}$	0.3	m	✓	✓
Feet orientation	$\{r_o^L, r_o^R\}$	3	$\{\mathbf{r}_L^{(z)} \cdot \mathbf{e}_z, \mathbf{r}_R^{(z)} \cdot \mathbf{e}_z\}$	1	0.01	-	✓	✓
CoM projection	r_G	10	$\boldsymbol{p}_G^{xy}$	in the CH of support polygon	-	-		✓
Feet in contact	r_c	2	$c_L \wedge c_R$	1	-	-	✓	✓
Links in contact	r_l	-10	c_l	0	-	-	✓	✓

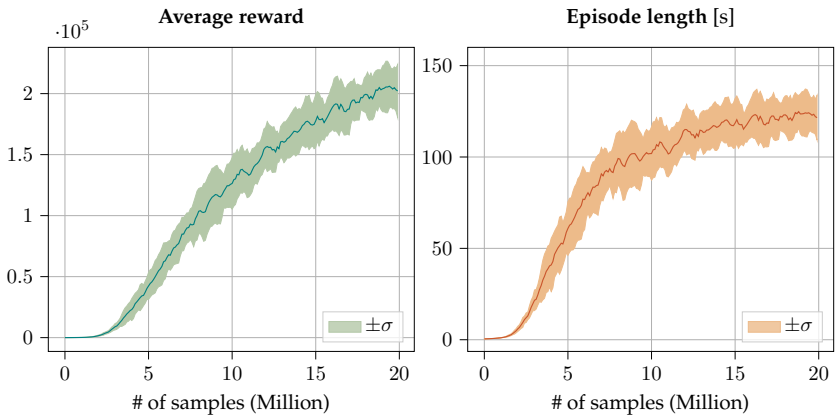


Figure 6.2: Learning curves over 11 training runs.

6.4 RESULTS

6.4.1 Training performance

Fig. 6.2 reports the learning curves of the average reward and episode duration over 11 independent agent training runs. Average reward across trials exhibits consistent growth and low variance (Fig. 6.2, left). We have also observed increasing values for all individual reward elements during training. Episode duration improves as well across trials and displays low variance (see Fig. 6.2, right), approaching maximum episode length more frequently as training progresses.

6.4.2 Emerging behaviours

Controlling the upper body enables rich recovery behaviours that involve the control of the total momentum of the kinematic structure. We succeed in triggering such behaviours by applying external forces during policy training. To make force profiles more realistic, we throw high-speed objects towards the balanced robot instead of applying constant forces for a fixed interval as done during training. Figure 6.4b shows two characteristic sequences. A larger variety of push-recovery strategies are displayed in the following video: `https://ami-iit.github.io/emergence-push-recovery-icub`.

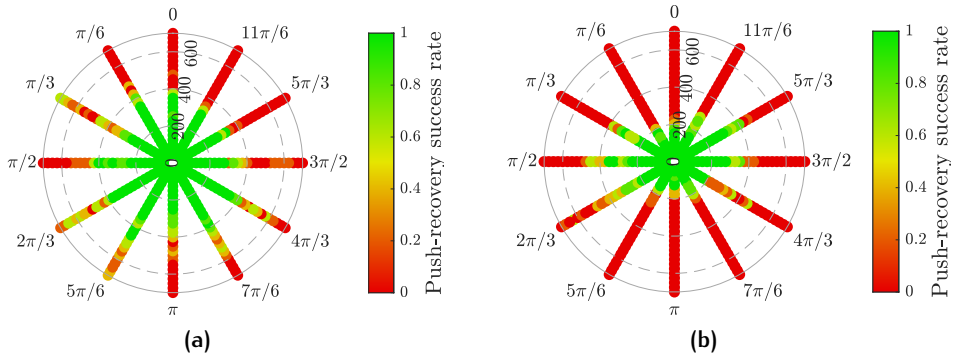


Figure 6.3: (a) Push-recovery success rates on the horizontal plane (forward push: 0 rad, $\mu_c = 1$). (b) Results with $\mu_c = 0.2$.

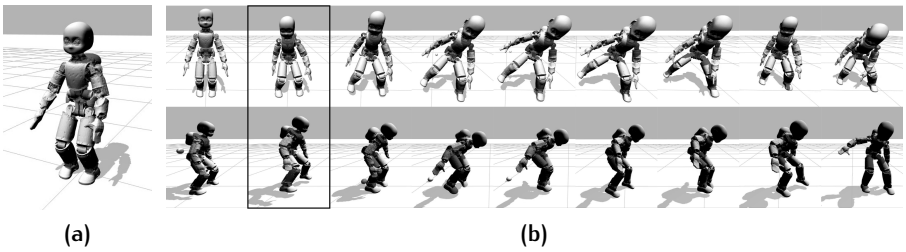


Figure 6.4: (a) The initial joint configuration s_0. (b) Sequences showing ankle, step, and momentum push-recovery strategies. The robot is pushed by a sphere shot from the left side of the image. Impact takes place in the second frame.

6.4.3 Deterministic planar forces

We evaluate the push-recovery performance by assessing the resilience from the application of external forces to the horizontal transverse plane of the robot. Forces are applied for 0.2 s after 3 s from the simulation start, when the robot is stably standing still and front-facing. Success is defined if the robot is still standing after 7 s, defining as a standing state the configuration in which only the feet can be in contact with the ground. Fig. 6.3a reports success rates for forces pointing in 12 directions. Magnitudes increase from 50 N to 700 N at 25 N intervals. Five repetitions are performed for each magnitude and direction, randomising the initial joints configuration by adding zero-mean Gaussian noise ($\sigma = 2$ deg). Magnitudes within the training range (0-200 N) are counteracted successfully. Remarkably, the policy is also robust to out-of-sample forces in all directions in (200-300 N), up to 400 N in some directions. Moreover, it successfully recovers from pushes in the training range (0-200 N) even with an out-of-sample test friction coefficient $\mu_c = 0.2$ (Fig. 6.3b).

6.4.4 Random spherical forces on the base links

We evaluate policy robustness in challenging scenarios involving sequences of random forces with different combinations of magnitude and duration. Forces are applied to the base in a random direction more frequently than during training, on average every 3 s. For each combination, 50 reproducible episodes with different seed initialisation and no domain randomisation are executed. Episodes terminate if the robot falls or after 60 s, averaging 20 applications in a complete episode. Our evaluation metric is the number of consecutive forces endured by the robot. Fig. 6.5 reports aggregate results for each combination of magnitude and duration. No matter their magnitude, forces lasting 0.1 s are appropriately balanced. As expected, performances decrease with growing magnitude and duration. Nevertheless, the agent can withstand repeated applications of out-of-sample forces. For instance, on average, it withstands 9 consecutive 300 N 0.2 s applications.

6.4.5 Random spherical forces on the chest and elbow links

We also evaluate the robustness of the learned policy to previously unseen forces applied to other links. Fig. 6.5 shows the results obtained on the chest and elbow links. As expected, forces applied on links that are far from the CoM turn out to be more challenging. Nevertheless, the policy is able to withstand a good number of them and generalise with good performances. For instance, it is on average able to recover from 10 consecutive 200 N 0.2 s forces 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. Notice that the randomness of the interval between two subsequent forces applications sometimes leads to challenging scenarios in which multiple forces are applied in a short time window.

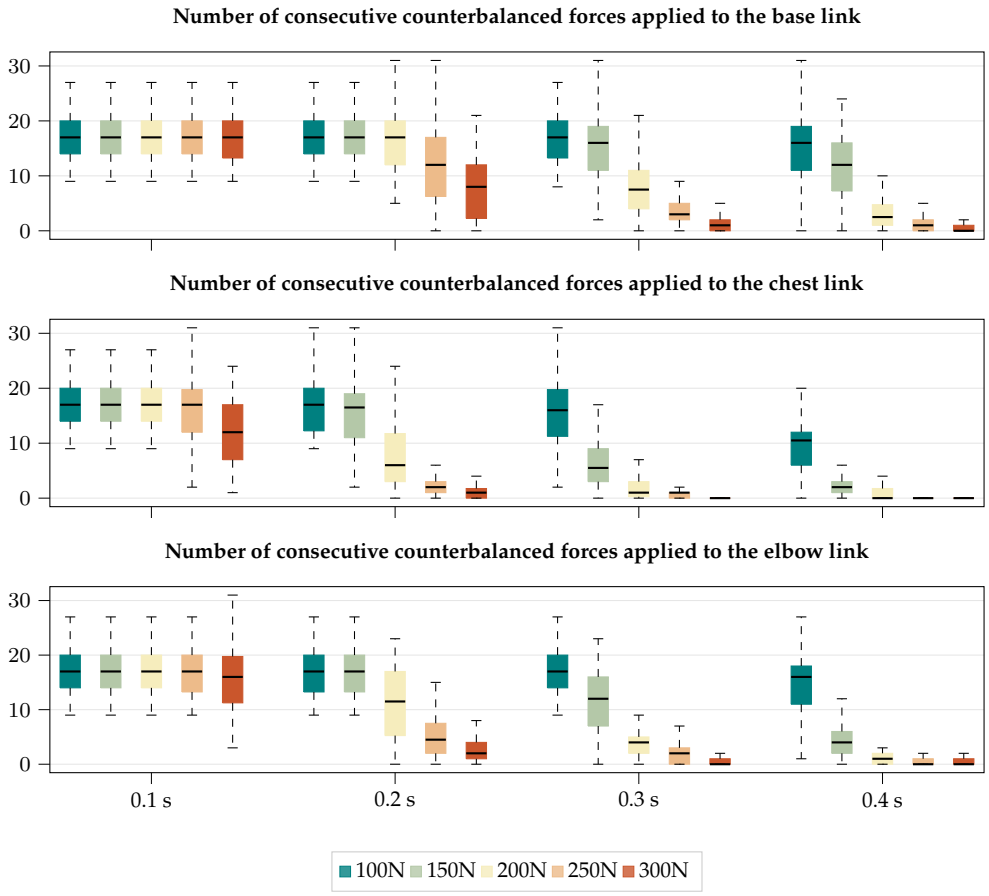


Figure 6.5: Consecutive counterbalanced forces in random directions over 50 trials for each combination of magnitude and duration. Forces are applied to the base, chest, and elbow links for an increasing duration.

6.5 CONCLUSIONS

In this chapter, we presented a control system composed of a high-level policy trained with RL generating joint references actuated by low-level PID controllers. The system operates on the humanoid robot iCub with the aim of providing push-recovery capabilities in the presence of external disturbances. We trained the RL policy with the model-free PPO algorithm. In order to balance the exploration-exploitation trade-off and guide the training towards the desired behaviour, we relied on a careful reward shaping process that integrates into the reward signal returned by the environment multiple terms computed from the robot description acting as a prior. We have shown that the resulting policy, operating on most of the joints of the iCub robot, is able to withstand repeated applications of strong external disturbances. Depending on the magnitude of the disturbance, and the state of the system, the policy adopts different push-recovery strategies that include the usage of ankles, hips, stepping, and momentum.

The learning pipeline described in this chapter presents different limitations and shortcomings. First, the training process to obtain a single policy requires an experience equivalent to approximately 7 simulated days, which is not surprising considering the low sample-efficiency typical of model-free RL algorithms based on policy gradient. Two possible ways to mitigate this problem are either switching to algorithms with better sample-efficiency, or optimising the speed of the experience generation. The execution of the 7 simulated days on a powerful workstation running 32 parallel simulations lasted more than 2 real-world days, resulting in a long and extenuating parameters tuning process. A second limitation comes from the chosen low-level control architecture, composed of independent PID controllers. Despite the benefits of their simplicity, they introduce in the system a stiffness that can prevent the emergence of natural and more human-like motions. Finally, the applicability to real robots has yet to be assessed, since the simulation does not take into account second-order dynamic effects characterising real systems. We tried to increase the robustness of the resulting policy by introducing multiple domain randomisation effects, but their effectiveness in the real world has not been assessed.

7 CONTACT-AWARE MULTIBODY DYNAMICS

In the first part of this thesis contributions, we proposed a general framework for creating robotic environments and, with it, introduced a scheme for training a policy for generating push-recovery control signals balancing the iCub humanoid robot in the presence of external disturbances. The simulations were performed using the general-purpose Gazebo Sim, which provides a rich ecosystem supporting many types of robots, sensors, physics engines, and rendering capabilities. However, the benefits of exploiting general-purpose solutions often introduce a trade-off with the achievable performance. As we have experienced with the push-recovery policy presented in Chapter 6, a single iteration of policy training could last multiple days, resulting in a long tuning process that could limit the search space. The bottleneck of the training pipeline are the computations performed by the rigid-body simulator, limiting the rate at which new trajectories can be sampled.

In the continuation of this thesis, we attempt to reply to the question: *"How can we optimise sampling performance of synthetic data?"*. In this chapter, we derive the state-space representation of a floating-base multibody system interacting with a known ground surface. Assuming the knowledge of the terrain height at any point in space and the smoothness of the terrain surface, this formulation of the dynamics enables calculating the robot's trajectory using a plain numerical integration scheme, regardless of the contact state. To this end, we introduce a continuous soft-contacts model for resolving points-surface collisions supporting both static (sticking) and dynamic (slipping) contacts. Each contact point's dynamics state is structured such that it can be included in the state-space representation and integrated with the robot's dynamics. The resulting representation will be used, in the next chapter, as a base for a novel physics engine targeted to exploit modern hardware accelerators for maximising trajectory sampling.

7.1 NOTATION

7.1.1 Frame kinematics with quaternions

In Chapter 2, we introduced the homogeneous transformation ${}^{A}\mathbf{H}_{B} \in \mathrm{SE}(3)$ to describe the pose of a frame (corresponding also to a rigid body). An alternative representation of the SE(3) group is the tuple $({}^{A}\mathbf{p}_{B}, {}^{A}\bar{q}_{B})$, where ${}^{A}\mathbf{p}_{B} \in \mathbb{R}^3$ is the *position vector* and ${}^{A}\bar{q}_{B}$ a *unit quaternion*. A generic quaternion can be defined as follows:

$$q \in \mathbb{H} := \left\{ q_w + \hat{i}q_x + \hat{j}q_y + \hat{k}q_z \ : \ \ q_w, q_x, q_y, q_z \in \mathbb{R} \right\},$$

where $\mathrm{Re}(q) = q_w$ is the *real* part of the quaternion and $\mathrm{Im}(q) = \hat{i}q_x + \hat{j}q_y + \hat{k}q_z$ is the *imaginary* part. Elements of SO(3) describing frame rotations can be described by *unit quaternions*:

$$\bar{q} \in \mathrm{Spin}(3) := \{ q \in \mathbb{H} \ : \ |q| = 1 \}.$$

A quaternion could be also described with a real vector of its coefficients:

$$\mathtt{Q} = (w, \mathbf{r}) \in \mathbb{R}^4,$$

where $w = q_w \in \mathbb{R}$ and $\mathbf{r} = (q_x, q_y, q_z) \in \mathbb{R}^3$. In this chapter, we mainly use this last representation since it has a direct relation with the practical usage in algorithms design. The pose of a generic frame can therefore be described by a vector $({}^{A}\mathbf{p}_{B}, {}^{A}\mathtt{Q}_{B}) \in \mathbb{R}^7$. For what regards the frame velocity, we can keep using any of the 6D vectors introduced in Section 2.3, i.e. $\mathbf{v}_{A,B} = (\boldsymbol{v}_{A,B}, \boldsymbol{\omega}_{A,B}) \in \mathbb{R}^6$.

7.1.2 Frame pose derivative and 6D velocity with quaternions

When the orientation of a frame is expressed with a quaternion, it's worth introducing the relation between the derivative of the pose $({}^{A}\dot{\mathbf{p}}_{B}, {}^{A}\dot{\mathtt{Q}}_{B}) \in \mathbb{R}^7$ and the 6D velocity $\mathbf{v}_{A,B} \in \mathbb{R}^6$. In fact, particularly in this case with the quaternion,

it's clear that the two cannot be related by a simple numerical differentiation since they also have different dimensions. The following relations hold:

$$\begin{cases} {}^{A}\dot{\mathbf{p}}_{B} = {}^{A[B]}\boldsymbol{v}_{A,B} \\ {}^{A}\dot{\mathtt{Q}}_{B} = \frac{1}{2} S({}^{B}\boldsymbol{\omega}_{A,B})\, {}^{A}\mathtt{Q}_{B} \end{cases}, \tag{7.1}$$

where we notice that the derivative of the position is the linear part of the mixed velocity defined in Equation (2.7) (denoted as ${}^{A}\dot{\mathbf{o}}_{B}$), and defined the derivative of the quaternion [Sola, 2017] by introducing the following matrix:

$$\mathrm{S}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\boldsymbol{\omega}^{\top} \\ \boldsymbol{\omega} & -\boldsymbol{\omega}^{\wedge} \end{bmatrix}.$$

The relations of Equation (7.1) can be used for integrating numerically the frame pose, using one of the schemes that will be introduced in Section 7.4. The quaternion $\mathtt{Q}$, in this setting, has to be treated with care, because only unit quaternions can describe rotations.

Numerical integration schemes do not enforce this property is maintained along the trajectory, and numeric approximations could lead to unwanted instabilities. This problem can be either solved or mitigated by different types of solutions. The most straightforward solution is normalising the quaternion after each integration step. Alternatively, a correction term orthogonal to the quaternion dynamics of the following form can be introduced:

$$\dot{\mathtt{Q}} = \frac{1}{2}\,\mathrm{S}\left({}^{B}\mathbf{R}_{W}\,{}^{W}\boldsymbol{\omega}_{W,B}\right)\mathtt{Q} + \frac{1}{2} K_{\mathtt{Q}}\,\mathtt{Q}\left(\|\mathtt{Q}\|^{-1} - 1\right),$$

that corresponds to a Baumgarte stabilization on $\mathrm{SO}(3)$ [Gros et al., 2015], where $K_{\mathtt{Q}} \in \mathbb{R}$ is the correction coefficient. This stabilization term progressively projects the quaternion towards a unity quaternion, restoring the norm in case of drifting. In the continuation of this thesis, we do not explicitly include any correction, considering its choice an implementation detail.

Remark 7.1.1 (Geometric integration of quaternions)**.** Quaternions, like other representations of elements of $\mathrm{SO}(3)$, have strong geometrical properties based on the underlying symmetries. These properties can be exploited to obtain more rigorous integration schemes performed directly on the manifold [Andrle et al.,

2013]. The quaternion relation of Equation (7.1) can be seen as a differential equation in ${}^{A}\mathtt{Q}_{B}$. By exploiting properties of the matrix exponential together with quaternion properties, it can be shown [Andrle et al., 2013; Sola, 2017; Solà et al., 2020] that the following relation implements a zero-*th* order integration:

$$\mathtt{Q}_{t_{k+1}} = \mathtt{Q}_{t_k} \otimes \Big(\|\boldsymbol{\omega}\| \cos(\boldsymbol{\omega}\Delta t/2),\ \sin(\boldsymbol{\omega}\Delta t/2) \Big)$$

where $\otimes$ denotes the quaternion multiplication, and $\boldsymbol{\omega}$ is assumed constant within the integration interval $[t_k, t_{k+1}]$.

While both integration approaches are valid options –albeit showing different numerical stability [Andrle et al., 2013]–, we develop the theory of this chapter using the numerical derivative. This choice allows treating the entire state of a multibody system, including the base quantities, as a vector in $\mathbb{R}^n$, simplifying the equations of this chapter. If needed, practical implementations of the presented results could adopt the geometrical integration if better numerical stability is required, paying the price to treat the base orientation separately. □

7.1.3 Multibody dynamics

The EoMs of a floating-base multibody system have been previously introduced in Section 2.8.4 in the following form:

$$M(\mathbf{q})\,\dot{\boldsymbol{\nu}} + C(\mathbf{q}, \boldsymbol{\nu})\,\boldsymbol{\nu} + g(\mathbf{q}) = B\boldsymbol{\tau} + \sum_{\mathcal{L}} J_L^{\top}(\mathbf{q})\,\mathbf{f}_L^{ext},$$

where $\mathbf{q}$ is the generalised position and $\boldsymbol{\nu}$ is the generalised velocity already introduced in Equation (2.24). If a quaternion ${}^{W}\mathtt{Q}_{B}$ is used to model the orientation of the base frame B, as described in Section 7.1.1, we can expand the generalised position and velocity as follows:

$$\mathbf{q} = \left({}^{W}\mathbf{p}_B, {}^{W}\mathtt{Q}_B, \mathbf{s}\right) \in \mathbb{R}^{n+7} \tag{7.2}$$

$$\boldsymbol{\nu} = {}^{W}\boldsymbol{\nu} = \left({}^{W}\boldsymbol{v}_{W,B}, {}^{W}\boldsymbol{\omega}_{W,B}, \dot{\mathbf{s}}\right) \in \mathbb{R}^{n+6}, \tag{7.3}$$

where we chose the inertial-fixed representation of the base velocity, introduced in Section 2.3.2, just for practical reasons.

When the explicit decomposition of the Coriolis and gravitational terms is not required, we combine their effects into the following vector of *bias forces*:

$$h(\mathbf{q}, \boldsymbol{\nu}) = C(\mathbf{q}, \boldsymbol{\nu})\, \boldsymbol{\nu} + g(\mathbf{q}) \in \mathbb{R}^{6+n}.$$

We can compact the Lagrangian formulation of the system's dynamics even more by updating the terms related to the external 6D forces. We assume, for each link $L \in \mathcal{L}$, that an external 6D force $\mathbf{f}_L^{ext}$ always exists but is zero if the link has no interaction with the environment. If we stack the Jacobians and the 6D forces of all links in the following new matrices:

$$J_{\mathcal{L}}(\mathbf{q}) = \begin{bmatrix} J_0 \\ J_1 \\ \vdots \\ J_{n_L-1} \end{bmatrix} \in \mathbb{R}^{6n_L \times 6+n}, \qquad \mathbf{f}_{\mathcal{L}}^{ext} = \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_{n_L-1} \end{bmatrix} \in \mathbb{R}^{6n_L},$$

we can replace the sum with a more compact matrix product:

$$M(\mathbf{q})\, \dot{\boldsymbol{\nu}} + h(\mathbf{q}, \boldsymbol{\nu}) = B\boldsymbol{\tau} + J_{\mathcal{L}}^{\top}(\mathbf{q})\, \mathbf{f}_{\mathcal{L}}^{ext}. \tag{7.4}$$

7.2 STATE-SPACE MULTIBODY DYNAMICS

The EoMs of a multibody system of Equation (7.4) are expressed as non-linear Ordinary Differential Equation (ODE). Modern control theory studies the properties of these systems studied by operating on their *state-space* representation [Friedland, 2005], which assumes the following general form:

$$\begin{cases} \mathbf{x}(t) &= f\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \\ \mathbf{y}(t) &= g\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \end{cases}, \tag{7.5}$$

where the first is the *state equation* and the second is the *output equation*. The variable $\mathbf{x}$ is called *state vector* and the variable $\mathbf{u}$ is called *input vector*. State-space models are particularly interesting for computing, starting from a given initial state $\mathbf{x}_0$, the future trajectory of an arbitrarily complex system, under

the assumption of the complete knowledge of its dynamics $f(\cdot)$ and inputs $\mathbf{u}(t)$, if any. In the most general case, even when closed-form solutions of Equation (7.5) cannot be found, future trajectories can be computed through numerical integration [Cellier et al., 2006], as it will be described in Section 7.4.

In the case of a rigid multibody system, the EoMs (7.4) can be expressed in state-space representation by introducing the following state and input vectors:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\nu} \end{bmatrix} \in \mathbb{R}^{2n+13}, \qquad \mathbf{u}(t) = \begin{bmatrix} \boldsymbol{\tau} \\ \mathbf{f}_{\mathcal{L}}^{ext} \end{bmatrix} \in \mathbb{R}^{n+6n_L}. \tag{7.6}$$

With these definitions, we can define the state equation of our system as:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\nu}} \end{bmatrix} = f\left(\mathbf{x}(t), \mathbf{u}(t)\right),$$

where we still need to find the equations of $\dot{\mathbf{q}}$ and $\dot{\boldsymbol{\nu}}$.

The variable $\dot{\boldsymbol{\nu}} \in \mathbb{R}^{6+n}$ is the *generalised acceleration* of the system. Assuming the mass matrix $M(\mathbf{q})$ non-singular, the dynamics of $\boldsymbol{\nu}$ can be extracted directly from the EoMs (7.4). As we will introduce in Chapter 8, the computation of $\dot{\boldsymbol{\nu}}$ is also known as *forward dynamics* of the multibody system, and the methodology that requires the inversion of the mass matrix is just one among the available methods, not necessarily the most computationally efficient. In order to maintain a degree of generality, in this chapter we mark this computation with the $\mathrm{FD}(\cdot)$ function.

For what regards $\dot{\mathbf{q}}$, we can simply extend the base quantities of Equation (7.1) with the joint velocities $\dot{\mathbf{s}}$. The advantage of using the numerical integration form of the quaternion should now be clear, since it enables its direct inclusion in the state vector.

The final form of the state-space representation can be found by combining all the previous elements:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\nu}} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} {}^{W}\dot{\mathbf{p}}_{B} \\ \frac{1}{2}\,\mathrm{S}\left({}^{B}\mathbf{R}_{W}\,{}^{W}\boldsymbol{\omega}_{W,B}\right){}^{W}\mathbb{Q}_{B} \\ \dot{\mathbf{s}} \end{pmatrix} \\ \mathrm{FD}(\mathcal{M}, \mathbf{q}, \boldsymbol{\nu}, \boldsymbol{\tau}, \mathbf{f}_{\mathcal{L}}^{ext}) \end{bmatrix} = f\left(\mathbf{x}(t), \mathbf{u}(t)\right). \quad (7.7)$$

Also in this case, it's worth noting that $\dot{\mathbf{q}} \neq \boldsymbol{\nu}$, due to the nature of the angular variables of the base link.

Computing the evolution of this system requires the knowledge of its inputs, represented by the joint torques $\boldsymbol{\tau}$ and the external forces $\mathbf{f}_{\mathcal{L}}^{ext}$. The external forces could either be known 6D forces supplied by the user, or unknown 6D forces resulting from the interaction with the environment:

$$\mathbf{f}_{\mathcal{L}}^{ext} = \mathbf{f}_{\mathcal{L}}^{user} + \mathbf{f}_{\mathcal{L}}^{contact} \quad (7.8)$$

In the next section, we describe a methodology to compute the unknown forces exchanged with the environment by assuming that the floating-base model only interacts with the terrain $\mathbf{f}_{\mathcal{L}}^{contact} := \mathbf{f}_{\mathcal{L}}^{terrain}$, assumption compatible with our locomotion setting. In a more general setting, $\mathbf{f}_{\mathcal{L}}^{contact}$ should also include the forces exchanged between bodies, for example to simulate either self-collisions or interaction with other bodies part of the scene, particularly useful for robot manipulation.

7.3 CONTACT MODEL

Detecting and handling contacts between bodies is one of the most challenging processes of a rigid-body simulation. For simplicity, we assume that only contacts between points belonging to the model and a terrain surface can occur. Considering our locomotion scenario, this assumption allows us to describe robots with collision shapes composed of a set of *collidable points*. This approach enables a unified logic for shapes ranging from simple boxes to complex meshes. Note, however, that point-surface collisions do not provide expected results when a primitive shape like a box, modelled for example with its eight corner points, falls over the tip of a triangle-shaped terrain surface. In this case, the collision detection should consider the box as a surface instead of a set of points. If this use case is relevant, a possible workaround would be adding new collidable points on the box's surface, at a higher computational cost. Despite this limitation, the point-surface model could suffice in many target scenarios.

Gilardi et al. [2002] distinguished two different approaches for impact and contact analysis: *discrete* methods (also known as impulse-momentum) and *continuous* methods. They have shown that continuous methods are better suited for scenarios involving multiple contacts and bodies, allow for a better description of real systems, and simplify the inclusion of frictional effects. The main drawback is the introduction of at least two parameters that need to be appropriately identified to match the real contact dynamics. In robot learning, often this limitation is not particularly relevant since we can apply domain randomization over a realistic range of values. Also, in our setting, a continuous contact model has the advantage of providing smooth gradients when used in an AD context.

In the following sections, we first provide a description of the point-surface setting, introducing all the necessary elements for the contact model. Then, we provide a more detailed analysis of continuous methods for *collisions handling*, specifying how they can model both the *normal* and the *tangential* forces. Finally, we describe how their effects can be included in our dynamical system defined in Equation (7.7). The algorithm implementing the proposed soft-contact model is reported in the next chapter in Section 8.1.8.

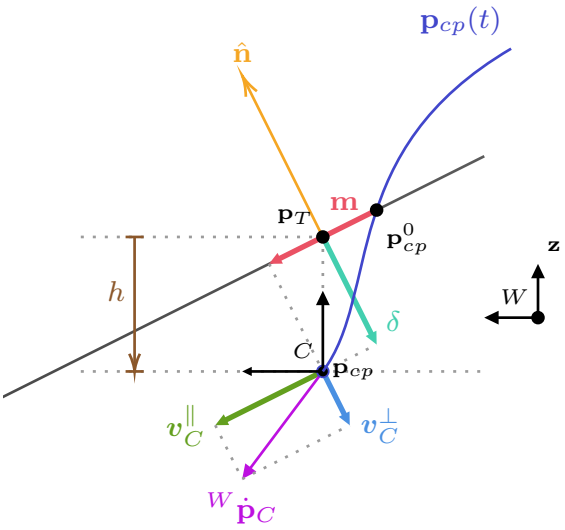


Figure 7.1: Illustration of the point-surface soft-contact model for non-planar terrains. The collidable point follow a trajectory $\mathbf{p}_{cp}(t)$, penetrating the ground in $\mathbf{p}^0_{cp} := (x^0, y^0, \mathcal{H}(x^0, y^0))$. While penetrating the material, the point reaches a generic point $\mathbf{p}_{cp}$, over which a local contact frame $C = (\mathbf{p}_{cp}, [W])$ is positioned, with a linear velocity ${}^{C[W]}\boldsymbol{v}_{W,C} = {}^W\dot{\mathbf{p}}_C \in \mathbb{R}^3$. The figure reports also the *penetration depth* $h \in \mathbb{R}$, the *normal deformation* $\delta \in \mathbb{R}$, and the compounded tangential deformation $\mathbf{m} \in \mathbb{R}^3$ of the material, used for the calculation of the 3D reaction force ${}_C\boldsymbol{f}_{cp}$ with the proposed soft-contact model.

7.3.1 Point-surface collisions

For each time-varying collidable point belonging to the simulated model, we introduce a new local frame $C = (\mathbf{o}_C, [W])$, having its origin located over the point's time-varying position ${}^W\mathbf{p}_{cp}(t)$ and orientation of W, illustrated in Figure 7.1. Beyond the position of the collidable point, the contact model accounts also its linear velocity $\boldsymbol{v}_{W,C} \in \mathbb{R}^3$. In the following formulation, we use the mixed representation of the point velocity, i.e. $\boldsymbol{v}_{W,C} = {}^W\dot{\mathbf{p}}_C$.

In this setup, *collision detection* is as easy as assessing if the z coordinate of the collidable point is lower than the terrain height. We can assume having a *heightmap* function $\mathcal{H} : (x, y) \mapsto z$ providing the terrain height at any location. We also assume to know the direction of the terrain normal $\hat{\mathbf{n}}$ in world coordinates at any location of the terrain's surface[32]. If ${}^W\mathbf{p}_T = (x_{cp}, y_{cp}, z_T)$

32 For smooth terrains, it can be shown that the normal can be estimated from $\mathcal{H}$.

is the point on the terrain surface vertical to the collidable point, where $z_T = \mathcal{H}(x_{cp}, y_{cp})$, we can compute the *penetration vector* as follows:

$$ {}^W\mathbf{h} = ({}^W\mathbf{p}_T - {}^W\mathbf{p}_{cp}) = \begin{bmatrix} 0 \\ 0 \\ (z_T - z_C) \end{bmatrix}, $$

where $h_z = z_T - z_C$ is the *penetration depth*, positive only for collidable points below the ground surface.

In the following sections, we need to project the penetration vector $\mathbf{h}$ and the linear velocity ${}^W\dot{\mathbf{p}}_C$ of the collidable point into the parallel and normal directions w.r.t. the ground surface. We denote the magnitude of the normal deformation as $\delta \in \mathbb{R}^+$, and the normal and tangential components of the velocity as $\boldsymbol{v}_C^{\perp}$, $\boldsymbol{v}_C^{\parallel} \in \mathbb{R}^3$:

$$ \begin{cases} \delta = {}^W\mathbf{h} \cdot \hat{\mathbf{n}}, \\ \boldsymbol{v}_C^{\perp} = \left({}^W\dot{\mathbf{p}}_C \cdot \hat{\mathbf{n}}\right) \hat{\mathbf{n}}, \\ \boldsymbol{v}_C^{\parallel} = {}^W\dot{\mathbf{p}}_C - \boldsymbol{v}_C^{\perp}. \end{cases} $$

We do not yet provide a geometrical equation to compute the *compounded tangential deformation* $\mathbf{m} \in \mathbb{R}^3$ of the terrain material, as it would require tracking over time the position of the initial penetration point ${}^W\mathbf{p}_{cp}^0$. Introducing in the system, for this purpose, an additional state component not part of its state-space representation would be difficult to handle. We will show in the next sections how to compute $\mathbf{m}$ such that both sticking and slipping contacts are supported. We also note that, assuming the knowledge of $\mathbf{m}$, the data required by the proposed contact model can be entirely computed from the floating-base configuration (generalized position $\mathbf{q}$ and velocity $\boldsymbol{\nu}$). Therefore, the contact force ${}_C\boldsymbol{f} \in \mathbb{R}^3$ becomes an instantaneous function of the kinematics.

Assuming that the effects of the normal and tangential deformations of the material can be decomposed, in the next sections we first compute the normal force ${}_C\boldsymbol{f}_{\perp} = f_{\perp}\hat{\mathbf{n}} \in \mathbb{R}^3$, and then the tangential force ${}_C\boldsymbol{f}_{\parallel} \in \mathbb{R}^3$, both applied to the origin of frame C.

7.3.2 Normal forces

Continuous contact models assume the existence of a relationship between the contact force and the deformation of the material [Romualdi et al., 2021]. Thanks to better properties in representing the physical nature of the energy transfer process, the most popular models adopted in the robotics community are those belonging to the non-linear family [Azad et al., 2016].

Considering the setting illustrated in Figure 7.1, we can compute the magnitude of the normal deformation and its rate as follows:

$$\begin{cases} \delta = {}^{W}\mathbf{h} \cdot \hat{\mathbf{n}}, \\ \dot{\delta} = {}^{W}\dot{\mathbf{h}} \cdot \hat{\mathbf{n}} = - {}^{W}\dot{\mathbf{p}}_C \cdot \hat{\mathbf{n}} = - \boldsymbol{v}_C^{\perp} \end{cases} .$$

A possible non-linear form of the relationship between the normal force $f_\perp$ and the deformation properties can be described by the following equation:

$$f_\perp = {}_C\boldsymbol{f}_\perp \cdot \hat{\mathbf{n}} = \begin{cases} k\delta^a + \lambda\delta^b\dot{\delta}^c, & \text{if } \delta \geq 0, \\ 0, & \text{if } \delta < 0, \end{cases}$$

where $k, \lambda \in \mathbb{R}$ are respectively the *stiffness* and *damping* coefficients of the material, and $a, b, c \in \mathbb{R}$ the parameters of the contact model. Note that this contact model does not present any discontinuity, in fact for what regards the term proportional to deformation rate we also have δ^b that zeros this term when $\delta = 0$.

This model has appeared with different coefficients proposed by various studies. In our implementation, we use the parameters $a = \frac{3}{2}$, $b = \frac{1}{2}$, and $c = 1$, as proposed by Azad et al. [2010]. Despite being formulated for sphere-plane collisions, we apply the same model to our point-surface setting, assuming

that the two collision types produce a comparable material deformation[33]. We can implement this model using the following logic:

$$f_\perp = \begin{cases} \max\left\{0, \sqrt{\delta}(k\delta + \lambda\dot{\delta})\right\} & \text{if } \delta \geq 0, \\ 0 & \text{if } \delta < 0. \end{cases} \tag{7.9}$$

As remarked by the same authors, this model has the advantage of not exposing any additional state variable. However, the implementation ignores the relaxation dynamics of the material in the normal direction after the contact is broken. This could cause incorrect dynamics if a new contact is made immediately following the deactivation of the previous one and before the spring-damper model could reach the steady state, but in practice the effect only occurs in the few instants before the contact becomes steady, not affecting the simulated dynamics significantly.

7.3.3 Tangential forces

The continuous contact dynamics introduced in the previous section allows for including frictional effects described with any friction model. We consider only effects due to *dry friction*. We approximate these effects with Coulomb's law of friction that, albeit being relatively simple, is widely adopted thanks to its versatility. The physical interaction between two materials is assumed to be independent of the contact area, accounting consistently for our point-surface modelling. The Coulomb friction for an object at rest is governed by the following model:

$$\left\| {}_C\boldsymbol{f}_\parallel \right\| \leq \mu_c f_\perp,$$

where $\boldsymbol{f}_\parallel \in \mathbb{R}^3$ is the tangential force that the material deformation exerts on the point in the direction opposite to the compounded tangential deformation $\mathbf{m}$, and $\mu_c \in \mathbb{R}^+$ is the *static friction coefficient*. This model depends on the unilateral force $f_\perp \geq 0$, and can be visualised as a cone considering a space

33 These parameters, being quite difficult to identify, often belong to the domain randomization set in RL settings.

having the three force components as axes. For this reason, it is often referred to as *friction cone*.

The friction cone defines two distinct contacts regimes: *sticking* if the tangential force magnitude is within the friction cone bounds, and *slipping* if outside:

$$ {}_C\boldsymbol{f}_{\parallel} = \begin{cases} \boldsymbol{f}_{stick} & \text{if } \|\boldsymbol{f}_{stick}\| \leq \mu_c f_{\perp}, \\ \boldsymbol{f}_{slip} & \text{otherwise.} \end{cases} $$

In practice, the two regimes are characterised respectively by the static friction coefficient $\mu_c \in \mathbb{R}^+$ and the sliding friction coefficient $\mu_k \in \mathbb{R}^+$, also called either kinetic or dynamic friction. As soon as the regime transitions from sticking to slipping, the considered coefficient of friction should be changed from the static to the dynamic. In the proposed model, in order to reduce the number of parameters to tune, we consider a unique coefficient $\mu = \mu_c = \mu_k$. The implementation can be changed trivially to use a different parameter in the slipping regime.

The same study we considered for the model of normal forces [Azad et al., 2010] proposes a spring-damper-clutch system for the tangential forces, where the additional clutch component controls the sticking-slipping condition. Extending their 2D formulation to our 3D point-surface setting, we can introduce the following relation between the tangential forces and the tangential material deformation:

$$ {}_C\boldsymbol{f}_{\parallel} = \alpha\mathbf{m} + \beta\dot{\mathbf{m}} = \alpha\mathbf{m} + \beta({}_C\boldsymbol{v}_C^{\parallel} - {}^{C[W]}\boldsymbol{v}_{W,clutch}), \tag{7.10} $$

where $\alpha, \beta \in \mathbb{R}^+$ are model parameters, $\mathbf{m} \in \mathbb{R}^3$ is the compounded 3D tangential deformation of the material as illustrated in Figure 7.1, and ${}^{C[W]}\boldsymbol{v}_{W,clutch}$ is the unknown clutch velocity.

When sticking, the clutch velocity is zero and, assuming the knowledge of $\mathbf{m}$, the tangential force can be computed with Equation (7.10). Instead, when the magnitude of the sticking force exceeds the friction cone bounds, the clutch is unlocked and the collidable point starts sliding. In slipping state, the tangential

force maintains the sticking direction, but enforces its magnitude to lay on the friction cone boundary:

$$ {}_C\boldsymbol{f}_{\parallel} = \begin{cases} \boldsymbol{f}_{stick} = \alpha\mathbf{m} + \beta\, \boldsymbol{v}_C^{\parallel} & \text{if sticking,} \\ \boldsymbol{f}_{slip} = \mu f_{\perp} \frac{\boldsymbol{f}_{stick}}{\|\boldsymbol{f}_{stick}\|} & \text{if slipping.} \end{cases} \tag{7.11} $$

We use $\alpha = -k_t\sqrt{\delta}$ and $\beta = -\lambda_t\sqrt{\delta}$ as presented by Azad et al. [2010], where also in this case we assume that collidable points produce a material deformation comparable to the sphere-plane setting.

The last missing point to discuss is how to calculate the compounded tangential deformation $\mathbf{m}$ of the material. Combining Equations (7.10) and (7.11), we can obtain the dynamics of the tangential deformation:

$$ \dot{\mathbf{m}} = \begin{cases} \boldsymbol{v}_C^{\parallel} & \text{if sticking,} \\ \beta^{-1}(\boldsymbol{f}_{slip} - \alpha\mathbf{m}) & \text{if slipping,} \\ -\alpha\beta^{-1}\mathbf{m} & \text{if contact is broken,} \end{cases} \tag{7.12} $$

that can be numerically integrated to obtain $\mathbf{m}$. It is worth noting that this formulation does not need to either know or keep track of the clutch velocity. In fact, as soon as a sticking contact transitions to slipping, we calculate $\dot{\mathbf{m}}$ such that we obtain exactly the desired $\boldsymbol{f}_{slip}$ (that is the projection of $\boldsymbol{f}_{stick}$ on the friction cone surface) given the current $(\mathbf{m}, \delta)$.

7.3.4 Augmented system dynamics

The effects of the contact model introduced in the previous sections can be included in the system's dynamics (7.7) by extending its state as follows:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\nu} \\ \mathrm{vec}(\mathbf{M}) \end{bmatrix} \in \mathbb{R}^{2n+3n_c+13}.$$

We introduced the matrix $\mathbf{M} \in \mathbb{R}^{3\times n_c}$ stacking the tangential deformations of all the n_c collidable points. The model's dynamics can be obtained from Equations (7.12) and plugged in the following contact-aware dynamic system:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\nu}} \\ \mathrm{vec}(\dot{\mathbf{M}}) \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} {}^W\dot{\mathbf{p}}_B \\ \frac{1}{2}\,\mathrm{S}\left({}^B\mathbf{R}_W\,{}^W\boldsymbol{\omega}_{W,B}\right){}^W\mathbb{Q}_B \\ \dot{\mathbf{s}} \end{pmatrix} \\ \mathrm{FD}(\mathcal{M}, \mathbf{q}, \boldsymbol{\nu}, \boldsymbol{\tau}, \mathbf{f}_{\mathcal{L}}^{ext}) \\ \mathrm{vec}(\dot{\mathbf{M}}) \end{bmatrix} = f\left(\mathbf{x}(t), \mathbf{u}(t)\right). \tag{7.13}$$

This final non-linear system, albeit being quite stiff when new contacts are made or broken, does not present any discontinuity.

The decomposition of the external forces of Equation (7.8) with the assumption of knowing the terrain profile (e.g. estimated by introducing perception algorithms) allows to simplify the input vector of Equation (7.6) as follows:

$$\mathbf{u}(t) = \left(\boldsymbol{\tau},\ \mathbf{f}_{\mathcal{L}}^{user}\right),$$

since the forces exchanged with the terrain can be computed from the kinematics and the terrain properties:

$$\mathbf{f}_{\mathcal{L}}^{contact} := \mathbf{f}_{\mathcal{L}}^{terrain} = \mathbf{f}_{\mathcal{L}}^{terrain}(\mathbf{q}, \boldsymbol{\nu}, \mathcal{H}, \mathcal{S}),$$

where $\mathcal{H} : (x, y) \mapsto z$ returns the terrain height and $\mathcal{S} : (x, y) \mapsto \hat{\mathbf{n}}$ returns the terrain normal.

7.4 INTEGRATORS

The evolution of dynamical systems described in state space forms like Equation (7.7) and Equation (7.13) can be obtained by numerical integration. Considering their simplicity and effectiveness, in this section we present two popular *iterative* and *explicit* integration methods to obtain the evolution in time of an *initial value problem* of the following form:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \dot{\mathbf{x}}(t) = f\left(\mathbf{x}(t), t\right), \qquad \mathbf{x}(t_0) = \mathbf{x}_0,$$

where $\mathbf{x}(t) \in \mathbb{R}^n$ and $t \in \mathbb{R}$. We want to obtain numerically the unknown $\mathbf{x}(t)$ from its known rate of change $\dot{\mathbf{x}}(t)$ and initial conditions $\mathbf{x}(t_0)$.

In this thesis, we will use *single-step* integration methods only, that compute the next value only from the previous one. These methods can be described by a function $v : \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n$:

$$\mathbf{x}(t + \mathrm{d}t) = v\left(\mathbf{x}(t), t\right).$$

The most common *single-step* method is *forward Euler*:

$$\mathbf{x}(t + \mathrm{d}t) = \mathbf{x}(t) + \mathrm{d}t\, f\left(\mathbf{x}(t), t\right).$$

It is the most basic explicit integration method that can be seen, by rearranging the equation, as an approximation of the forward finite difference formula:

$$\frac{\mathbf{x}(t + \mathrm{d}t) - \mathbf{x}(t)}{\mathrm{d}t} \approx \dot{\mathbf{x}}(t).$$

For sufficiently large integration steps $\mathrm{d}t$, and particularly for stiff systems, the numerical error due to the approximation of the forward Euler integration method could yield a diverging solution $\mathbf{x}(t)$, even if the system has globally and asymptotically stable equilibrium points. In these circumstances, the instability can be prevented by lowering the integration error through the reduction of the step size, at the expense of a greater computational cost. The forward Euler method, combined with a proper integration step size, remains a valid and widely employed method.

More complex integration schemes evaluate $f\left(\mathbf{x}(t), t\right)$ multiple times within the integration interval $[t, t + \mathrm{d}t]$. For example, the popular *Runge-Kutta* explicit methods implement the following integration scheme:

$$\mathbf{x}(t + \mathrm{d}t) = \mathbf{x}(t) + \mathrm{d}t \sum_{i=1}^{s} b_i \mathbf{k}_i,$$

where $\mathbf{k}_i = f\left(\mathbf{x}_i(t), t_i\right)$ is an intermediate evaluation of the system's dynamics and $\mathbf{x}_i(t)$ the reached intermediate states. Intuitively, these methods reduce the integration error by using an averaged value of the state derivative over the interval. The most widely adopted instance of this family is the Runge-Kutta 4 (RK4) corresponding to a 4th-order, which computes the slopes

$$\begin{aligned}
\mathbf{k}_1 &= f\left(\mathbf{x}(t), t\right) \\
\mathbf{k}_2 &= f\left(\mathbf{x}(t) + \frac{\mathrm{d}t}{2}\mathbf{k}_1, t + \frac{\mathrm{d}t}{2}\right) \\
\mathbf{k}_3 &= f\left(\mathbf{x}(t) + \frac{\mathrm{d}t}{2}\mathbf{k}_2, t + \frac{\mathrm{d}t}{2}\right) \\
\mathbf{k}_4 &= f\left(\mathbf{x}(t) + \mathrm{d}t\,\mathbf{k}_3, t + \mathrm{d}t\right)
\end{aligned}$$

weighted with $b_1 = b_4 = \frac{1}{6}$ and $b_2 = b_3 = \frac{2}{6}$. It can be seen that $(\mathbf{k}_1, \mathbf{k}_4)$ are the slopes evaluated at the extremes of the integration interval, and $(\mathbf{k}_2, \mathbf{k}_3)$ at the midpoint. In the average, a higher weight is given to the slopes at the midpoint.

7.5 VALIDATION

In this section, we validate the properties of the system through simple simulations of a rigid body interacting with the terrain. A single rigid body is equivalent to the multibody system defined Equation (7.13) having only variables related to the base and contacts. In all experiments, the soft-contact model is configured with $k = k_t = 10^6$, $\lambda = \lambda_t = 2000$, and $\mu = 0.5$, and the simulation is configured with a standard gravity of $g = 9.8\, m/s^2$.

We start the validation of the contact model on flat terrain by performing two experiments. In the first one, we calculate the mechanical energy of a bouncing ball, validating that it changes only upon impact due to the damping of the contact model. Given the continuous nature of the contact model, all the quantities should vary continuously. In the second experiment, we apply a step-wise force with increasing magnitude to the CoM of a box resting on a flat surface. The box should start accelerating only when the applied force is able to overcome the opposing effects due to friction. We conclude this section by validating the contact model on non-flat terrain. We simulate a falling box over an inclined plane characterised by different coefficients of friction, and compare its trajectory with the Mujoco simulator [E. Todorov et al., 2012]. The specifications of the machine used to execute the validation experiments are reported in Table 8.3.

7.5.1 Bouncing Ball

We consider a model composed of a single spherical-shaped link. The sphere has a mass of $0.1\, Kg$ and a radius of $10\, cm$. We approximate its collision shape with 500 points, all considered as collidable points for the collision detection and soft-contact model.

The sphere is positioned $1\, m$ above a flat surface, and left falling starting from an initial linear velocity of ${}^{B}\boldsymbol{v}_{W,B} = (2, 0, -1)\, m/s$. We simulate this setting for $1.5\, s$ using the RK4 integration scheme with a step size of $100\, \mu s$. The sphere's trajectory is illustrated in Figure 7.2.

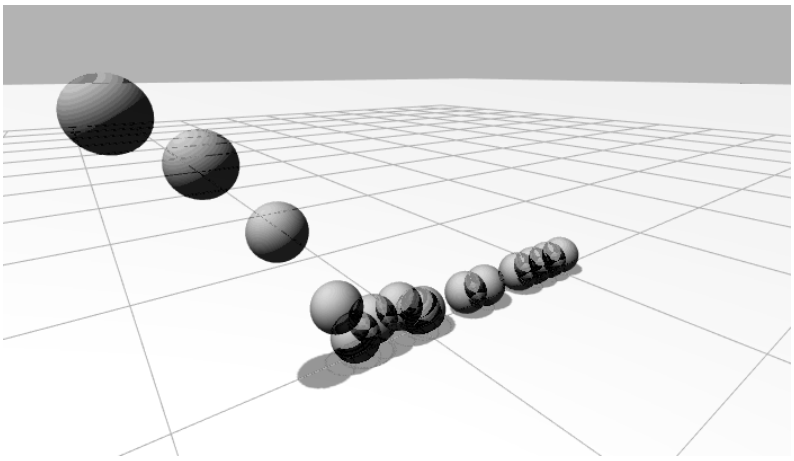
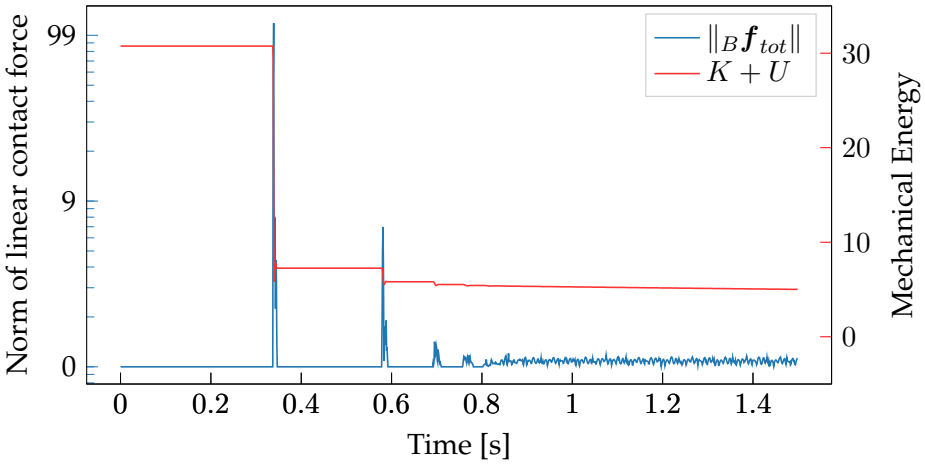


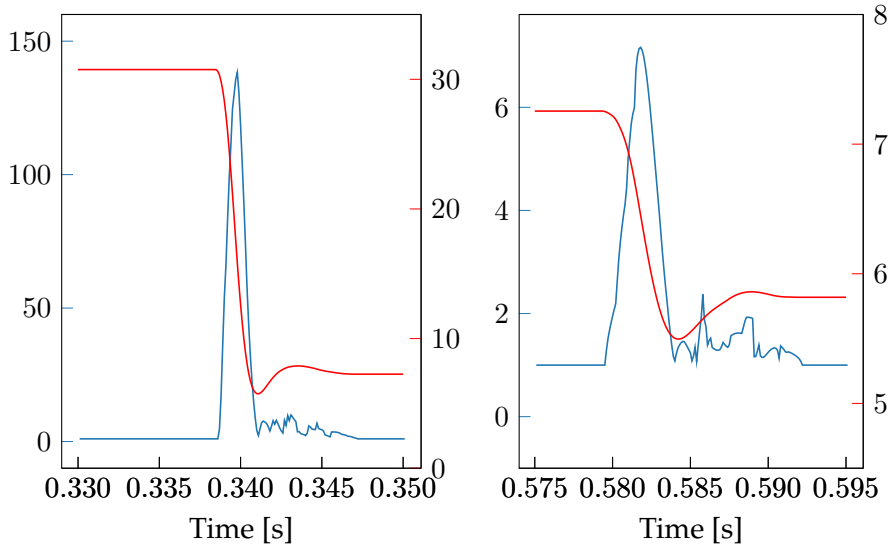
Figure 7.2: Sphere trajectory of the bouncing experiment.

At each time instant, we compute the system's mechanical energy by summing the potential and the kinetic energies obtained from Equation (2.15). We also get the linear contact forces ${}_{C_1}\boldsymbol{f}_1$, ${}_{C_2}\boldsymbol{f}_2$, ... computed with the soft-contact model of each collidable point, and combine them together in the frame B of the spherical link as ${}_B\mathbf{f}_{tot} = \sum_i {}_B\mathbf{X}^{C_i} \begin{bmatrix} {}_{C_i}\boldsymbol{f}_i^\top & \mathbf{0}_3^\top \end{bmatrix}^\top \in \mathbb{R}^6$.

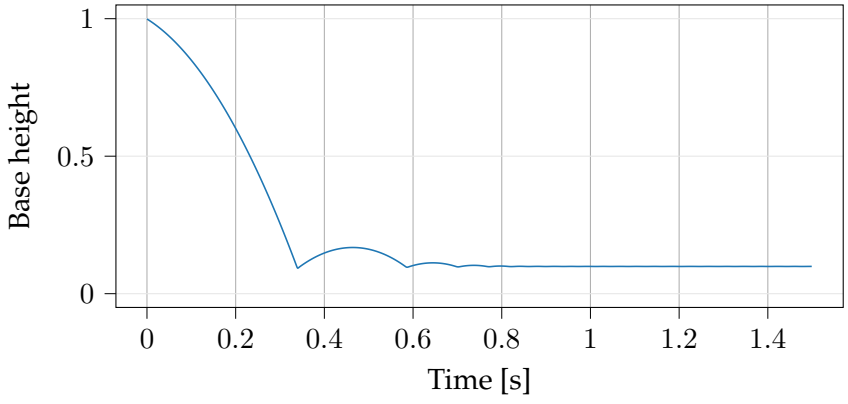
Figure 7.3 reports the height of the sphere corresponding to the z component of ${}^W\mathbf{p}_B$, the plot of the mechanical energy, and the plot norm of the contact force's linear component. It can be noticed that during the flight phase, the mechanical energy remains constant. It gets dissipated abruptly through the contact damping upon bouncing collisions, and linearly through the terrain friction when bouncing finishes and the sphere starts rolling (corresponding to the flat region in Figure 7.3d). From the detailed view of the first two impacts reported in Figure 7.3b and Figure 7.3c, it can be noticed that the abrupt energy drop actually varies continuously. From the same images, it can be seen that the soft-contact model produces contact forces that do not present marked discontinuities. However, if the step size of the simulation becomes larger, we observed that the initial penetration depth could generate a big initial reaction force that depends on the stiffness of the terrain. Possible solutions to mitigate this effect consists of either tuning the terrain parameters or adopting integration schemes with *zero-crossing* logic that allow obtaining small initial penetration depths by shortening the integration step of the instant when the contact is made.



(a)



(b) (c)



(d)

Figure 7.3: Evolution over time of the bouncing ball experiment's data. (7.3a) reports the mechanical energy of the system and the norm of the linear component of the contact forces summed and expressed in the B frame of the spherical link. (7.3b) and (7.3b) report a closer view of the first two impacts. (7.3d) reports the plot of the base height, where both the bouncing and rolling phases can be observed.

7.5.2 Sliding Box on Flat Terrain

We consider a model composed of a single box-shaped link. The box has a mass of $1\,Kg$ and (x, y, z) dimensions equal to $(1.5, 1, 0.5)\,m$. Its collision shape is approximated considering the 8 points corresponding to its corners.

The box is positioned on a flat ground surface at rest. We simulate this setting for $4\,s$ using the RK4 integration scheme with a step size of $1\,ms$. In this window, considering the frame of CoM $G = ({}^{W}\mathbf{p}_{CoM}, [B])$, we apply to the CoM of the box an external linear force ${}_{G}\boldsymbol{f}_{CoM} = (f_{CoM}, 0, 0) \in \mathbb{R}^3$ with a profile reported in Figure 7.4.

In this setting, the threshold of the friction cone separating the sticking and the slipping regimes is $\mu f_{\perp} = 4.9\,N$, averaged over the four contacts points of the bottom box surface. Figure 7.4 reports the plots of the x components of the CoM's position and linear velocity. It can be seen that, as expected, when the applied force is smaller than the threshold, the box stays still. As soon as the external force exceeds the threshold, the box starts accelerating. As soon as the external force goes to zero, the frictional effects of the contact model produce a reaction force that decelerates the box with a fast transient until it reaches the sticking regime again. Small velocity oscillations can be noticed when the external force is applied at $t = 0.5\,s$ and $t = 1\,s$, and when it is removed at $t = 3.5\,s$. They can be explained by the modelled dynamics of the material that can generate small tangential deformations without leaving the sticking regime.

7.5.3 Sliding Box on Inclined Plane

We consider a model composed of a single box-shaped link. The box has a mass of $1\,Kg$ and (x, y, z) dimensions equal to $(0.15, 0.1, 0.05)\,m$. Its collision shape is approximated considering the 8 points corresponding to its corners.

The box starts floating in the air having its CoM positioned in ${}^{W}\mathbf{p}_{G} = (0, 0, 1.0)\,m$. Then, we let the box fall due to gravity over a plane inclined by $20°$ so that the box slides down in the direction of the x axis. In this setting, we perform three experiment each characterised by a different coefficient of friction μ. We simulate this setting for $2.5\,s$ using the RK4 integration scheme

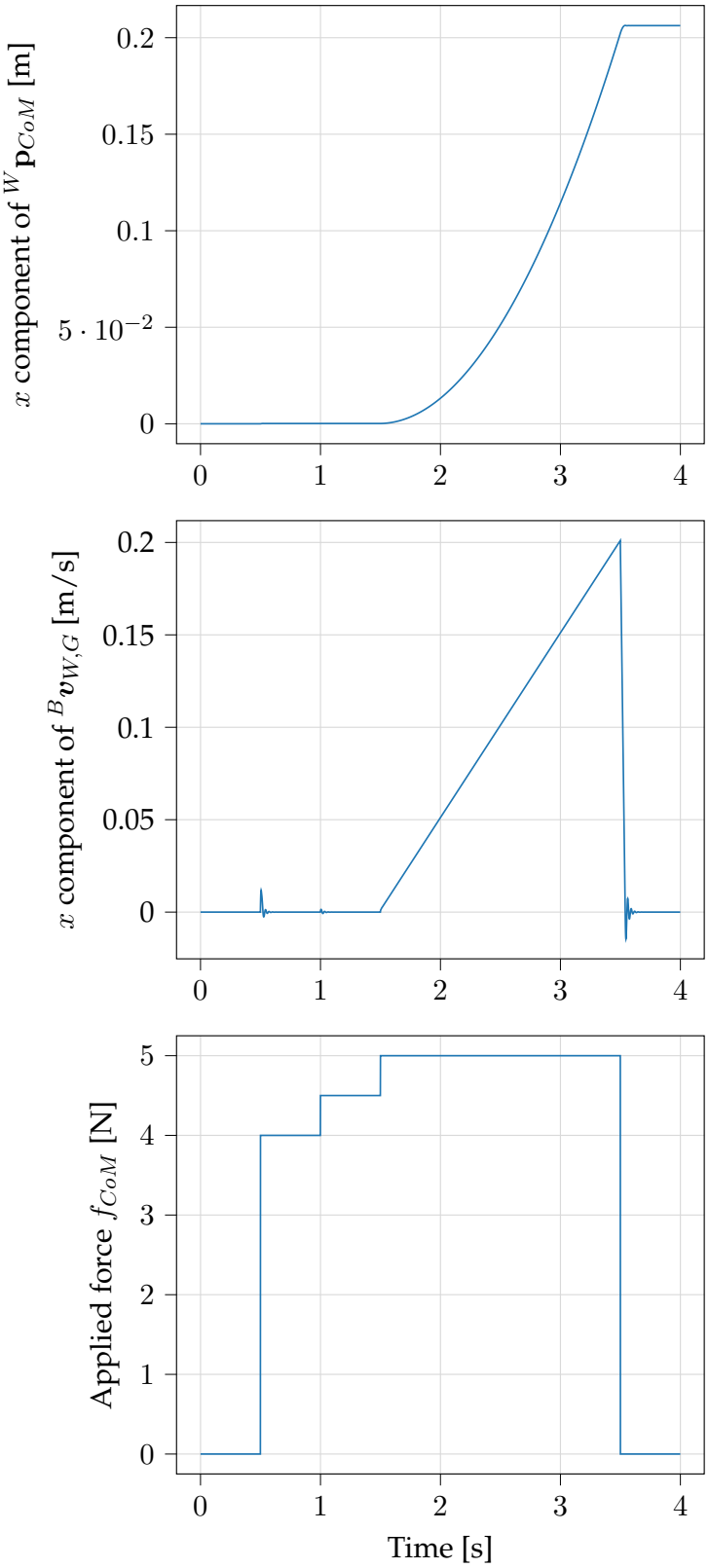
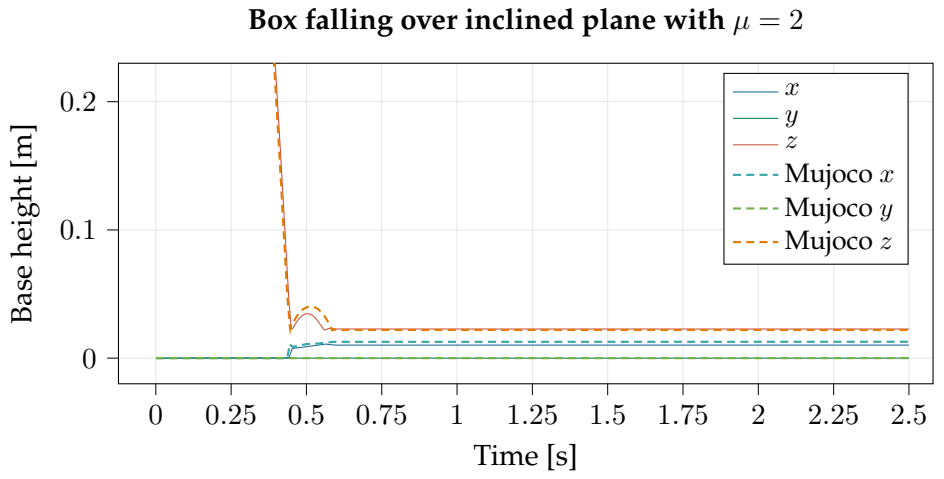
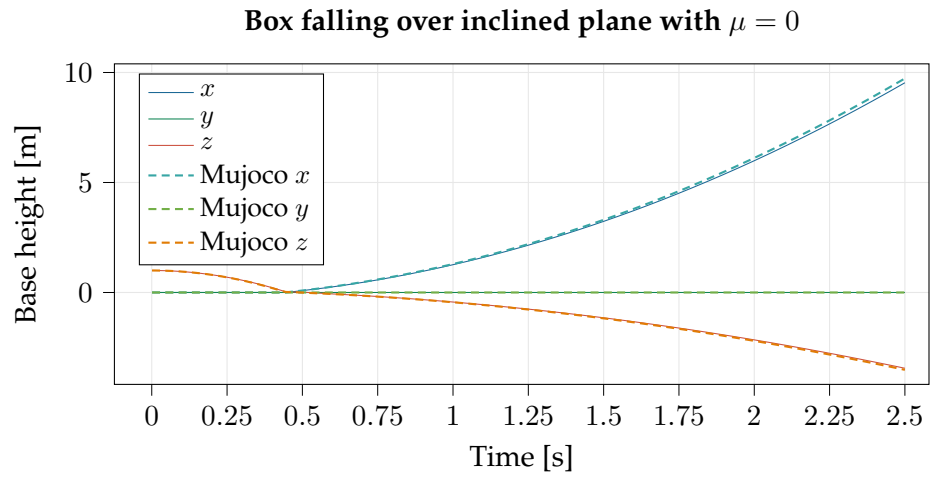


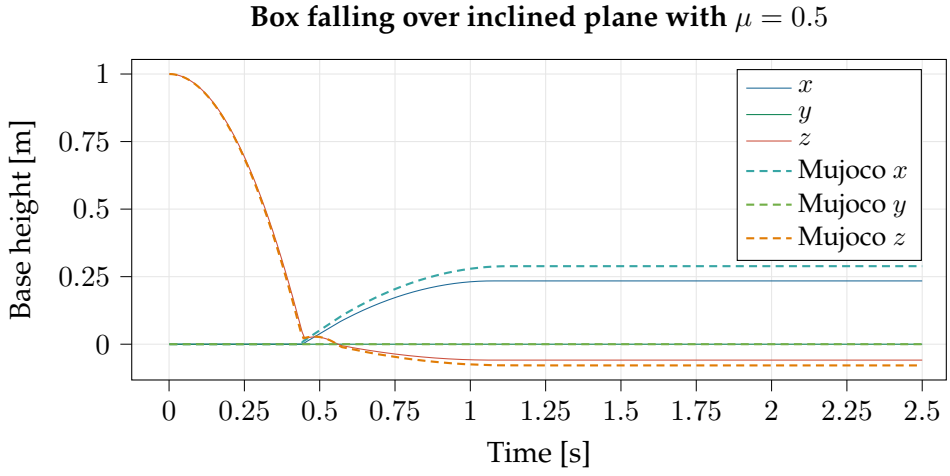
Figure 7.4: Evolution over time of the sliding box experiment's data. From top to bottom, the first plot shows the x component of the CoM position, the second plot shows the x component of the CoM velocity, and the third plot shows the profile of the applied external force to the CoM frame G.



(a)



(b)



(c)

Figure 7.5: Comparison of the box's CoM trajectory simulated with the proposed soft-contact model and the Mujoco simulator, considering a coefficient of friction (a) $\mu = 2$, (b) $\mu = 0$, (c) $\mu = 0.5$.

Table 7.1: Mujoco configuration considered in the experiments of the sliding box on inclined surface matching as close as possible the setting and properties of our soft-contact model. Refer to the official documentation at `https://mujoco.readthedocs.io` for a detailed explanation of the options.

Property	Value
timestep	0.001
integrator	RK4
solver	Newton
iterations	50
cone	elliptic
friction	$(\mu, 0, 0)$
solref	$(-1e6, -2000)$
condim	3

with a step size of $1\,ms$. For validation purpose, we simulate the same setting with the Mujoco simulator configured with comparable physics parameters, reported in Table 7.1. In particular, Mujoco provides a similar spring-damper soft-contact model that we can consider as ground truth.

In the first experiment, we consider an extremely large $\mu = 2.0$ so that we can assess how the selected stiffness k and damping λ affect the landing over the inclined plane without accounting for major sliding effects. From Figure 7.5a, it can be noticed that the falling trajectories due to gravity overlap perfectly until the impact. After the impact, the y position of both boxes always remains constant, showing that in both cases the generated tangential forces do not have any y component. In our soft-contact model this means that no material deformation occurs in this direction, as expected. The x and z components, although not matching perfectly due to the different formulation of the soft contacts –mainly due to Equation (7.9)– are sufficiently close and show the same qualitative behaviour.

In the second experiment, we consider a friction-less simulation with $\mu = 0$. In this setting, our soft-contact model only produces normal forces proportional to the penetration depth. The friction cone cannot be evaluated and no tangential forces are produced, therefore the only possible regime is sliding. Figure 7.5b reports the trajectories of the two falling boxes, showing in this case almost a perfect match between our soft-contacts model and Mujoco.

In the third experiment, we consider a realistic coefficient of friction $\mu = 0.5$. The combination of this coefficient of friction with the inclination of the terrain

has been selected for reaching a steady sticking regime after an initial sliding regime right after the impact. Figure 7.5c shows the trajectories of the two boxes. Also in this case, after the impact, both simulations correctly do not produce tangential forces in the y direction. The x and z components, similarly to the $\mu = 2.0$ case, do not match perfectly but also in this case show the same qualitative behaviour where the box transitions from the sliding to a steady sticking regime approximately at $t = 1.2\ s$.

All three experiments show that the proposed contact model behaves similarly to the ground truth represented by the soft-contact model implemented in the Mujoco simulator. The qualitative behaviour always look alike, although the trajectories do not necessarily match quantitatively. The main reason for the numerical mismatch is twofold. First, the formulation used to compute the normal force differs from Equation (7.9). This difference propagates also to the tangential component since the sticking-slipping boundary is a function of the normal force. Furthermore, the methodology to compute the contact force is completely different. In fact, we use a continuous contact model that introduces an additional state to the ODE describing the dynamics of the multi-body system. Mujoco, instead, at each time step solves a constrained quadratic optimisation problem. Further details on the internal details of Mujoco contacts can be found in [Emanuel Todorov, 2014; Vousten, 2022; Yoon et al., 2023].

7.6 CONCLUSIONS

In this chapter, we described how to model a floating-base multibody system interacting with a non-flat surface. This formulation lays the fundamentals of a physics engine, capable of simulating the evolution of such systems in time. We introduced a point-surface soft-contact model to compute the forces exchanged between points belonging to the links of the multibody system and a non-flat terrain, assuming to know its height profile at any point in space. This assumption simplifies the collision detection process, requiring just trivial geometrical assessments. The main benefit of our collisions and contacts modelling is the possibility of obtaining an extended state-space representation that includes both the dynamics of the multibody system and the dynamics of the contacts. The contact-aware evolution of the system can be derived by any numerical integration scheme. We validated the dynamical system in two simplified settings. In the first one, we considered single bodies interacting with a flat terrain, showing the continuity of the soft-contacts model and the switching between the sticking and slipping regimes. In the second one, we considered non-flat terrain, showing the trajectory of a box falling over an inclined plane characterised by different coefficients of friction. While sphere and box collisions might seem trivial examples, they often represent the typical collision shapes adopted to simulate robot's feet.

The final contact-aware dynamical system combined with the point-surface collision detection logic presents different limitations, especially when compared to implementations provided by general-purpose simulators. Our solution does not support detecting collisions between different bodies and does not consider joint limits and other types of constraints. To address the former, more advanced geometrical processing is necessary for detecting all ranges of collisions between points, primitive shapes, edges, surfaces, etc. For the latter, instead, it is possible to introduce an additional phase in the simulation step that computes the generalized forces to apply to the system for enforcing those constraints.

8 SCALING RIGID-BODY SIMULATIONS

In this final chapter, we provide our most recent results for addressing the problem of generating synthetic data for robot planning and control. As an initial attempt, in Chapter 5 we proposed the Scenario APIs and the Gym-Ignition framework that exploited the general purpose simulator Gazebo Sim to sample trajectories for training policies with RL. In Chapter 6, we have validated the framework proposing a scheme that, with experience sampled from Gazebo Sim, was able to train a RL policy capable of balancing a humanoid robot by adopting different push-recovery strategies for compensating external disturbances. We have observed that such a pipeline was characterised by long training times, and identified the simulator as the main bottleneck.

In this chapter, we propose our final simulation architecture for maximising the sampling performance of synthetic data for robot locomotion applications. Starting from the contact-aware state-space representation proposed in Chapter 7, we introduce state-of-the-art RBDAs for obtaining the terms forming the multibody EoMs and computing the *forward dynamics* function $\mathrm{FD}(\mathcal{M}, \mathbf{q}, \boldsymbol{\nu}, \boldsymbol{\tau}, \mathbf{f}_L)$ used in the $\dot{\mathbf{x}}(t)$ definition of Equation 7.12 and Equation 7.13. Excluding inverse dynamics, that we formulate as the true inverse of the forward dynamics also for floating-base systems, the implementations of the other RBDAs contain only minor differences compared to the reference formulation presented by Featherstone [2008]. Beyond being denoted with the notation introduced in Chapter 2 that makes explicit the reference frame of quantities like 6D velocities and 6D forces, we provide a unified implementation for both fixed-base and floating-base robots, and our inverse dynamics extends its acceleration to force mapping also to the base link. In the second part of this chapter, we propose JAXsim, a new physics engine in reduced coordinates that, thanks to the implementation of these algorithms in JAX, enables running rigid-body simulations on modern hardware accelerators like GPUs and TPUs. In fact, the problem of the previous architectures was not really the general-purpose simulator, but the limited parallel capabilities of CPUs in which it was executed. The RBDAs presented at the beginning of this chapter enable the combination of model-based algorithms with applications requiring high sampling rates like RL. They have been implemented such that they can be executed transparently on any supported hardware.

8.1 FLOATING-BASE RIGID-BODY DYNAMICS ALGORITHMS

In previous sections, we described how the dynamics of a floating-base system interacting with a non-flat terrain surface can be described, and how its evolution over time can be obtained through numerical integration. When we introduced the state-space representations, in Equation (7.7) and Equation (7.13), we used the *forward dynamics* function $\mathrm{FD}(\cdot)$ to describe the dynamics of the floating-base system velocity $\boldsymbol{\nu}$. We also mentioned that it can be calculated by inverting the following EoM, already defined in Equation (7.4):

$$M(\mathbf{q})\,\dot{\boldsymbol{\nu}} + h(\mathbf{q}, \boldsymbol{\nu}) = B\boldsymbol{\tau} + J_{\mathcal{L}}^{\top}(\mathbf{q})\,\mathbf{f}_{\mathcal{L}}^{ext}, \tag{8.1}$$

by assuming the knowledge, at any integration step, of the floating-base version of the mass matrix $M(\mathbf{q})$, the bias-vector $h(\mathbf{q}, \boldsymbol{\nu})$, and the Jacobians matrix $J_{\mathcal{L}}(\mathbf{q})$. Calculating the forward dynamics from the EoMs involves the inversion of the mass matrix that, depending on the number of the number of DoFs, could be computationally expensive when performed in a simulation loop. Furthermore, depending on the selected integrator, it might be needed to evaluate the system's dynamics multiple times for each simulation step, computing and inverting $M(\mathbf{q})$ at every evaluation.

Starting from the 80's, efficient iterative algorithms have been proposed to compute the quantities forming Equation (8.1) [Featherstone, 2008]. In this section, we provide an adaptation of such algorithms using the notation introduced in Chapter 2. We will particularly focus on two operations called *forward dynamics* and *inverse dynamics*, where the latter is the inverse of the former.

Definition 8.1.1 (Forward Dynamics)**.** Forward dynamics allows to compute the acceleration $\dot{\boldsymbol{\nu}}$ of the multibody system $\mathcal{M}$ given the position $\mathbf{q}$, the velocity $\boldsymbol{\nu}$, the generalised input joint forces $\boldsymbol{\tau}$, and the external link forces $\mathbf{f}_{\mathcal{L}}^{ext}$. It can be described with the following function:

$$\dot{\boldsymbol{\nu}} = \mathrm{FD}(\mathcal{M}, \mathbf{q}, \boldsymbol{\nu}, \boldsymbol{\tau}, \mathbf{f}_{\mathcal{L}}^{ext}).$$

□

Definition 8.1.2 (Inverse Dynamics)**.** Inverse dynamics allows to compute the joint torques $\boldsymbol{\tau}$ to apply on the multibody system $\mathcal{M}$ to produce a desired acceleration $\dot{\boldsymbol{\nu}}$ starting from the system's position $\mathbf{q}$ and velocity $\boldsymbol{\nu}$ together with the link external forces $\mathbf{f}_{\mathcal{L}}^{ext}$. It also calculates the 6D force ${}_{W}\mathbf{f}_{B}$ that, when applied to the base link, explains the component of the base acceleration ${}^{W}\mathbf{a}_{W,B}$ part of $\dot{\boldsymbol{\nu}}$ that is not due gravitational effects nor external forces. It can be described with the following function:

$$({}_{W}\mathbf{f}_{B}, \boldsymbol{\tau}) = \mathrm{ID}(\mathcal{M}, \mathbf{q}, \boldsymbol{\nu}, \dot{\boldsymbol{\nu}}, \mathbf{f}_{\mathcal{L}}^{ext}).$$

□

Beyond contact-related functionality, forward dynamics is the only necessary function for implementing a rigid-body simulator in reduced coordinates. However, we will see that inverse dynamics is necessary to compute all the terms forming the multibody EoMs (8.1). In fact, the most straightforward implementation of forward dynamics involves isolating $\dot{\boldsymbol{\nu}}$ from Equation (7.4). This computation, however, can be optimised by exploiting the sparsity of the kinematic tree that describes the multibody system. We will also present an algorithm that can efficiently compute forward dynamics by exploiting an iterative approach.

8.1.1 Implementation differences

The algorithms introduced in the next sections are an adaptation of those presented by Featherstone [2008]. Compared to the reference algorithms, our implementation contains the following modifications:

1. We present unified algorithms that work on both fixed-base and floating-base models. We allow specifying a custom pose of fixed-base models, and assume their base velocity and acceleration to be zero.

2. We use the notation of 6D quantities introduced in Chapter 2 that makes explicit the reference frame of 6D quantities like velocities, accelerations, and forces.

3. All the input and output quantities are expressed in inertial representation, with the exclusion of the mass matrix $M(\mathbf{q})$ and the Jacobian $J_{\mathcal{L}}(\mathbf{q})$ that are computed for efficiency reasons in body-fixed representation. Section 2.8.5 provides the transformations to apply for changing the velocity representation of the computed quantities.

4. We assume the model $\mathcal{M}$ only containing joints with 1 DoF. As a consequence, all joint quantities $\mathbf{s}, \dot{\mathbf{s}}, \ddot{\mathbf{s}}, \boldsymbol{\tau} \in \mathbb{R}^n$, where $n = N_B - 1$.

5. If the model description contains fixed joints, we assume they can be removed from the kinematic tree through a *lumping* process. If links P and C are connected with a fixed joint, the lumping process replaces the link pair (P, C) with an equivalent link $\tilde{P}$ associated to the frame of P, having an equivalent 6D inertia ${}_{\tilde{P}}\mathbb{M}_{\tilde{P}} = {}_{P}\mathbb{M}_{P} + {}_{P}\mathbf{X}^{C}\,{}_{C}\mathbb{M}_{C}\,{}^{C}\mathbf{X}_{P}$.

6. The floating-base RNEA implemented in Featherstone [2008, Section 9.5] is not a proper ID function as it was defined in Definition 8.1.2, since it treats the base acceleration as an output instead of being an input. In other words, the reference FD and ID are not inverse functions. In fact, the reference implementation is presented as a hybrid-dynamics problem, where the base acceleration is computed through a forward pass. Our RNEA implementation instead accepts as additional input the acceleration of the base ${}^{W}\mathbf{a}_{W,B}$, that forms the first six elements of the generalized acceleration ${}^{W}\dot{\boldsymbol{\nu}}$. We assume this acceleration being provided externally, either measured or estimated if applied on real robots. Our algorithm returns the 6D force ${}_{W}\mathbf{f}_{B}$ that, when exerted on the base link together with the optional external forces, produces the provided base acceleration. The effects of gravity are handled internally.

8.1.2 Model specification

The information about the kinematics, the inertial properties of all links, and the joint types, are included in the model specification $\mathcal{M}$[34]:

1. The numbering of the links follows the scheme introduced in Section 2.8.1, starting from the index 0 assigned to the base link. The base link B is selected when parsing the model description, and it is the root of the kinematic tree. N_B is the total number of bodies belonging to $\mathcal{M}$. When applied to fixed-base models, the algorithms ignore their base link dynamics.

2. The parent-to-child transforms ${}^{\lambda(i)}\mathbf{H}_i$ for all links $L \in \{\mathcal{L}/B\}$. We denote the link frames with the link index, i.e. ${}^{0}\mathbf{H}_1$ denotes the transform between link 0 and link 1.

3. The *parent array* $\lambda(i)$ that, given a link with index i, returns the index of its parent link.

4. The array $\kappa(i)$ that, given a link L with index i, returns the joints connecting all links of the path $\pi_B(L)$.

5. The 6D inertia of all links $\mathbb{M}_L$, expressed in link frame.

6. For each joint i, connecting the link pair $(\lambda(i), i)$, $\mathcal{M}$ includes the velocity transformation ${}^{\mathrm{pre}(i)}\mathbf{X}_{\lambda(i)}$ that locates the predecessor frame $\mathrm{pre}(i)$ of joint i from the frame $\lambda(i)$ of its parent link (refer to Figure 2.1 for a visual feedback).

7. The type of all joints, retrieved through the $\textsf{JTYPE}(\cdot)$ function from the joint index.

8. A function $\textsf{JCALC}(\cdot)$ accepting a joint type and a joint position $s \in \mathbb{R}$ that returns the motion subspace $\mathbf{S}$ and the joint velocity transform ${}^{\mathrm{suc}(\cdot)}\mathbf{X}_{\mathrm{pre}(\cdot)}$ corresponding to the given joint position (refer to Figure 2.1 for a visual feedback).

It is worth noting that all these quantities are constant, therefore $\mathcal{M}$ is immutable. In the algorithm listings, we use a superscript $(\cdot)^{\mathcal{M}}$ to mark these constant model quantities.

34 All information included in the model $\mathcal{M}$ could be directly computed from model descriptions like SDF, URDF, MJCF, etc.

8.1.3 Remarks

In the algorithm listings of the next sections, we use the following notation and assumptions:

1. The relation between the transformation of 6D forces and 6D velocities is ${}_A\mathbf{X}^B = {}^B\mathbf{X}_A^\top$. When needed, we often exploit the relation ${}_{\lambda(i)}\mathbf{X}^i = {}^i\mathbf{X}_{\lambda(i)}^\top$.

2. If $g \in \mathbb{R}^+$ is the gravitational acceleration, we denote the corresponding 6D acceleration as ${}^W\mathbf{a}_g = (0; 0; -g; \mathbf{0}_3) \in \mathbb{R}^6$.

3. Kinematic and dynamic quantities are propagated between links connected with joints with the relations introduced in Section 2.7. Regarding joint accelerations, we assume the presence of joints with only 1 DoF and a constant motion subspace, resulting to Equation (2.22) and the considerations reported in Definition 2.7.3.

4. We use 1-based indexing for vectors and matrices. This means that, if $A \in \mathbb{R}^{n\times m}$, the top-left element is $A_{(1,1)}$ and the bottom-right $A_{(n,m)}$. Selectors of multiple elements use the : delimiter, for example $A_{(1:3,1:3)}$ is the top-left 3×3 block, and $A_{(1:3,1)}$ its first column. Note that, while this notation simplifies pseudocode, it requires a careful implementation with 0-based array libraries.

8.1.4 CRBA: Composite Rigid Body Algorithm

The Composite Rigid Body Algorithm (CRBA) is an efficient algorithm that computes the body-fixed mass matrix $M_B(\mathbf{s})$ defined in Equation (2.29) having the following structure, already discussed in Remark 2.8.3:

$$M_B(\mathbf{s}) = \begin{bmatrix} {}_B\mathbb{M}(\mathbf{s}) & F(\mathbf{s}) \\ F^\top(\mathbf{s}) & H(\mathbf{s}) \end{bmatrix} \in \mathbb{R}^{(6+n)\times(6+n)}.$$

A reference of the CRBA for fixed-base and floating-base systems can be found in [Featherstone, 2008, Section 6.2 and Section 9.4]. Algorithm 1 reports a unified version of the CRBA for both floating-base and fixed-base systems. It calculates the body-fixed mass matrix $M_B(\mathbf{s})$, that in this representation only

depends on the shape $\mathbf{s}$. The knowledge of the mass matrix is the first element that allows the calculation of the forward dynamics from the system's EoMs. The algorithm takes the following inputs:

- $\mathcal{M}$: the model containing the multibody system constant data;
- $\mathbf{s} \in \mathbb{R}^n$: the joint positions of all joints belonging to the multibody system;

and provides the following output:

- $M \in \mathbb{R}^{(6+n)\times(6+n)}$: the floating-base mass matrix in body-fixed velocity representation.

The resulting mass matrix can be converted to other velocity representations with the change of coordinates introduced in Section 2.8.5.

Algorithm 1 Composite Rigid Body Algorithm

inputs $(\mathcal{M}, \mathbf{s})$
$\mathbb{M}_0^c = \mathbb{M}_0^{\mathcal{M}}$
${}^0\mathbf{X}_0 = \mathbf{I}_6$
for $i = 1$ **to** $N_B^{\mathcal{M}}$ **do** ▷ *Propagate kinematics*
  $[{}^i\mathbf{X}_{pre}, \mathbf{S}_i] = \text{JCALC}(\text{JTYPE}(i), s_i)$
  ${}^i\mathbf{X}_{\lambda(i)} = {}^i\mathbf{X}_{pre}\, {}^{pre}\mathbf{X}_{\lambda(i)}^{\mathcal{M}}$
  ${}^i\mathbf{X}_0 = {}^i\mathbf{X}_{\lambda(i)}\, {}^{\lambda(i)}\mathbf{X}_0$
  $\mathbb{M}_i^c = \mathbb{M}_i^{\mathcal{M}}$ ▷ *Initialize composite inertia*
$M = \mathbf{0}_{(6+n)\times(6+n)}$
for $i = N_B^{\mathcal{M}}$ **to** 1 **do**
  $\mathbb{M}_{\lambda(i)}^c = \mathbb{M}_{\lambda(i)}^c + {}^i\mathbf{X}_{\lambda(i)}^\top\, \mathbb{M}_i^c\, {}^i\mathbf{X}_{\lambda(i)}$
  ▷ *Compute M_{jj}* ◁
  $\mathbf{F}_i = \mathbb{M}_i^c \mathbf{S}_i$
  $M_{(i+6,i+6)} = \mathbf{S}_i^\top \mathbf{F}_i$
  $j = i$
  while $\lambda(j) \neq 0$ **do**
    $\mathbf{F}_i = {}^j\mathbf{X}_{\lambda(j)}^\top \mathbf{F}_i$
    $j = \lambda(j)$
    $M_{(i+6,j+6)} = \mathbf{F}_i^\top \mathbf{S}_j$
    $M_{(j+6,i+6)} = M_{(i+6,j+6)}$
  ▷ *Compute M_{jb} and M_{bj}* ◁
  $\mathbf{F}_i = {}^j\mathbf{X}_0^\top \mathbf{F}_i$
  $M_{(i+6,1:6)} = \mathbf{F}_i^\top$
  $M_{(1:6,i+6)} = \mathbf{F}_i$
$M_{(1:6,1:6)} = \mathbb{M}_0^c$ ▷ *Compute M_{bb}*
output M

8.1.5 RNEA: Recursive Newton-Euler Algorithm

The RNEA is an efficient algorithm that computes the inverse dynamics of a multibody system:

$$\boldsymbol{\tau} = \mathrm{ID}(\mathcal{M}, \mathbf{q}, \boldsymbol{\nu}, \dot{\boldsymbol{\nu}}, \mathbf{f}_{\mathcal{L}}^{ext}).$$

A reference of the RNEA for fixed-base and floating base systems can be found in [Featherstone, 2008, Section 5.3 and Section 9.5]. Algorithm 2 reports a unified version of the RNEA for both floating-base and fixed-base systems. It calculates the joint torques to be applied to the multibody system to produce the provided acceleration starting from a given position and velocity. In particular, it takes the following inputs:

- $\mathcal{M}$: the model containing the multibody system constant data;
- $\mathbf{s}, \dot{\mathbf{s}}, \ddot{\mathbf{s}} \in \mathbb{R}^n$: the positions, velocities, and accelerations of all joints belonging to the multibody system;
- ${}^{W}\mathbf{X}_{B} \in \mathbb{R}^{6\times6}$: the velocity transform from the base link B to the world link W;
- ${}^{W}\mathbf{v}_{W,B} \in \mathbb{R}^6$: the inertial-fixed velocity of the base link B;
- ${}^{W}\mathbf{a}_{W,B} \in \mathbb{R}^6$: the 6D inertial-fixed intrinsic acceleration of the base link B that, considering the reference frame, corresponds to the apparent acceleration ${}^{W}\mathbf{a}_{W,B} = {}^{W}\dot{\mathbf{v}}_{W,B}$;
- ${}_{W}\mathbf{f}_{\mathcal{L}}^{ext} \in \mathbb{R}^{N_B^{\mathcal{M}}\times6}$: the matrix stacking, for all $N_B^{\mathcal{M}}$ links belonging to the model, their corresponding external force ${}_{W}\mathbf{f}_{i}^{ext} \in \mathbb{R}^6$ is expressed in world coordinates;
- ${}^{W}\mathbf{a}_{g} \in \mathbb{R}^6$: the gravitational 6D acceleration;

and provides the following outputs:

- $\boldsymbol{\tau} \in \mathbb{R}^n$: the generalized forces of all joints belonging to the multibody system;

- $\mathbf{f}_0 \in \mathbb{R}^6$: if the model is floating base, the 6D force ${}_W\mathbf{f}_B$ that, applied to the base link, produces the real acceleration ${}^W\mathbf{a}_{W,B}$ provided as input, or zero if fixed-base.

RNEA can also be used to efficiently compute other components of the EoM necessary to compute the forward dynamics. In fact, the following relations hold:

$$\begin{cases} g(\mathbf{q}) = \mathrm{ID}(\mathcal{M}, \mathbf{q}, \mathbf{0}_{6+n}, \mathbf{0}_{6+n}, \mathbf{0}_{6n_L}) \\ h(\mathbf{q}, \boldsymbol{\nu}) = \mathrm{ID}(\mathcal{M}, \mathbf{q}, \boldsymbol{\nu}, \mathbf{0}_{6+n}, \mathbf{0}_{6n_L}) \\ C(\mathbf{q}, \boldsymbol{\nu})\, \boldsymbol{\nu} = h(\mathbf{q}, \boldsymbol{\nu}) - g(\mathbf{q}) \end{cases} .$$

This implementation provides such quantities in inertial-fixed representation, and can be converted to the other ones using the change of coordinates provided in Section 2.8.5.

Algorithm 2 Recursive Newton-Euler Algorithm

inputs $\left(\mathcal{M}, \mathbf{s}, \dot{\mathbf{s}}, \ddot{\mathbf{s}}, {}^W\mathbf{X}_B, {}^W\mathbf{v}_{W,B}, {}^W\mathbf{a}_{W,B}, {}_W\mathbf{f}_{\mathcal{L}}^{ext}, {}^W\mathbf{a}_g\right)$
${}^0\mathbf{X}_W = {}^B\mathbf{X}_W$ ▷ *Initialise base transform*
${}^0\mathbf{X}_0 = \mathbf{I}_6$
$\mathbf{f}_0 = \mathbf{0}_6$
if floating **then** ▷ *Initialise base quantities*
  ${}_0\mathbf{X}^W = {}^0\mathbf{X}_W^{-\top}$
  $\mathbf{v}_0 = {}^0\mathbf{v}_{W,0} = {}^0\mathbf{X}_W\, {}^W\mathbf{v}_{W,0}$
  $\bar{\mathbf{a}}_0 = {}^0\bar{\mathbf{a}}_{W,0} = {}^0\mathbf{X}_W({}^W\mathbf{a}_{W,B} - {}^W\mathbf{a}_g)$
  $\mathbf{f}_0 = {}_B\mathbf{f}_B = \mathbb{M}_0^{\mathcal{M}}\, \bar{\mathbf{a}}_0 + \mathbf{v}_0 \bar{\times}^*\, \mathbb{M}_0^{\mathcal{M}}\, \mathbf{v}_0 - {}_0\mathbf{X}^W\, {}_W\mathbf{f}_0^{ext}$
for $i = 1$ **to** $N_B^{\mathcal{M}}$ **do** ▷ *Forward pass*
  ▷ *Compute parent-to-child transform* ◁
  $[{}^i\mathbf{X}_{pre}, \mathbf{S}_i] = \textsc{jcalc}(\textsc{jtype}(i), s_i)$
  ${}^i\mathbf{X}_{\lambda(i)} = {}^i\mathbf{X}_{pre}\, {}^{pre}\mathbf{X}_{\lambda(i)}^{\mathcal{M}}$
  ▷ *Propagate link velocity and acceleration* ◁
  $\mathbf{v}_i = {}^i\mathbf{v}_{\lambda(i),i} = {}^i\mathbf{X}_{\lambda(i)}\, \mathbf{v}_{\lambda(i)} + \mathbf{S}_i\dot{s}_i$
  $\bar{\mathbf{a}}_i = {}^i\bar{\mathbf{a}}_{\lambda(i),i} = {}^i\mathbf{X}_{\lambda(i)}\, \bar{\mathbf{a}}_{\lambda(i)} + \mathbf{S}_i\ddot{s}_i + \mathbf{v}_i \bar{\times} \mathbf{S}_i\dot{s}_i$
  ▷ *Start propagating link force* ◁
  ${}^i\mathbf{X}_0 = {}^i\mathbf{X}_{\lambda(i)}\, {}^{\lambda(i)}\mathbf{X}_0$
  ${}_i\mathbf{X}^W = ({}^i\mathbf{X}_0\, {}^0\mathbf{X}_W)^{-\top}$
  $\mathbf{f}_i = {}_i\mathbf{f}_i = \mathbb{M}_i^{\mathcal{M}}\, \bar{\mathbf{a}}_i + \mathbf{v}_i \bar{\times}^*\, \mathbb{M}_i^{\mathcal{M}}\, \mathbf{v}_i - {}_i\mathbf{X}^W\, {}_W\mathbf{f}_i^{ext}$
for $i = N_B^{\mathcal{M}}$ **to** 1 **do** ▷ *Backward pass*
  $\tau_i = \mathbf{S}_i^\top\, \mathbf{f}_i$
  if $\lambda(i) \neq 0$ or floating **then** ▷ *Finalize link force propagation*
    $\mathbf{f}_{\lambda(i)} = \mathbf{f}_{\lambda(i)} + {}^i\mathbf{X}_{\lambda(i)}^\top\, \mathbf{f}_i$
outputs $(\boldsymbol{\tau}, \mathbf{f}_0)$

8.1.6 Free-floating Jacobian

The free-floating Jacobian of a link $E \in \mathcal{L}$, already defined in Definition 2.8.8 has the following block structure:

$$ {}^{Y}J_{W,E/X}(\mathbf{q}) = \begin{bmatrix} {}^{Y}\mathbf{X}_{X} & {}^{Y}S_{B,E}(\mathbf{s}) \end{bmatrix}. $$

The relevant equations to compute the Jacobian matrix for fixed-base and floating-base systems can be found in [Featherstone, 2008, Section 4.1 and Section 9.5]. Algorithm 3 reports a unified version for the computation of the left-trivialized floating-base Jacobian for left-trivialized system velocities ${}^{E}J_{W,E/B}$. The following relation provides the intended usage that highlights the correct representations:

$$ {}^{E}\mathbf{v}_{W,E} = {}^{E}J_{W,E/B}(\mathbf{q})\, {}^{B}\boldsymbol{\nu}. $$

Note that using these representations for input and output velocities, the free-floating Jacobian does not depend on base variables. The algorithm takes the following inputs:

- $\mathcal{M}$: the model containing the multibody system constant data;
- $\mathbf{s} \in \mathbb{R}^n$: the joint positions of all joints belonging to the multibody system;
- $\mathrm{idx}(L_i) \in 0 \cup \mathbb{N}$: the index of the link of the returned Jacobian;

and returns the following output:

- $J \in \mathbb{R}^{6\times n}$: the free-floating Jacobian of the link L_i.

Algorithm 3 Floating-base doubly-left Jacobian

inputs $(\mathcal{M}, \mathbf{s}, L_i)$
${}^{0}\mathbf{X}_{0} = \mathbf{I}_6$
for $i = 1$ **to** $N_B^{\mathcal{M}}$ **do** ▷ *Propagate kinematics*
  $[{}^{i}\mathbf{X}_{pre}, \mathbf{S}_i] = \textsc{jcalc}(\textsc{jtype}(i), s_i)$
  ${}^{i}\mathbf{X}_{\lambda(i)} = {}^{i}\mathbf{X}_{pre}\, {}^{pre}\mathbf{X}_{\lambda(i)}^{\mathcal{M}}$
  ${}^{i}\mathbf{X}_{0} = {}^{i}\mathbf{X}_{\lambda(i)}\, {}^{\lambda(i)}\mathbf{X}_{0}$
$J = \mathbf{0}_{6\times(6+N_B^{\mathcal{M}})}$
$J_{(1:6,1:6)} = {}^{L_i}\mathbf{X}_{0}$ ▷ *Compute J_b*
for $i = 1$ **to** $N_B^{\mathcal{M}}$ **do** ▷ *Compute J_s*
  if $i \in \kappa^{\mathcal{M}}(L_i)$ **then**
    $J_{(1:6,6+i)} = {}^{L_i}\mathbf{X}_{0}\, {}^{0}\mathbf{X}_{i}\mathbf{S}_i$
output J

8.1.7 ABA: Articulated-Body Algorithm

Given a model $\mathcal{M}$, the forward dynamics computation is defined with the following signature:

$$\dot{\boldsymbol{\nu}} = \mathrm{FD}(\mathcal{M}, \mathbf{q}, \boldsymbol{\nu}, \boldsymbol{\tau}, \mathbf{f}^{ext}_{\mathcal{L}}).$$

As we already mentioned earlier in this chapter, forward dynamics can be computed by inverting the EoM:

$$\dot{\boldsymbol{\nu}} = \begin{bmatrix} \mathbf{a}_{W,B} \\ \ddot{\mathbf{s}} \end{bmatrix} = M^{-1}(\mathbf{q}) \left[B\boldsymbol{\tau} - h(\mathbf{q}, \boldsymbol{\nu}) + J^{\top}_{\mathcal{L}}(\mathbf{q})\, \mathbf{f}^{ext}_{\mathcal{L}} \right].$$

The previous algorithms provide all the necessary components for this computation.

The computation and inversion of M, depending on the number of the system's DoFs, could be expensive. The ABA is an efficient iterative algorithm to compute the forward dynamics. A reference for fixed-base and floating-base systems can be found in [Featherstone, 2008, Section 7.3 and Section 9.4]. Algorithm 4 reports a unified version of ABA for both floating-base and fixed-based systems. It takes the following inputs:

- $\mathcal{M}$: the model containing the multibody system constant data;
- $\mathbf{s}, \dot{\mathbf{s}} \in \mathbb{R}^n$: the positions and velocities of all joints belonging to the multibody system;
- $\boldsymbol{\tau} \in \mathbb{R}^n$: the generalised forces of all joints belonging to the multibody system;
- ${}^{W}\mathbf{X}_{B} \in \mathbb{R}^{6\times 6}$: the velocity transform from the base link B to the world link W;
- ${}^{W}\mathbf{v}_{W,B} \in \mathbb{R}^6$: the inertial-fixed velocity of the base link B;
- ${}_{W}\mathbf{f}^{ext}_{\mathcal{L}} \in \mathbb{R}^{N_B^{\mathcal{M}}\times 6}$: the matrix stacking, for all $N_B^{\mathcal{M}}$ links belonging to the model, their corresponding external force ${}_{W}\mathbf{f}^{ext}_{i} \in \mathbb{R}^6$ is expressed in world coordinates;

- ${}^{W}\mathbf{a}_g \in \mathbb{R}^6$: the gravitational 6D acceleration;

and provides the following outputs:

- $\ddot{\mathbf{s}} \in \mathbb{R}^n$: the accelerations of all joints belonging to the multibody system;
- ${}^{W}\mathbf{a}_{W,B} \in \mathbb{R}^6$: if the model is floating base, the 6D inertial-fixed intrinsic acceleration of the base link B that, considering the reference frame, corresponds to the apparent acceleration ${}^{W}\mathbf{a}_{W,B} = {}^{W}\dot{\mathbf{v}}_{W,B}$.

For simulation purposes, forward dynamics is the only necessary function to implement, and ABA provides an efficient option. Considering the state-space system introduced in Equation (7.7), we can now define the state variables $\mathbf{x}$ and it derivative $\dot{\mathbf{x}}$ as:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} {}^{W}\mathbf{p}_B \\ {}^{W}\mathtt{Q}_B \\ \mathbf{s} \end{pmatrix} \\ \begin{pmatrix} {}^{W}\boldsymbol{v}_{W,B} \\ {}^{W}\boldsymbol{\omega}_{W,B} \\ \dot{\mathbf{s}} \end{pmatrix} \end{bmatrix},$$

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\boldsymbol{\nu}} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} {}^{B[W]}\boldsymbol{v}_{W,B} \\ \frac{1}{2}\,\mathrm{S}\left({}^{B}\mathbf{R}_{W}\,{}^{W}\boldsymbol{\omega}_{W,B}\right){}^{W}\mathtt{Q}_B \\ \dot{\mathbf{s}} \end{pmatrix} \\ \begin{pmatrix} {}^{W}\mathbf{a}_{W,B} \\ \ddot{\mathbf{s}} \end{pmatrix} \end{bmatrix}.$$

Algorithm 4 Articulated Body Algorithm

inputs $\left(\mathcal{M}, \mathbf{s}, \dot{\mathbf{s}}, \boldsymbol{\tau}, {}^{W}\mathbf{X}_{B}, {}^{W}\mathbf{v}_{W,B}, {}_{W}\mathbf{f}_{\mathcal{L}}^{ext}, {}^{W}\mathbf{a}_{g}\right)$
${}^{0}\mathbf{X}_{W} = {}^{B}\mathbf{X}_{W}$ ▷ *Initialise base transform*
${}^{0}\mathbf{X}_{0} = \mathbf{I}_{6}$
if floating **then** ▷ *Initialise base quantities*
  ${}_{0}\mathbf{X}^{W} = {}^{0}\mathbf{X}_{W}^{-\top}$
  $\mathbf{v}_{0} = {}^{0}\mathbf{v}_{W,0} = {}^{0}\mathbf{X}_{W}\, {}^{W}\mathbf{v}_{W,0}$
  $\mathbb{M}_{0}^{A} = \mathbb{M}_{0}^{\mathcal{M}}$
  $\mathbf{p}_{0}^{A} = \mathbf{v}_{0}\bar{\times}^{*}\, \mathbb{M}_{0}^{A}\, \mathbf{v}_{0} - {}_{0}\mathbf{X}^{W}\, {}_{W}\mathbf{f}_{0}^{ext}$
for $i = 1$ **to** $N_{B}^{\mathcal{M}}$ **do** ▷ *Pass 1*
  ▷ *Compute parent-to-child transform* ◁
  $[{}^{i}\mathbf{X}_{pre}, \mathbf{S}_{i}] = \textsc{jcalc}(\textsc{jtype}(i), s_{i})$
  ${}^{i}\mathbf{X}_{\lambda(i)} = {}^{i}\mathbf{X}_{pre}\, {}^{pre}\mathbf{X}_{\lambda(i)}^{\mathcal{M}}$
  ▷ *Propagate link velocity* ◁
  $\mathbf{v}_{J} = \mathbf{S}_{i}\dot{s}_{i}$
  $\mathbf{v}_{i} = {}^{i}\mathbf{v}_{\lambda(i),i} = {}^{i}\mathbf{X}_{\lambda(i)}\, \mathbf{v}_{\lambda(i)} + \mathbf{v}_{J}$
  $\mathbf{c}_{i} = \mathbf{v}_{i}\times\, \mathbf{v}_{J}$
  $\mathbb{M}_{i}^{A} = \mathbb{M}_{i}^{\mathcal{M}}$ ▷ *Initialise articulated-body inertia*
  ▷ *Initialise articulated-body bias forces* ◁
  ${}^{i}\mathbf{X}_{0} = {}^{i}\mathbf{X}_{\lambda(i)}\, {}^{\lambda(i)}\mathbf{X}_{0}$
  ${}_{i}\mathbf{X}^{W} = ({}^{i}\mathbf{X}_{0}\, {}^{0}\mathbf{X}_{W})^{-\top}$
  $\mathbf{p}_{i}^{A} = \mathbf{v}_{i}\bar{\times}^{*}\, \mathbb{M}_{i}\, \mathbf{v}_{i} - {}_{i}\mathbf{X}^{W}\, {}_{W}\mathbf{f}_{i}^{ext}$
for $i = N_{B}^{\mathcal{M}}$ **to** 1 **do** ▷ *Pass 2*
  ▷ *Compute intermediate results* ◁
  $\mathbf{U}_{i} = \mathbb{M}_{i}^{A}\mathbf{S}_{i}$
  $D_{i} = \mathbf{S}_{i}^{\top}\mathbf{U}_{i}$
  $u_{i} = \tau_{i} - \mathbf{S}_{i}^{\top}\mathbf{p}_{i}^{A}$
  ▷ *Compute the articulated-body inertia and bias forces of this link...* ◁
  $\mathbb{M}_{i}^{a} = \mathbb{M}_{i}^{A} - \mathbf{U}_{i}\mathbf{U}_{i}^{\top} D^{-1}$
  $\mathbf{p}_{i}^{a} = \mathbf{p}_{i}^{A} + \mathbb{M}_{i}^{a}\mathbf{c}_{i} + \mathbf{U}_{i}D_{i}^{-1}u_{i}$
  if $\lambda(i) \neq 0$ or floating **then** ▷ *... and propagate them to the parent*
    $\mathbb{M}_{\lambda(i)}^{A} = \mathbb{M}_{\lambda(i)}^{A} + {}^{i}\mathbf{X}_{\lambda(i)}^{\top}\, \mathbb{M}_{i}^{a}\, {}^{i}\mathbf{X}_{\lambda(i)}$
    $\mathbf{p}_{\lambda(i)}^{A} = \mathbf{p}_{\lambda(i)}^{A} + {}^{i}\mathbf{X}_{\lambda(i)}^{\top}\mathbf{p}_{i}^{a}$
if fixed **then** ▷ *Set world gravity*
  $\bar{\mathbf{a}}_{0} = -\, {}^{0}\mathbf{X}_{W}\, {}^{W}\mathbf{a}_{g}$
else ▷ *Consider base acceleration without gravity*
  $\bar{\mathbf{a}}_{0} = -(\mathbb{M}_{0}^{A})^{-1}\mathbf{p}_{0}^{A}$
for $i = 1$ **to** $N_{B}^{\mathcal{M}}$ **do** ▷ *Pass 3*
  ▷ *Propagate link accelerations and compute joint accelerations* ◁
  $\mathbf{a}_{i}^{a} = {}^{i}\mathbf{X}_{\lambda(i)}\, \bar{\mathbf{a}}_{\lambda(i)} + \mathbf{c}_{i}$
  $\ddot{s}_{i} = D^{-1}(u_{i} - \mathbf{U}^{\top}\, \mathbf{a}_{i}^{a})$
  $\mathbf{a}_{i} = \mathbf{a}_{i}^{a} + \mathbf{S}_{i}\ddot{s}_{i}$
${}^{W}\mathbf{a}_{W,B} = 0$
if floating **then** ▷ *Add gravitational effects*
  ${}^{W}\mathbf{a}_{W,B} = {}^{W}\mathbf{X}_{0}\, \bar{\mathbf{a}}_{0} + {}^{W}\mathbf{a}_{g}$
outputs $\left(\ddot{\mathbf{s}}, {}^{W}\mathbf{a}_{W,B}\right)$

8.1.8 Soft-contacts Algorithm

Most of the algorithms listed in this chapter accept a matrix ${}_W\mathbf{f}_{\mathcal{C}}^{ext} \in \mathbb{R}^{N_B^{\mathcal{M}} \times 6}$ containing the 6D forces applied to each link of the model. These external forces are the sum of two components: the forces that could be applied to the links by the user, and the forces corresponding to active contacts with the terrain surface.

Algorithm 5 provides the implementation of the soft-contacts model introduced in Section 7.3 to compute the contact force corresponding to each collidable point of $\mathcal{M}$. Assuming the knowledge of the location of each collidable point w.r.t. the frame of their associated link, we can compute their position ${}^W\mathbf{p}_{cp}$ and mixed linear velocity ${}^{C[W]}\boldsymbol{v}_{W,C}$ with forward kinematics from the pair $(\mathbf{q}, \boldsymbol{\nu})$. The algorithm takes the following inputs:

- ${}^W\mathbf{p}_{cp} \in \mathbb{R}^3$: the position of the contact point in world coordinates;
- ${}^W\dot{\mathbf{p}}_C \in \mathbb{R}^3$: the linear velocity of the contact point, that matches the linear component of the mixed velocity ${}^{C[W]}\mathbf{v}_{W,C} \in \mathbb{R}^6$ of frame C;
- ${}^W\mathbf{m} \in \mathbb{R}^3$: the compounded tangential deformation of the terrain associated to the contact point;
- $K, D \in \mathbb{R}^+$: the parameters of the spring-damper model used for both the normal and tangential force calculation;
- $\mu \in \mathbb{R}^+$: the coefficient of friction of the contact point;
- $\mathcal{H} : \mathbb{R}^2 \to \mathbb{R}$: a function returning the terrain height z_T at given (x, y) coordinates;
- $\mathcal{S} : \mathbb{R}^2 \to \mathbb{R}^3$: a function returning the normal of the surface $\hat{\mathbf{n}}$ at given (x, y) coordinates[35];

and provides the following outputs:

- ${}_W\mathbf{f}_{cp} \in \mathbb{R}^6$: the 6D force computed by the soft-contacts model;
- ${}^W\dot{\mathbf{m}} \in \mathbb{R}^3$: the derivative of the tangential deformation of the terrain material associated to the contact point.

35 Under the assumption of smooth terrain, an approximation of $\mathcal{S}$ could be calculated from $\mathcal{H}$, i.e. $\mathcal{S}(x, y) = f\left(\mathcal{H}(x, y)\right)$

In order to optimise performance, the algorithm can be vectorised to process all the collidable points belonging to the model.

Algorithm 5 Soft contact

inputs $\left({}^{W}\mathbf{p}_{cp}, {}^{W}\dot{\mathbf{p}}_{C}, {}^{W}\mathbf{m}, K, D, \mu, \mathcal{H}, \mathcal{S}\right)$
$x_{cp}, y_{cp}, z_{cp} = {}^{W}\mathbf{p}_{cp}$
$z_T = \mathcal{H}(x_{cp}, y_{cp})$
${}^{W}\dot{\mathbf{m}} = -(K/D)\,{}^{W}\mathbf{m}$ ▷ *Material relaxation dynamics*
${}_{W}\mathbf{f}_{cp} = \mathbf{0}_6$
▷ *Compute normal force* ◁
$\hat{\mathbf{n}} = \mathcal{S}(x_{cp}, y_{cp})$
$\mathbf{h} = [0, 0, z_T - z_{cp}]^{\top}$
$\delta = \max(0,\ \mathbf{h} \cdot \hat{\mathbf{n}})$
$\dot{\delta} = -\,{}^{W}\dot{\mathbf{p}}_{C} \cdot \hat{\mathbf{n}}$
$\boldsymbol{f}_{\perp} = \max\left(0,\ \sqrt{\delta}\left(K\delta + D\dot{\delta}\right)\right)\hat{\mathbf{n}}$
▷ *Compute tangential force* ◁
$\boldsymbol{f}_{\parallel} = \mathbf{0}_3$
if $\mu \neq 0$ and $z_{cp} < z_T$ **then**
    $\boldsymbol{v}^{\perp} = \left({}^{W}\dot{\mathbf{p}}_{C} \cdot \hat{\mathbf{n}}\right)\hat{\mathbf{n}}$
    $\boldsymbol{v}^{\parallel} = {}^{W}\dot{\mathbf{p}}_{C} - \boldsymbol{v}^{\perp}$
    $\boldsymbol{f}_{\parallel} = -\sqrt{\delta}(K\,{}^{W}\mathbf{m} + D\,\boldsymbol{v}^{\parallel})$ ▷ *Compute sticking force*
    ${}^{W}\dot{\mathbf{m}} = \boldsymbol{v}_{\parallel}$
    $f_{cone} = \mu\|\boldsymbol{f}_{\perp}\|$
    if $\left\|\boldsymbol{f}_{\parallel}\right\| > f_{cone}$ **then** ▷ *Compute slipping force*
        $\boldsymbol{f}_{\parallel} = \left(f_{cone}/\left\|\boldsymbol{f}_{\parallel}\right\|\right)\boldsymbol{f}_{\parallel}$
        ${}^{W}\dot{\mathbf{m}} = -(\boldsymbol{f}_{\parallel} + K\sqrt{\delta}\,{}^{W}\mathbf{m})/(D\sqrt{\delta})$
▷ *Compute 6D contact force in the world frame* ◁
${}_{W}\mathbf{f}_{cp} = {}_{W}\mathbf{X}^{C}[(\boldsymbol{f}_{\perp} + \boldsymbol{f}_{\parallel})^{\top}, \mathbf{0}_3^{\top}]^{\top}$
outputs $\left({}_{W}\mathbf{f}_{cp}, {}^{W}\dot{\mathbf{m}}\right)$

8.2 JAXSIM

The previous sections described how we can model and simulate a floating-base robot locomoting in an environment, and efficiently compute relevant quantities useful, for example, for model-based control. For robot-learning applications, and particularly the domain of RL, the execution of the dynamics on CPU often represents the major bottleneck of the training pipeline. We can identify two principal sources affecting performance in such setup:

1. CPUs can only run concurrently –albeit with great speed– a small number of threads, that usually match the double of their physical cores;
2. When a training pipeline exploits GPUs for optimising its function approximators (in the form of NNs, for example), the data sampled from the simulation has to be moved from the CPU to the GPU, incurring into overheads related to the data transport.

In this section, we describe how to overcome these two limitations by introducing JAXsim, a highly scalable rigid-body simulator of floating-base systems for robot locomotion research. We start presenting the main features currently implemented. Then, we assess its execution performance, its simulation accuracy, and its parallelization capabilities when executed on CPU and GPUs.

8.2.1 Features

JAXsim is the first simulator in reduced coordinates implemented entirely in Python that can be executed seamlessly on either CPU or modern hardware accelerators like GPUs and TPUs. In Table 8.1, we compared some of the JAXsim properties with other recent simulators briefly introduced in Section 4.2. Here below a complete list of features of JAXsim:

- Physics engine implemented in reduced coordinates;
- Forward Euler, Semi-implicit Euler, and RK4 integrators;
- Collision detection between points associated with the model's collision shapes and a customisable smooth terrain surface;

- Continuous soft-contacts model to compute interaction forces without introducing friction cone approximations;
- Support of the complete set of link's inertial parameters;
- Support of revolute, prismatic, and fixed joints;
- Support of SDF and URDF model descriptions[36];
- Implemented in plain Python using the JAX framework for fast development and readability;
- Possibility to maximise runtime performance by JIT-compiling Python code and executing physics transparently on CPUs, GPUs, and TPUs;
- Seamless integration with JAX's auto-vectorization capability for parallelizing simulation steps on hardware accelerators;
- High-level API for computing model-based kinematics and dynamics quantities based on the algorithms presented in Section 8.1 and notation introduced in Chapter 2.

Remark 8.2.1 (AD support)**.** Being implemented with JAX, gradients of the simulation step w.r.t. any model or simulation parameters could theoretically be computed. However, at the current stage of development, the AD support of the algorithms and the simulator has yet to be properly assessed, and left for future work. □

36 https://github.com/ami-iit/rod

Table 8.1: Comparison of modern physics engines similar to JAXsim. [*] JAXsim is developed with a differentiable framework, but this functionality has to be finalised.

Software	Language	Coordinates	CPU	GPU	TPU	Differentiable	Ground Collisions	Body Collisions	High-level APIs	Open Source
Tiny Differentiable Simulator [Heiden et al., 2021]	C++	Reduced	✓	✓		✓	✓	✓		✓
Nimble Physics Werling et al. [2021]	C++	Reduced	✓			✓	✓	✓	✓	✓
Nvidia ISAAC Makoviychuk et al. [2021]	C++	Reduced	✓	✓			✓	✓	✓	
Dojo Howell et al. [2022]	Julia	Maximal	✓			✓	✓			✓
Brax Freeman et al. [2021]	Python	Maximal	✓	✓	✓	✓	✓	✓		✓
JAXsim	Python	Reduced	✓	✓	✓	[*]	✓		✓	✓

Table 8.2: Specifications of the settings in which the benchmarks are executed.

Specification	Laptop	Workstation
Intel CPU	i7 7700HQ	Xeon Silver 4214
Nvidia GPU	GeForce GTX 1050 Mobile	Quadro RTX 6000
CUDA cores	640	4608
Operating system	Ubuntu 20.04	Ubuntu 20.04
Nvidia driver	510.73.05	510.73.05
CUDA	11.2.2	11.2.2
cuDNN	8.2.1.32	8.2.1.32
JAX	0.3.14	0.3.14
Gazebo Sim	Fortress	-
DART	6.10	-

8.2.2 Benchmarks

In this section, we evaluate the characteristics of JAXsim by performing some benchmarks to assess its accuracy, performance, and scalability properties. Table 8.2 reports the specifications of the machines in which the benchmarks have been executed. The laptop is used for all the benchmarks, while the workstation only for the scalability assessment.

Accuracy

In this section, we evaluate the simulation accuracy of JAXsim through the astronaut simulation [Erez et al., 2015; Howell et al., 2022], in which a robot model is simulated in a world without gravity. The experiment is composed of two phases, both of which can be illustrated by Figure 8.1. In the first phase, the model is actuated for 1 second with random torques starting from a configuration with zero velocity and, therefore, zero momentum. The simulation is performed with all lossy components (like joint friction) disabled, therefore, due to momentum conservation, the momentum should remain zero over the entire horizon. In the second phase, the model evolves for 100 seconds without actuation from the configuration reached in the previous phase. Also in this case, since there is no loss, the total mechanical energy of the system (i.e. the sum of the kinetic and potential energies defined in Equation (2.15)) should remain constant over the entire horizon due to energy conservation.

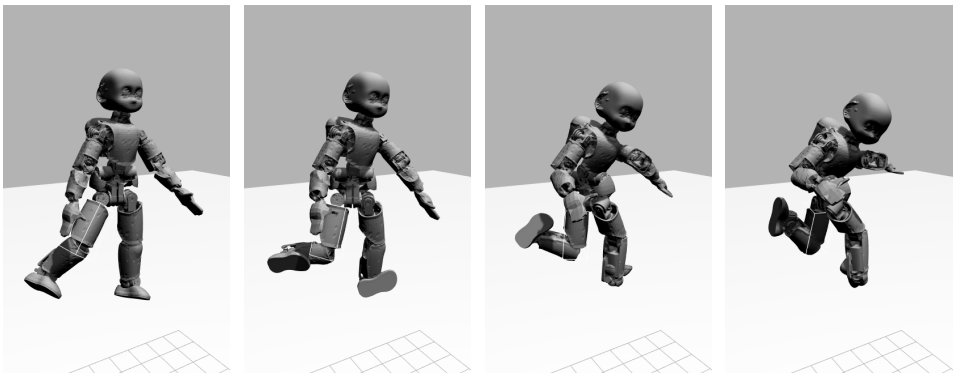


Figure 8.1: Sequence showing four time instants of the astronaut simulation used for both the assessments of the momentum conservation and the energy conservation. In the former experiment, the joints are actuated with random torques, while in the latter, the joints are not actuated and evolve in open-loop accordingly to their initial velocity.

We perform this experiment using the model of the iCub robot with a configuration characterised by 23 of its DoFs. The experiment is executed at different integration step sizes and with both the forward Euler and RK4 integrators. We compare the results against Gazebo Sim running with its default DART physics engine. The two simulators use the same model description of iCub. The random actuation applied in the first phase is generated by sampling a torque for each joint from a uniform distribution $\tau \sim \mathcal{U}(-0.500, 0.500)\ Nm$. The torques trajectory is generated offline at the lowest simulated frequency and up-sampled with zero-hold for higher frequencies. In order to get a fair comparison between JAXsim and Gazebo, we enable the optional 64-bit support of JAX. The results of the momentum conservation and energy conservation experiments are shown, respectively, in Figure 8.2 and Figure 8.3.

In the results of the first phase, it can be seen that in almost all configurations the 6D momentum drifts from its initial value. Particularly, all configurations show a more substantial drift in conjunction with larger integration steps. The configuration of JAXsim with the RK4 integrator can outperform the other ones, showing acceptable drifts with steps below 20 ms. Considering as acceptable a drift of 0.1% after 1 s of simulation, also the forward Euler integration scheme stepping at 0.001 s falls within this limit.

In the results of the second phase, a similar drift could be noticed in all the tested configurations. Also in this case, JAXsim with the RK4 integrator yields the lowest drift comparable to machine precision. The results of Gazebo

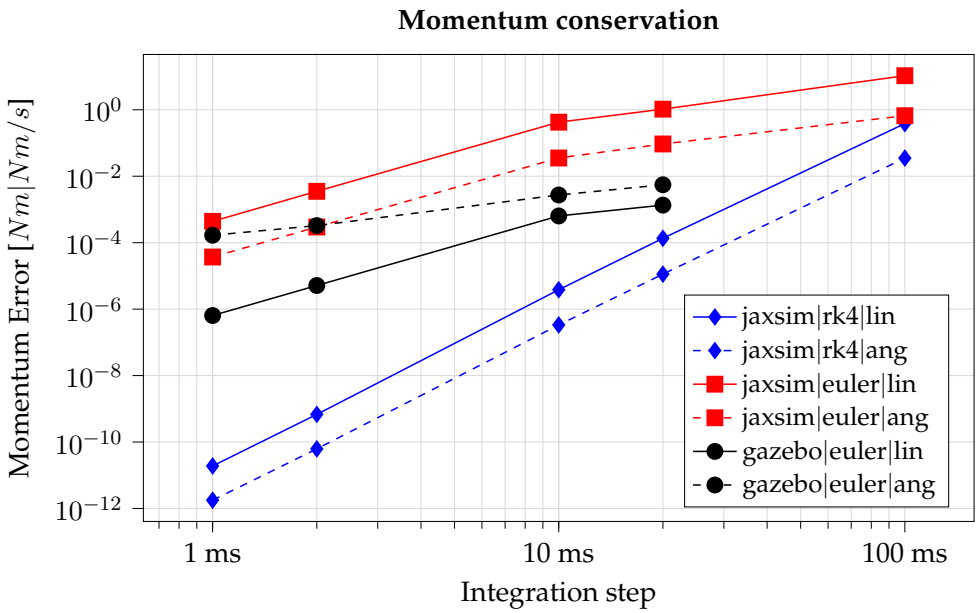


Figure 8.2: Momentum drift after 1 simulated second of the iCub humanoid robot in a world without gravity, starting from a configuration with zero velocity and applying random joint forces. The plot shows the norm of the linear and angular component of the momentum computed in inertial-fixed coordinates. Gazebo Sim failed to simulate the configuration with the 100 ms step.

Sim and JAXsim with forward Euler integrating with a step size of 0.001 s are comparable, while Gazebo Sim performs better in the configuration with the larger 0.010 s step. The drift of JAXsim with the larger integration step 0.010 s and the RK4 integrator are comparable to the other configurations integrating with 0.001 s steps, highlighting the benefits of higher-order integration schemes already discussed in Section 7.4.

An excessive increase of the integration step is generally not advisable also for reasons not strictly related to the integration accuracy. In fact, the contact detection routine has no control over the stepping strategy, and excessively large steps could result in substantial instantaneous penetration depths that, depending on the selected terrain stiffness, could produce unrealistically big contact forces.

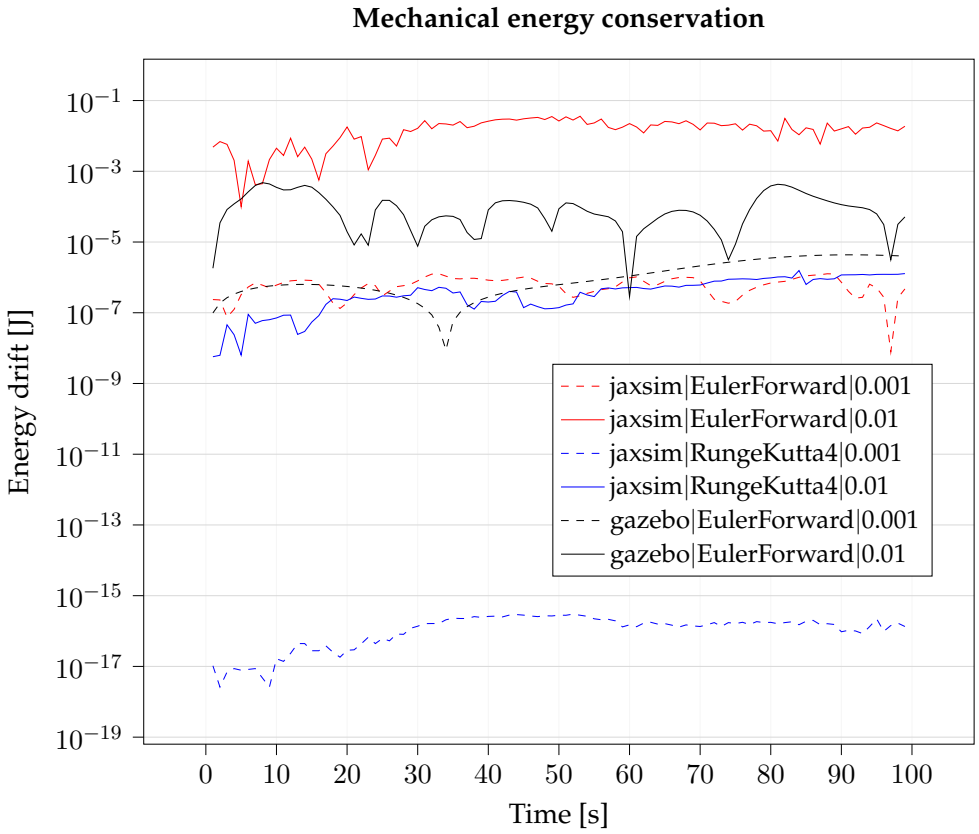


Figure 8.3: Mechanical energy drift over 100 simulated seconds of the iCub humanoid robot in a world without gravity, starting from a configuration with a given generalized velocity.

Performance

In this section, we evaluate the runtime performance of the Rigid Body Dynamics Algorithms implemented in JAXsim against a state-of-the-art C++ implementation included in Pinocchio [Carpentier et al., 2015; Carpentier et al., 2019].

We compute the total time necessary to compute the output of the CRBA, RNEA, and ABA algorithms. To assess how much the execution time is affected by the topology of the simulated robot, we consider three different robot models with an increased number of DoFs. In particular, we benchmark the 9 DoFs fixed-base Panda manipulator from Franka Emika, the 12 DoFs quadruped ANYmal C from ANYbotics, and the humanoid iCub from IIT. Before running any test with JAXsim, we call each algorithm one time so that it can get JIT compiled since in this experiment we are interested in their runtime performance rather than compilation time. For each bar in the plot, we first compute the mean time taken by a single run of 1000 executions of the algorithms, and then report the average over 10 of these runs. Figure 8.4 shows the resulting mean, where the variance of the 10 runs has been omitted since it's negligible for all algorithms.

The results are comparable for all three tested robot models. The execution time of the JAXsim algorithms compared to the implementation of Pinocchio are about 10 times higher when executed on CPU, and 100 times higher when executed on GPU. These numbers are expected since Pinocchio algorithms are implemented entirely in C++ and have been optimised for almost a decade. Regardless, for all three robot models, the JAXsim algorithms executed on CPU do not exceed $250\ \mu s$, making them compatible with a real-time loop running at a target rate of 1 kHz, in which a model-based controller might have to compute the mass matrix $M(\mathbf{q})$ through CRBA and the bias forces $h(\mathbf{q}, \boldsymbol{\nu})$ through RNEA. Instead, the GPU execution of a single instance of the algorithms exceeds 1 ms in most cases, making them incompatible with real-time usage.

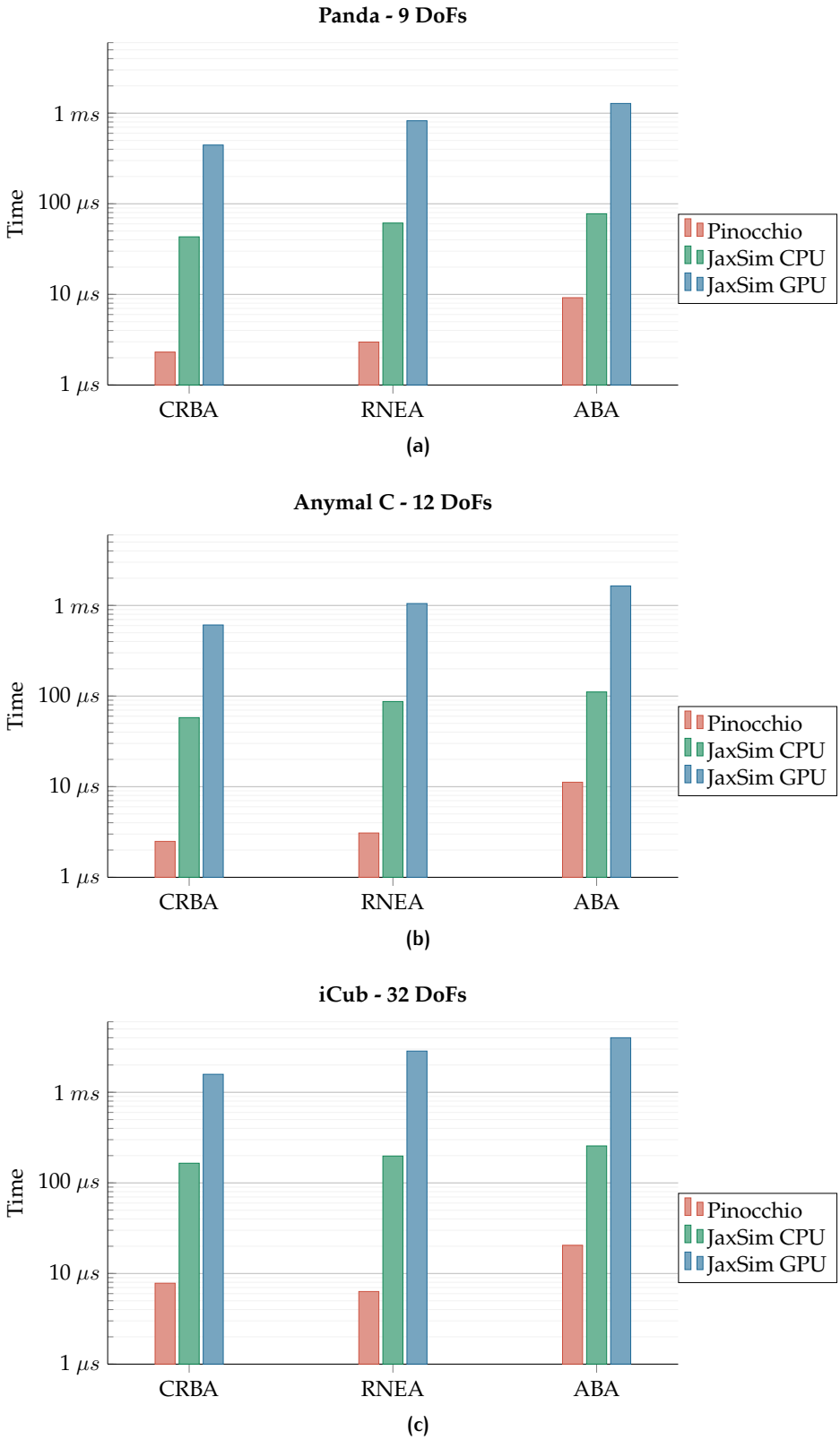


Figure 8.4: Benchmark of the RBDAs implemented in JAXsim against those implemented in Pinocchio. (a) shows the results of the 9-DoFs fixed-base Panda manipulator from Franka Emika, (b) the results of the 12-DoFs quadruped ANYmal C from ANYbotics, and (c) the results of the 32-DoFs humanoid iCub from IIT. The execution of JAXsim's algorithms run on average 10 times slower than Pinocchio when executed on CPU, and 100 times when executed on GPU.

Scalability

In the previous test, we assessed the performance of a single execution of the benchmarked algorithms implemented in JAXsim. The major strengths of JAXsim appear when the characteristics of hardware accelerators are exploited, performing multiple executions in parallel. In this test, we evaluate the scalability of JAXsim by increasing the number of parallel instances exploiting the auto-vectorization capabilities of JAX. Instead of testing parallel calls of the RBDAs, we consider a more practical scenario of a 1 ms simulation step with the forward Euler integration scheme. We benchmark the performance on the 23 DoFs model of the iCub humanoid with 8 collidable points corresponding to the vertices of the two boxes that model its feet collision shapes. We compute the *equivalent* RTF of the parallel integration, which consists of the ratio between the total simulated time and the time it took to compute it. For example, if the parallel integration of 10 models takes 1 ms, the equivalent RTF is 10. A higher RTF corresponds to a better sampling efficiency. This test is performed on the same laptop as the previous tests, and also on a workstation with a more powerful GPU, whose specifications are reported in Table 8.2. For each point in the plot, we first compute the mean time of a single run over 100 integration steps, and then report the average over 10 of these runs. The results of the CPU and GPU executions are shown in Figure 8.5, where the variance of the 10 runs has been omitted since it is negligible for all executions.

On the laptop setup, the integration step on CPU starts with a RTF greater than 1 already with a single instance. A single CPU core is able to reach a RTF of about 5 with 16 parallel models, showing some benefits of parallel integration also on this type of hardware. With more than 16 models, increasing the number of models does not give any benefit to the equivalent RTF as the execution time grows linearly with the number of models. The integration step of GPU starts with a lower RTF of 0.11. This effect is expected since a single GPU core is typically less powerful than a CPU core. However, the parallel integration on GPU is able to scale without showing any overhead until 128 models. The equivalent RTF on GPU peaks at a value of about 19 between 512 and 1024 parallel models. On this hardware, the 512-1024 range is justified by the number of CUDA cores equal to 640.

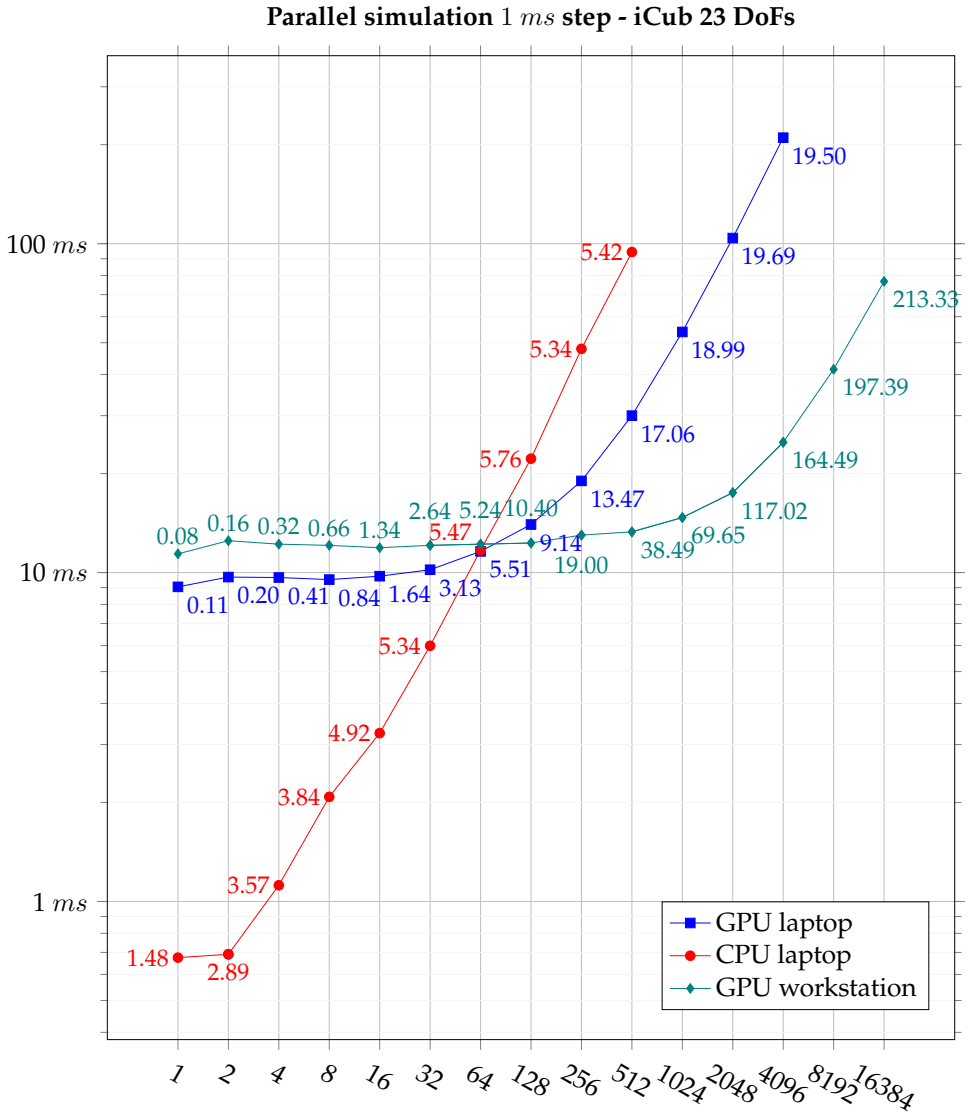


Figure 8.5: Comparison of parallelization performance of a simulation step executed on CPU and GPUs. The simulated models are 23 DoFs replicas of the iCub humanoid robot, and the simulation step length is 1 ms with a forward Euler scheme. The time taken by the CPU scales mostly linearly with the number of simulated models, while the GPUs are able to exploit the parallel capability almost up to their CUDA cores (640 on the laptop, 4608 on the workstation). For each sample, we show the equivalent RTF. The CPU cannot scale well over the number of models when integrating more than 16 replicas. Instead, the GPUs show an interval that depends on their parallellization capabilities in which the execution time is not affected significantly by the number of integrated models. Also on GPUs, however, the performance start degrading when the number of integrated models exceeds the available CUDA cores.

A similar trend is observed regarding the workstation GPU. Although being more modern and powerful, the execution is slightly slower on this high-end GPU compared to the laptop, starting from a RTF of 0.08. In this case, the GPU has 4608 CUDA cores. It can be noticed that the region in which the performance on GPU scales horizontally without additional overhead is much larger, and start flattening in the range 4096-8192, which also in this case is close to the number of CUDA cores of the setup. The peak RTF of this high-end GPU is about 200.

8.3 VALIDATION

In this section, we perform a validation of JAXsim for generating synthetic data for robot learning. We develop an environment exposing the `gym.Env` interface with JAXsim, and show the sampling performance that can be reached by stepping a large number of parallel environments on a laptop GPU. We then plug the vectorized environment in a RL pipeline for training a policy with the PPO algorithm. Finally, for presenting evidence that the data generated through the methods proposed in this thesis can be used in an out-of-distribution setting, we execute and evaluate the policy on a comparable dynamics simulated this time with Mujoco.

The out-of-distribution validation is also known in the literature as sim-to-sim [Salvato et al., 2021; Muratore et al., 2022; Bellegarda et al., 2021; Du et al., 2021]. Given that one of the assumptions for an effective transfer is the availability of a robust policy trained in the original setting, we consider as target task to learn the swing-up of an underactuated cartpole. This task is similar to the canonical benchmark of cartpole balancing [Brockman et al., 2016], in which the pole starts from an almost balanced configuration and the actions space is discrete (selecting either a positive of negative constant force to apply to the cart). This cartpole balancing, however, is excessively simple, and it is not really representative of the typical problems in robotics, usually characterised by continuous action spaces. The swing-up task makes the policy learning much more difficult by starting the episodes with an arbitrary pole position (also pointing down). This diversity makes a big difference since it requires the policy to be considerably long-sighted, to the extent to learn to perform some initial swing to build up momentum before attempting to perform a proper balancing.

All experiments presented in this section have been executed on a laptop whose specifications are reported in Table 8.3.

Table 8.3: Specifications of the machine used to execute the validation experiments.

Specification	Value
Intel CPU	i7-10750H
Nvidia GPU	GeForce GTX 1650 Ti Mobile
CUDA cores	1024
Operating system	Ubuntu 22.04
Nvidia driver	530.41.03
CUDA	11.2.2
cuDNN	8.8.0.121
JAX	0.3.15
Mujoco	2.3.5

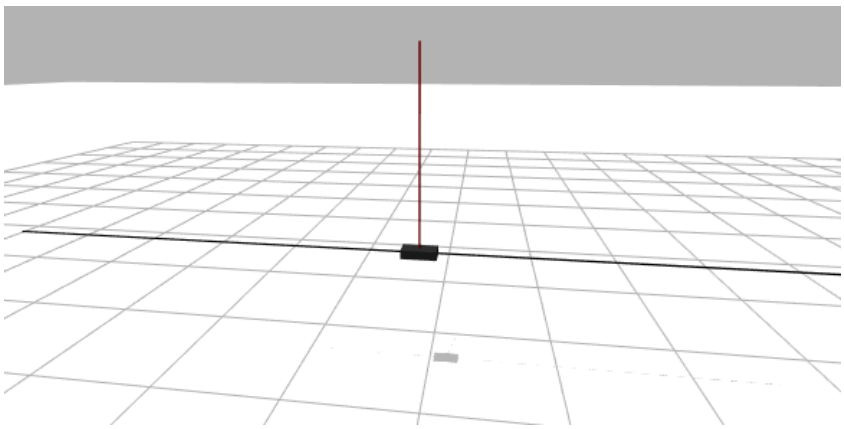


Figure 8.6: Illustration of the cartpole model in the $\theta = d = 0$ configuration.

Table 8.4: Properties of the environment implementing the cartpole swing-up task.

Property	Value
Integrator	Semi-implicit Euler
Integrator step	0.0005 s
Environment step	0.050 s
Control frequency	20 Hz
Action dimension	1
Observation dimension	4
Action space	$[-50, 50]$ N
Maximum episode steps	200
Parallel environments	512
Equivalent RTF	24.38

Table 8.5: PPO parameters for the the cartpole swing-up environment.

Parameter	Value
Discount rate γ	0.95
Clip parameter ϵ	0.1
Target KL divergence	0.025
Learning rate α	0.0003
GAE parameter λ	0.9
Batch size	2560
Minibatch size	256
Number of SGD epochs	10

8.3.1 Sampling Experience for Robot Learning

The cartpole is a fixed-base model composed by a pole (shaped as a long and thin cylinder) connected through an un-actuated revolute joint to a cart (shaped as a box). The cart can move along a track, whose displacement is simulated as a prismatic joint. The linear force corresponding to this prismatic joint is the control input of the system. Figure 8.6 reports a visualisation of the cartpole model.

OBSERVATION The observation of the system is composed of the position and velocity of both joints. If $\theta \in [-\pi, \pi]$ rad is the angle of the un-actuated revolute joint (where $\theta = 0$ is a balanced pole), $d \in [-2.5, 2.5]$ m the displacement of the prismatic joint, and $\omega = \dot{\theta}$ the pole velocity, we can define the observation as $\mathbf{o} = (d, \dot{d}, \theta, \omega) \in \mathbb{R}^4$.

ACTION The action of the system is the linear force $a = f \in [-50, 50]\,\mathrm{N}$ corresponding to the joint moving the cart. It's worth noting that the action is continuous and its magnitude is not large enough to accelerate the cart and bring the pole in vertical position without swinging first.

ENVIRONMENT DETAILS The environment simulates the physics with JAXsim. We use a semi-implicit Euler integrator with a step of $500\,\mu\mathrm{s}$. After the agent sets an action through the exposed `gym.Env` interface, we step the environment for $0.050\,\mathrm{s}$, therefore performing 100 physics steps each time. The resulting control frequency, that is the update rate at which the policy applies its actions, results equal to $20\,\mathrm{Hz}$. The environment is implemented as a continuous control task with termination only occurring if the observation is outside its space. In absence of termination, episodes are truncated after 200 steps. All environment properties are reported in Table 8.4.

REWARD The reward function used to learn the swing-up task is the following:

$$
\begin{aligned}
&r_t(s_t, a_t, s_{t+1}) = \\
&\quad r_{\text{alive}} + r_{\text{balance}} - 0.001\, c_{\text{action}} - 0.1\, c_{\text{vel}} - 0.5\, c_{\text{displacement}},
\end{aligned}
$$

where r_{alive} is set to 1.0 when the environment is not terminal and 0 otherwise, $r_{\text{balance}} = \cos\theta$ rewards the pole to be in a balanced state (characterised by $\theta = 0$), $c_{\text{action}} = \|\tau\|$ penalises large actuated forces, $c_{\text{vel}} = \|\dot{\mathbf{s}}\|$ penalises large joint velocities, and $c_{\text{displacement}} = |d|$ penalises the cart to be far from the middle point of the track.

AGENT NETWORKS We want to train a policy with RL capable of bringing the pole to a balancing position from any state belonging to the observation space, and maintain the balance as long as possible. The agent is composed of two neural networks corresponding to the actor –the policy– and the critic –the return–, having two hidden layers with 512 neurons each with ReLU activation function. The input layer of both networks has a size of 4 (the observation dimension). The critic network has an output layer with just one dimension corresponding to the return, correctly bootstrapped from the value function is case of episode truncation. The

actor network encodes a univariate Gaussian distribution (the environment action is a scalar), therefore it has one output corresponding to the distribution mean and has a free parameter part of the optimisation parameters corresponding to the logarithm of the distribution's variance. The variance is initialised with the value of $\log \sigma = \log(0.05)$. The neural networks are optimised with Adam [Kingma et al., 2017] using a learning rate $\alpha = 0.0003$.

ALGORITHM For the same reasons explained in Chapter 6, we train the policy with the PPO algorithm introduced in Section 3.4.3, configured with the clip parameter $\epsilon = 0.1$. We estimate the return from the advantage computed with GAE, introduced in Section 3.4.2 as $R_t = \hat{A}_t^{\text{GAE}} + V_t$, configured with $\lambda = 0.9$ and a discount rate $\gamma = 0.95$. We use the PPO implementation of `stable-baselines3` [Raffin et al., 2021] that, instead of using a KL penalty, stops the training epochs when the approximated KL divergence exceeds a given value. All the PPO parameters are reported in Table 8.5.

SAMPLING We optimise the policy by acquiring five samples from 512 parallelised environments running on GPU, resulting to an equivalent RTF of about 25 on the machine used to run the experiments. The number of environments has been selected by choosing the best equivalent RTF of the JIT-compiled vectorized `gym.Env.step` through a grid-search in the $N_{\text{envs}} = 2^p$, $p \in \{1, 2, 3, \dots, 12\}$ range. The collected batch of trajectories, containing a total of 2560 samples and equivalent to about 2 minutes of experience, is then split in 10 minibatches of 256 samples. We perform 10 optimisation epochs on the same batch of trajectories, that can be interrupted earlier in case the approximated KL divergence w.r.t. the distribution corresponding to the previous policy exceeds 0.025. Before optimising the NNs of the agent, the collected observations and rewards are normalized by computing a running mean and standard variance. Finally, in order to obtain a more robust policy, we inject some Gaussian noise with zero mean and $\sigma = 0.05$ to the actions before being applied to the environment.

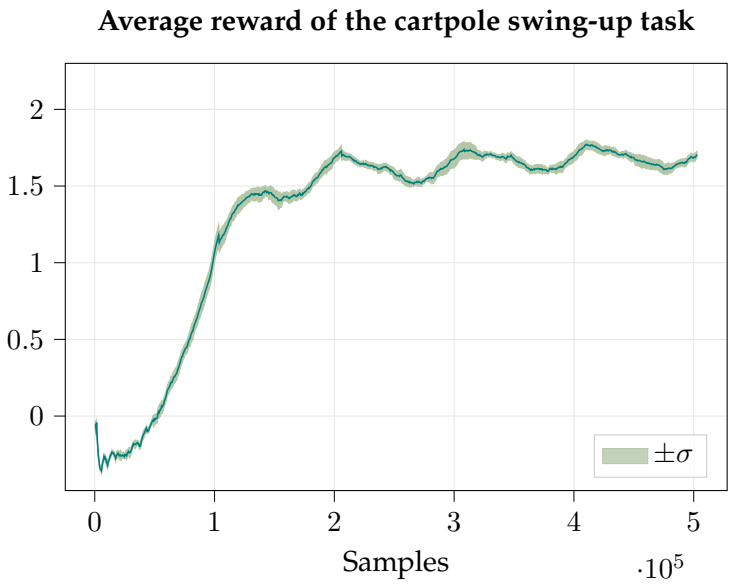


Figure 8.7: Learning curves of the cartpole swing-up task. The plot reports the mean and standard deviation of the average rewards $\hat{r}_t^{(k)}$ computed over $k = 10$ different training executions. For each individual training, the average reward $\hat{r}_t$ in the considered parallel setting is computed by averaging at each time step the 512 rewards received from the vectorized `gym.Env.step`.

Figure 8.7 shows the learning curve of 10 trainings initialised with different seeds. The policy is able to learn effective swing-up and balancing behaviours in 500000 steps, corresponding to about 7 hours of equivalent experience. On the machine used to run this experiment, each training runs for approximately 15 minutes. The reward grows mostly monotonically with a limited variance, showing that the chosen parameters ensure stable policy updates, preventing optimisation steps too large that would destroy the previously obtained performance.

Table 8.6: Mujoco properties used for the sim-to-sim evaluation of the trained cartpole swing-up policy. Refer to the official documentation at `https://mujoco.readthedocs.io` for a detailed explanation of the options.

Property	Value
`timestep`	0.001
`integrator`	`RK4`
`solver`	`Newton`
`iterations`	100
`contype`	0

8.3.2 Sim-to-sim Policy Transfer

In this section, we attempt to evaluate the trained policy in an out-of-distribution setting. This setting could represent any environment that differs from the one where the policy has been trained. It serves as evidence that it's possible to deploy a policy obtained from training over synthetic data generated efficiently by a parallel simulator into an equivalent environment running on a different runtime. In particular, this experiment can be seen as a sim-to-sim policy transfer.

Similarly to Section 7.5.3, we use the Mujoco simulator as out-of-distribution setting. We translated the URDF of the cartpole model loaded in JAXsim in the training environment to an equivalent MJCF that can be imported in Mujoco. Then, we included in the same file the configuration of the physics engine, whose parameters are reported in Table 8.6. The chosen parameters expose a simulation characterised by the same control rate (20 Hz), but in this case the physics is simulated in a different simulator using an integrator of a different family and different constraint solver.

The Mujoco environment is only used for producing the state-action trajectory τ from a given initial observation $\mathbf{o}_0$, where the action is obtained by performing inference of the trained policy. In order to perform a proper exploitation of the policy, we sample deterministically by taking the inferred mean of the Gaussian distribution described by the policy, i.e. $a_t = \mu_{\boldsymbol{\theta}}(\mathbf{o}_t)$.

The first evaluation we perform in this setting is a comparison between the swing-up trajectory obtained by running the policy in the training environment simulated with JAXsim and the out-of-distribution environment simulated with Mujoco. In order to obtain a meaningful comparison, we consider as initial

observation the cart resting in the center of its track and the pole pointing down, both with zero velocity, i.e. $\mathbf{o}_0 = (0, 0, \pi, 0)$. The two trajectories are shown in Figure 8.8, where it can be noticed that the policy performance are comparable in both simulators. The policy is able to succeed in the swing-up task and the resulting trajectories in both simulators are almost identical.

As second evaluation, starting from the same initial observation $\mathbf{o}_0$ considered in the first evaluation, we assess the policy swing-up performance on different variations of the cartpole environment simulated with the out-of-distribution Mujoco. In particular, we modify some physical parameters and evaluate whether the learned policy is robust enough to succeed in the task. In the first variation, we double the mass of both the cart and the pole, taking care to compute the new 3×3 inertia matrices corresponding to the primitive shape of the bodies. In the second variation, in addition to the increased masses, we include joint damping, i.e. a frictional force proportional to the joint velocity that opposes the motion direction. For a revolute joint, its contribution is $\tau_{\text{damping}} = -k_v \dot{\theta}$. Figure 8.9 reports the curves simulated in the out-of-distribution Mujoco simulator using out-of-distribution model parameters. Despite the out-of-distribution environment, it can be seen that the policy learned in a highly parallel setting simulated with JAXsim on GPU succeeds in swing-up task. As it can be expected, the performance are affected by the change of parameters. For example, the policy is able to reach the balancing state by using only one swing in the setting with nominal parameters. Instead, in the two variations, the policy needs two swings.

It's worth noting that the policy has been trained only using the nominal parameters. Often, in order to obtain more robust policies, the inertial parameters of the simulated model become part of domain randomization. In our case, we did not randomize any parameter.

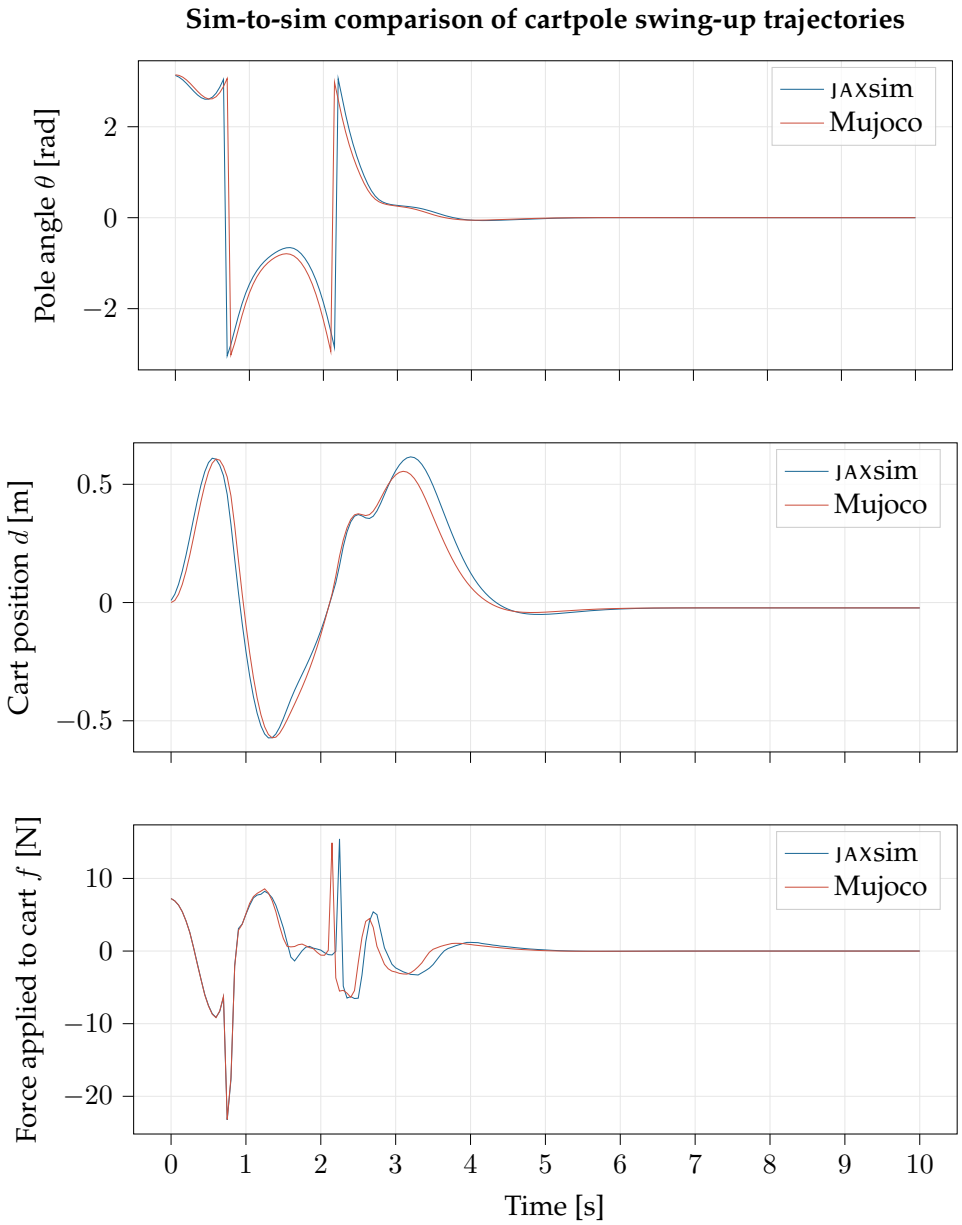


Figure 8.8: Sim-to-sim comparison of the trajectories obtained by exploiting the swing-up policy learned in a JAXsim environment. The JAXsim curves correspond to an in-distribution setting, where the policy is evaluated in the same simulator that generated training data. Instead, the Mujoco curves correspond to an out-of-distribution setting, where the policy is evaluate in a simulator different from the one that generated training data. Note that θ, due to its range, is projected in the $[-\pi, \pi]$ range.

Cartpole swing-up trajectories with out-of-distribution parameters and simulator

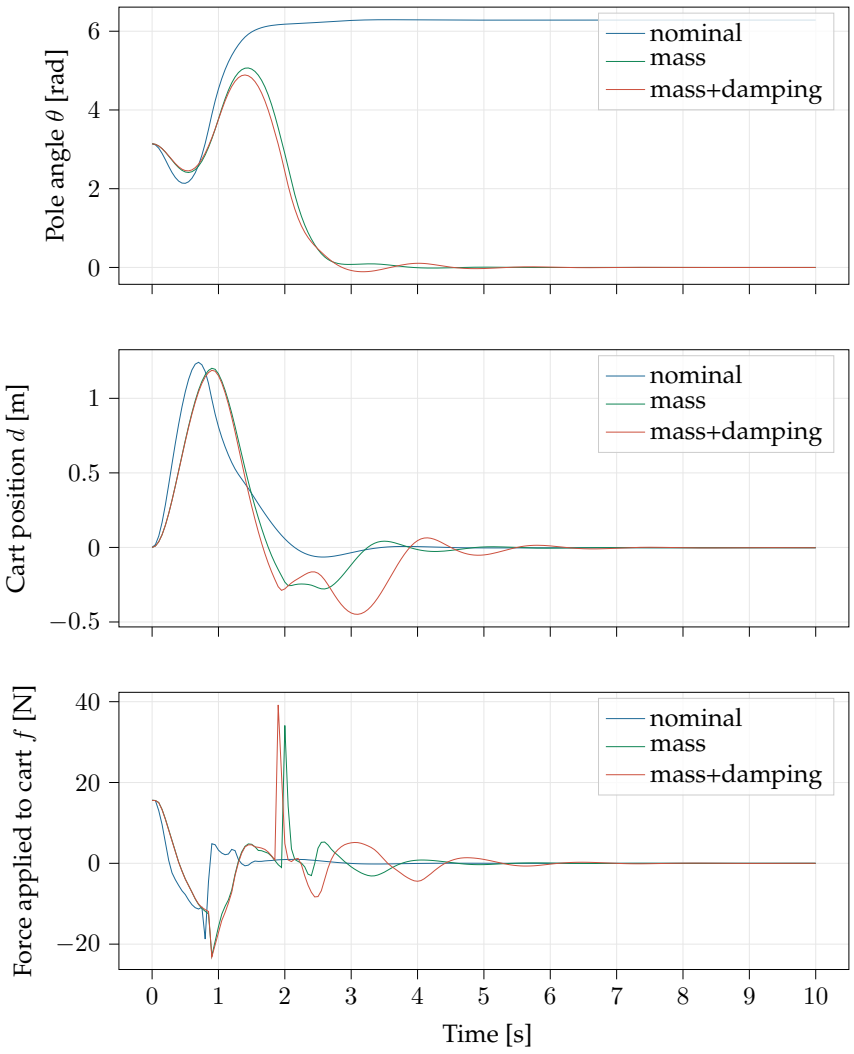


Figure 8.9: Trajectories of the cartpole swing-up policy acting on the out-of-distribution environment simulated in Mujoco. The *nominal* curves are obtained by running the policy on a cartpole model having nominal masses of both the cart and the pole, and with no joint damping. The *mass* curves show the obtained trajectories with the model having the masses of both bodies multiplied by $2x$. The *mass+damping* curves are generated in a setting that extends the *mass* one by also considering for both joints a damping with $k_v = 0.015$. Note that in this case, we removed the bounds of the pole angle, showing more clearly the number of swings used by the policy to reach the balancing state.

8.4 CONCLUSIONS

In this chapter, we proposed JAXsim, a new physics engine capable of executing multibody simulations on modern hardware accelerators. JAXsim simulates the dynamics of free-floating mechanical systems through a reduced coordinates approach, therefore guaranteeing that joint constraints are never violated. Regarding the interaction with the environment, it is currently capable of detecting collisions between a smooth ground surface not necessarily planar and points belonging to the collision shapes of the links. Being implemented in reduced coordinates, it allows to efficiently compute all terms of the EoM that are fundamental for model-based control at any simulated step. We have validated its integration accuracy, obtaining either comparable or better results than Gazebo Sim, depending on the adopted integration scheme. We also benchmarked the RBDAs performance, finding that JIT-compiled Python code on CPU runs 10 times slower compared to a state-of-the-art C++ implementation, remaining in any case compatible with real-time usage. Nonetheless, the best characteristics of JAXsim emerge when modern hardware accelerators are exploited in highly parallel computations. We have shown that it can reach a RTF of 20 on a laptop and 200 on a workstation. Applications requiring sampling experience with high throughput such as those characterising RL research, would benefit the most from these performances. Furthermore, generating physics data directly in the same device where function approximators are optimised would remove any overhead related to the data transfer from the CPU. We intend to investigate these directions in future activities. Finally, we trained a policy to solve a continuous control problem simulated with JAXsim, providing details on the parallelization level that can be reached for sampling synthetic data in a characteristic robot learning application. Then, to provide evidence that it's possible to deploy policies obtained from training by sampling from highly parallel simulators, we performed a sim-to-sim transfer and evaluated the policy performance on a out-of-distribution simulator and physical parameters. The obtained RL setting is much simpler compared to one adopted in Chapter 6. The possibility to parallelize sampling on hardware accelerators removes the need to rely on distributed settings running on a cluster of machines, that is difficult to create, maintain, and handle. A single-

process application can deploy both sampling and NNs on the same GPU, with no overhead related to network transport. Beyond being easier to deploy in a HPC setting, this approach may lead to a faster development and prototyping.

JAXsim still presents multiple limitations in this first version. Firstly, the shown performance were obtained in a setting where no exteroception was necessary. Integrating basic rendering functionality would surely negatively affect them. Furthermore, collisions between links are not yet supported, limiting the adoption for robot manipulation. Finally, it does not yet allow enforcing additional dynamic constraints like joint limits, and closed kinematic chains. To conclude, although the automatic differentiation capability provided by the JAX framework has not yet been thoroughly tested with the JAXsim algorithms, its combination with the smooth dynamics given by the contact-aware state-space representation opens up exciting research directions that will be explored in the future.

Not all those who wander are lost.

— J. R. R. Tolkien.

EPILOGUE

SUMMARY

Part i introduced the reader to the fundamental concepts and notation used throughout this thesis. In particular, Chapter 1 introduced robotic simulators, describing their main components and properties, and provided a brief description of the technologies that enabled the activities related to this thesis. Chapter 2, after an initial overview of frame kinematics and rigid body dynamics, derived the EoMs of a rigid multibody system describing the dynamics of floating-base robots. Then, Chapter 3 provided a bird-eye view of RL. With the goal of formulating the theory behind PPO, which is the RL algorithm used in Part ii, this second background chapter first defined all the elements necessary to formalise the RL problem, then it described the taxonomy of the available algorithms available to compute a solution, and finally provided the theory of policy optimisation.

The thesis continued with Part ii, describing the contributions to knowledge of this thesis. The first two chapters analysed the challenges of creating robotic environments for RL research for the aim of sampling experience. Motivated by the fragmented state of the existing frameworks providing environments for robotics and the desire to develop environments that could run on either simulated or real-time runtimes without the need for major refactoring, Chapter 5 presented Scenario and Gym-Ignition. Scenario (SCENe interfAces for Robot Input/Output) provides unified interfaces for entities part of a scene like World and Model. Gym-Ignition, instead, provides abstraction layers of different components that structure a robotic environment, like the Task and the Runtime. The combination of the two projects enables a modular development of environments for robotics, where the decision-making logic could be implemented only once and executed transparently

on all the supported runtimes, either simulated or in real-time. Furthermore, the modular structure of Gym-Ignition isolates the boilerplate code of all environments sharing a specific runtime, reducing code duplication and speeding up the development by only focusing on the definition of decision-making logic. We validated the overall learning pipeline in Chapter 6, where we introduced the definition of the decision-making logic for training a RL policy capable of synthesising whole-body push recovery strategies for the humanoid robot iCub. The emergence of the policy's final behaviours was guided by a reward shaping approach that, while remaining within the model-free category from the RL perspective, enabled the inclusion of prior information computed from the model description of the robot.

To conclude, the last chapters of Part ii analysed the problem of the high computational cost associated with the sampling of synthetic data from rigid multibody simulations. Motivated by the long time required for training the policy presented in Chapter 6, and the increased interest by the robot learning community to offload and parallelize computation on hardware accelerators, we proposed a simulation architecture to maximise the sampling performance of simulated data for robot locomotion applications. Towards this aim, in Chapter 7, we introduced a continuous state-space representation modelling the contact-aware dynamics of floating-base robots. Assuming the knowledge of the terrain surface's smooth profile, and utilising a soft-contacts model to compute the interaction forces for point-surface collisions, we obtained a continuous ODE system that can be integrated numerically to simulate its dynamical evolution over time. Finally, exploiting this representation, in Chapter 8 we presented JAXsim, a new physics engine capable to execute simulations of floating-base robots entirely on hardware accelerators like GPUs and TPUs. We adapted widely used Rigid Body Dynamics Algorithms to run in this context, formulating them with the notation introduced in Chapter 2. Their definition was unified to be applicable also on fixed-based robots. We have benchmarked, among other properties, the scalability of JAXsim when executed on GPUs, showing its potential when experience sampling can be performed with large parallelism.

DISCUSSIONS, LIMITATIONS, AND FUTURE WORK

To conclude the thesis, we provide final discussion pointing of each contribution to knowledge, outlining pitfalls and future work directions.

Chapter 5: Reinforcement Learning Environments for Robotics

The proposed framework, composed of Scenario and Gym-Ignition, was initially aimed to be applicable to both simulated and real-world robots. We started the development of the simulation backend in late 2018, when we decided to leverage the Gazebo Sim general-purpose simulator that, at that time, did not even reach its first major release. Considering the status of RL research for robotics in the same period, which was mainly performed either in PyBullet or the closed-source Mujoco, we saw the potential to obtain a complete, open-source, and actively maintained simulator to base our research. The possibility to extend the simulator through custom plugins, the support of integrating third-party physics engines, the new architecture allowing to use it as a library through programmatic calls, and the knowledge and infrastructure already developed for its predecessor Gazebo Classic, were all appealing to the direction we envisioned for our research. The Gazebo Scenario backend enabled us to experiment with our first RL problem described in Chapter 6, whose concluding details will be discussed in the next section. The framework remains a valid alternative to the other options freely available online, especially nowadays since Gazebo Sim reached almost feature parity with its predecessor. Upstream activities are currently ongoing to bridge Gazebo Sim with Nvidia Omniverse and the Isaac simulator, expecting significant advances particularly regarding rendering capabilities. Considering the limitations of newer domain-specific solutions like those presented in Chapter 8, general-purpose simulators will remain the standard when perception is required for the aim of sampling synthetic data for RL-based robotics research. Nonetheless, our Gazebo Scenario backend still has limitations. Our APIs do not yet support the sensor interfaces of Gazebo Sim, therefore their data can only be gathered from the network, a solution that does not ensure the reproducibility of sampled data. Considering

the generic Scenario project, instead, after we refocused the research project to rely mainly on simulations, the activities of a real-time backend to communicate with YARP-based robots like the iCub remained on hold since. The possibility of running the same environment implementation on both simulated and real-world robots is an interesting solution for sim-to-real research, and it will be considered for future activities. From the Gym-Ignition point of view, instead, limitations are less impactful. The framework is generic enough for developing most categories of robotic environments. What is still missing is a collection of canonical examples and benchmarks similar to the robotic environments provided by OpenAI. Access to a new collection would help the community to have a more diverse range of environments that, thanks to Gazebo Sim, could run on any physics engine supported by the simulator.

Chapter 6: Learning from scratch exploiting robot models

In this chapter, we proposed a scheme to train a policy with RL for balancing a simulated humanoid robot in the presence of external disturbances, sampling synthetic data from the framework proposed in the previous chapter. We adopted a process based on reward shaping to guide the state space exploration throughout the training phase. Starting from a simple reward structure, we tried to address undesired behaviour by iteratively adding new terms, until its final form. We decided to control most of the DoFs of the iCub robot, acknowledging that the kinematics is highly redundant, and the policy optimisation could have stalled to local optima. Parameter tuning is paramount for these applications, and details are too often left out from research discussions. For tuning our reward function (for each term, its weights and kernel parameters), we adopted a heuristic method in which we analysed the learning curves of individual terms of the reward, tuning the sensitivity of the corresponding kernel if the algorithm was not improving its performance, and then balancing the weight to obtain the desired trade-off among all the reward terms. Nonetheless, each experiment was days long on powerful workstations, and parameter tuning resulted in a long and, at some point, quite extenuating process. Much work also went into the training infrastructure, leveraging a cluster of machines

and implementing the proper experiment deployment, with logging and synchronisation support.

Beyond the training process, our results also show limitations, particularly when possible future sim-to-real applications are considered. Our control architecture relies on a policy providing velocity references, which are integrated and given as inputs to independent PID joint controllers, generating the final joint torques actuated by the simulator. Beyond being difficult to tune with performance comparable to those of the real-world counterpart, the low-level PID controllers present a trade-off between accuracy and compliance. In our experiment, the position PID controllers resulted in a stiff robot, that together with rigid contacts, limited the emergence of a natural, smooth, and more human-like behaviour. We think two different directions can be considered for improvements. As a first direction, PID controllers could be replaced by a single whole-body controller that typically exploits the information of the model dynamics. It would consider the entire robot as a whole and possibly reduce the differences from real robots. However, they are more complex to design, more computationally expensive, and making them work reliably on all the robot configurations allowed by the state space of the MDP is difficult. A second direction could consider different policy outputs (joint torques, velocities, positions, etc.), which means a different nature of the high-level trajectory. Previous studies [Peng et al., 2017; Reda et al., 2020] on the subject were not conclusive, and the choice of the action space remains highly related to the type of the decision making logic. In both cases, a change in policy output and its corresponding action space could have major effects on exploration. Common RL algorithms for continuous actions usually learn a distribution from which actions are sampled during the training process for exploration purposes. The typical choice is a multivariate Gaussian distribution, but it does not play well with bounded spaces. Other studies have found other distributions like the Beta [Chou et al., 2017] that might behave better in this context.

Studies investigating robot control with RL could be considered part of the bigger umbrella of trajectory optimisation, in which applications like push-recovery and locomotion are instances. As reviewed in Section 4.1, applications targeting quadrupeds already managed to successfully target real robots. The situation for real bipeds, instead, from when our research project started

in 2018, didn't progress noticeably. The gap between the latest simulated results [Peng et al., 2022] and the few ones targeting real-world robots [Castillo et al., 2021; Li et al., 2021; Rodriguez et al., 2021; Bloesch et al., 2022] is still wide. Common characteristics of these studies are either the usage of large simulations with techniques of domain randomization and imitation learning, or the usage of curriculum learning and the introduction of non-ideal effects in simulation like actuation delays.

In view of this discussion, for locomotion purposes, we think that imitation learning could provide a suitable trade-off between exploration guidance, avoidance of local minima, and learning stability. Future activities will focus on integrating novel motion generation techniques [Viceconte et al., 2022] within RL environments. Practitioners working in RL applied to robotics must be aware that this field suffers from most of the challenges that have been identified by the community [Dulac-Arnold et al., 2021], and can learn from their successes and failures [Ibarz et al., 2021].

Chapter 7: Contact-aware Multibody Dynamics

This chapter proposed a state-space representation modelling the dynamics of a floating-base robot in contact with a ground surface, which can be integrated numerically over time to compute the trajectory of the system. Being formulated in reduced coordinates, the system dynamics is forced to evolve enforcing the joint constraints. While being quite generic and providing a compact formulation applicable to any articulated structure, it presents different limitations. The collision detection corresponding to the contact model considers only points rigidly attached to the links and a smooth ground surface. Despite being able to approximate generic collision shapes like arbitrarily complex meshes, the cost of collision detection grows linearly with the number of considered points. For what regards primitive shapes, better methodologies based on geometrical properties exists, and would be much more efficient (think, as an example, the simple case of a sphere, that now has to be approximated with dozens or hundreds of points). With these methodologies in place, the implemented logic could also be extended to detect collisions between bodies, enabling the applicability to neighbour domains like

robot manipulation. Another direction of possible improvements regards the integration schemes. As shown in the benchmarks of Chapter 8, the forward Euler scheme is the fastest (and simpler) integrator, but its accuracy is not as good as what is achieved by the RK4 scheme. Other common methods often used in similar physics engines, like semi-implicit and implicit schemes, could provide performance similar to RK4 at a cost comparable to forward Euler. Finally, the state-space representation, in its default form, does not allow to enforce bounds to state space variables, useful for example to enforce joint position limits. Common workarounds involve the introduction of penalty-based continuous forces [Xu et al., 2022], or introducing exogenous variables mapping the unbounded joint positions to a bounded range. Future work will address all these limitations, extending the supported use-cases for the physics engine proposed in Chapter 8.

Chapter 8: Scaling Rigid Body Simulations

In this last chapter, we presented and benchmarked JAXsim, a new physics engine in reduced coordinates capable of executing rigid-body simulations on modern hardware accelerators, including CPUs, GPUs, and TPUs. It is based on JAX, a new framework developed by Google that, thanks to its properties, is experiencing a quick and wide adoption in diverse domains. Its key features are the possibility to compile kernels developed in Python with a JIT approach, auto-vectorization support, NumPy compatibility, and high-order AD support. All these features are inherited by JAXsim, whose algorithms can be executed with all the benefits of the underlying technology. JAXsim, however, also inherits the limitations of JAX. The need to compile code at its first execution could take several minutes, depending on the complexity of the logic and the active hardware acceleration. The GPU and TPU backends of JAX are much more optimised compared to the CPU backend, that nonetheless, despite longer compiling time, is able to run code faster than plain Python. We have not yet optimised our algorithms aggressively to improve compilation time, especially because we expect to see soon caching support of compiled code. Regarding scalability, our benchmarks considered two GPUs with 640 and 4608 CUDA cores. The potential of executing code on these types of hardware,

when the problem permits parallelization, becomes year after year more appealing considering the technological progress. For example, the newest Nvidia GPU architecture comes with more than 10 thousand CUDA cores. Furthermore, JAX also supports multi-GPU configurations, but setting them up is more complicated and requires small changes to the implemented logic. To conclude the discussion on the features related to JAX, our algorithms are not yet compatible with its AD capability. The activities to assess the support and implement AD support are ongoing, and we expect they will enable us to start investigating all the new emerging methodologies involving differentiable simulations.

Other activities planned for the near future involve enhancing the RL stack built over JAXsim. The combination of an environment interfacing with JAXsim and RL algorithms implemented in JAX results in a single application whose data never leaves the hardware accelerator. Therefore, beyond the sampling performance of parallel simulations, the complete pipeline would also prevent the data transfer overhead that is always present when some computation has to happen on CPUs. In Section 8.3, we provided a continuous control validation by sampling from a cartpole environment simulated entirely on GPU. However, we used an existing PPO implementation not developed in JAX, therefore it was not possible to compile in JIT the entire collection of the batch but only an individual parallelized sample. Future work will continue this activity, extending the investigation to contact-rich locomotion problems. Finally, we would like to embed these environments in Gym-Ignition, creating a new JAXsim Scenario component, so that all the benefits of future real-time backends could be applicable on JAXsim experiments. Towards this goal, Gym-Ignition should switch to the upcoming functional version of `gym.Env` that has been recently proposed upstream.

Machines are so stupid;
if you instruct them to perform a task perfectly, they will do it.

CONCLUSIONS

In modern times, having access to a large amount of data and massive computational power has become paramount for succeeding in any context related to machine learning and, more generally, artificial intelligence. In this thesis, we considered the challenging domain of robotics, focusing on locomotion applications for humanoid robot planning and control. Throughout the chapters, we explored how modern technology could help us generate synthetic data considering the domain-specific characteristics and limitations of the targeted application. We believe that, particularly in this domain, the infrastructure plays such an essential role to the extent that those possessing a large enough technological advantage would stand out with ease. We hope to have helped readers reaching this final paragraph understand the challenges and research directions that are still necessary to obtain the next generation of robots capable of seamlessly operating around us. Simulation technology is evolving rapidly. We believe that future progress that will inevitably characterise the next decades is going to set aside the real breakthroughs long awaited by all robotic practitioners. We can not wait to keep contributing with all the enthusiasm that characterised the activities carried out for this thesis.

BIBLIOGRAPHY

Achiam, Josh (2018). *Spinning Up in Deep RL*. URL: https://spinningup.openai.com (visited on 23/06/2022).

Aftab, Zohaib, Thomas Robert and Pierre-Brice Wieber (Nov. 2012). 'Ankle, hip and stepping strategies for humanoid balance recovery with a single Model Predictive Control scheme'. In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pp. 159–164.

Andrle, Michael S and John L Crassidis (2013). 'Geometric Integration of Quaternions'. In: *Journal of Guidance, Control, and Dynamics*, p. 10.

Atkeson, Christopher G and Stefan Schaal (1997). 'Robot Learning From Demonstration'. In: p. 9.

Azad, Morteza and Roy Featherstone (2010). 'Modeling the contact between a rolling sphere and a compliant ground plane'. In: *Proceedings of ACRA 2010*.

Azad, Morteza, Valerio Ortenzi, Hsiu-Chin Lin, Elmar Rueckert and Michael Mistry (Nov. 2016). 'Model estimation and control of compliant contact normal force'. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 442–447.

Baydin, Atılım Gunes, Barak A Pearlmutter, Alexey Andreyevich Radul and Jeffrey Mark Siskind (2018). 'Automatic Differentiation in Machine Learning: a Survey'. In: p. 43.

Belbute-Peres, Filipe de A, Kevin A Smith, Kelsey R Allen, Joshua B Tenenbaum and J Zico Kolter (2018). 'End-to-End Differentiable Physics for Learning and Control'. In: p. 12.

Bellegarda, Guillaume and Quan Nguyen (Mar. 2021). 'Robust High-speed Running for Quadruped Robots via Deep Reinforcement Learning'. In: *arXiv:2103.06484 [cs, eess]*. URL: http://arxiv.org/abs/2103.06484 (visited on 27/05/2021).

Benbrahim, Hamid and Judy A. Franklin (Dec. 1997). 'Biped dynamic walking using reinforcement learning'. In: *Robotics and Autonomous Systems* 22.3-4, pp. 283–302.

Bishop, Christopher M. (2006). *Pattern recognition and machine learning*. Information science and statistics. New York: Springer.

Bloesch, Michael et al. (Jan. 2022). 'Towards Real Robot Learning in the Wild: A Case Study in Bipedal Locomotion'. In: *Proceedings of the 5th Conference on Robot Learning*. PMLR, pp. 1502–1511.

Blondel, Mathieu, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa and Jean-Philippe Vert (May 2022). *Efficient and Modular Implicit Differentiation*. Tech. rep. arXiv:2105.15183. arXiv. URL: http://arxiv.org/abs/2105.15183 (visited on 01/06/2022).

Bradbury, James et al. (2018). *JAX: composable transformations of Python+NumPy programs*. URL: https://github.com/google/jax (visited on 07/03/2022).

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang and Wojciech Zaremba (June 2016). 'OpenAI Gym'. In: *arXiv:1606.01540 [cs]*. URL: http://arxiv.org/abs/1606.01540 (visited on 22/05/2019).

Bullo, Francesco and Andrew D. Lewis (2004). *Geometric Control of Mechanical Systems: Modeling, Analysis, and Design for Simple Mechanical Control Systems*. Vol. 49. Springer Science & Business Media.

Carpentier, Justin and Nicolas Mansard (June 2018). 'Analytical Derivatives of Rigid Body Dynamics Algorithms'. In: *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation. URL: http://www.roboticsproceedings.org/rss14/p38.pdf (visited on 18/05/2021).

Carpentier, Justin, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse and Nicolas Mansard (Jan. 2019). 'The Pinocchio C++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives'. In: *2019 IEEE/SICE International Symposium on System Integration (SII)*, pp. 614–619.

Carpentier, Justin, Florian Valenza and Nicolas Mansard (2015). *Pinocchio: fast forward and inverse dynamics for poly-articulated systems*. URL: https://stack-of-tasks.github.io/pinocchio.

Castillo, Guillermo A., Bowen Weng, Wei Zhang and Ayonga Hereid (Sept. 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)*. Prague, Czech Republic: IEEE, pp. 5136–

5143. URL: https://ieeexplore.ieee.org/document/9636467/ (visited on 19/04/2022).

Cellier, François E. and Ernesto Kofman (2006). *Continuous system simulation*. New York: Springer.

Choi, HeeSun et al. (2021). 'On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward'. In: *PNAS*. URL: https://dspace.mit.edu/handle/1721.1/139616 (visited on 06/05/2022).

Chou, Po-Wei, Daniel Maturana and Sebastian Scherer (2017). 'Improving Stochastic Policy Gradients in Continuous Control with Deep Reinforcement Learning using the Beta Distribution'. In: p. 10.

Collins, J., S. Chand, A. Vanderkop and D. Howard (2021). 'A Review of Physics Simulators for Robotic Applications'. In: *IEEE Access* 9, pp. 51416–51431.

Coumans, Erwin and Yunfei Bai (2016). *Pybullet, a python module for physics simulation in robotics, games and machine learning*.

Dafarra, Stefano et al. (Oct. 2018). 'A Control Architecture with Online Predictive Planning for Position and Torque Controlled Walking of Humanoid Robots'. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9.

Delhaisse, Brian, Leonel Rozo and Darwin G Caldwell (2019). 'PyRoboLearn: A Python Framework for Robot Learning Practitioners'. In: p. 11.

Dong, Hao, Zihan Ding and Shanghang Zhang, eds. (2020). *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Singapore: Springer Singapore. URL: http://link.springer.com/10.1007/978-981-15-4095-0 (visited on 24/02/2022).

Du, Yuqing, Olivia Watkins, Trevor Darrell, Pieter Abbeel and Deepak Pathak (May 2021). 'Auto-Tuned Sim-to-Real Transfer'. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1290–1296.

Duburcq, Alexis, Fabian Schramm, Guilhem Boéris, Nicolas Bredeche and Yann Chevaleyre (Mar. 2022). *Reactive Stepping for Humanoid Robots using Reinforcement Learning: Application to Standing Push Recovery on the Exoskeleton Atalante*. Tech. rep. arXiv:2203.01148. arXiv. URL: http://arxiv.org/abs/2203.01148 (visited on 21/05/2022).

Dulac-Arnold, Gabriel, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal and Todd Hester (Mar. 2021). 'An empirical in-

vestigation of the challenges of real-world reinforcement learning'. In: *arXiv:2003.11881 [cs]*. URL: http://arxiv.org/abs/2003.11881 (visited on 13/04/2022).

Erez, T., Y. Tassa and E. Todorov (2015). 'Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX'. In.

Featherstone, Roy (2008). *Rigid body dynamics algorithms*. Springer.

Feng, Siyuan, Eric Whitman, X Xinjilefu and Christopher G. Atkeson (Nov. 2014). 'Optimization based full body control for the atlas robot'. In: *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 120–127.

Ferigo, Diego, Silvio Traversaro, Giorgio Metta and Daniele Pucci (2020). 'Gym-Ignition: Reproducible Robotic Simulations for Reinforcement Learning'. In: *IEEE/SICE International Symposium on System Integration (SII)*, pp. 885–890.

Freeman, C. Daniel, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch and Olivier Bachem (June 2021). 'Brax – A Differentiable Physics Engine for Large Scale Rigid Body Simulation'. In: *arXiv:2106.13281 [cs]*. URL: http://arxiv.org/abs/2106.13281 (visited on 09/12/2021).

Friedland, Bernard (2005). *Control system design: an introduction to state-space methods*. Dover ed. Mineola, NY: Dover Publications.

Frostig, Roy, Matthew James Johnson and Chris Leary (2018). 'Compiling machine learning programs via high-level tracing'. In.

Gangapurwala, Siddhant, Mathieu Geisert, Romeo Orsolino, Maurice Fallon and Ioannis Havoutis (May 2021). 'Real-Time Trajectory Adaptation for Quadrupedal Locomotion using Deep Reinforcement Learning'. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE, pp. 5973–5979. URL: https://ieeexplore.ieee.org/document/9561639/ (visited on 19/04/2022).

Gilardi, G. and I. Sharf (Oct. 2002). 'Literature survey of contact dynamics modelling'. In: *Mechanism and Machine Theory* 37.10, pp. 1213–1239. URL: https://linkinghub.elsevier.com/retrieve/pii/S0094114X02000459 (visited on 02/03/2022).

Gillen, Sean and Katie Byl (Mar. 2022). 'Leveraging Reward Gradients For Reinforcement Learning in Differentiable Physics Simulations'. In: *arXiv:2203.02857 [cs, eess]*. URL: http://arxiv.org/abs/2203.02857 (visited on 13/04/2022).

Gros, Sebastien, Marion Zanon and Moritz Diehl (Dec. 2015). 'Baumgarte stabilisation over the SO(3) rotation group for control'. In: *2015 54th IEEE Conference on Decision and Control (CDC)*. Osaka: IEEE, pp. 620–625. URL: http://ieeexplore.ieee.org/document/7402298/ (visited on 02/03/2022).

Gullapalli, V., J.A. Franklin and H. Benbrahim (Feb. 1994). 'Acquiring robot skills via reinforcement learning'. In: *IEEE Control Systems Magazine* 14.1, pp. 13–24.

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel and Sergey Levine (Aug. 2018). 'Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor'. In: *arXiv:1801.01290 [cs, stat]*. URL: http://arxiv.org/abs/1801.01290 (visited on 07/05/2020).

Harris, Charles R. et al. (Sept. 2020). 'Array programming with NumPy'. In: *Nature* 585.7825, pp. 357–362. URL: https://www.nature.com/articles/s41586-020-2649-2 (visited on 01/06/2022).

Heess, Nicolas et al. (2017). 'Emergence of Locomotion Behaviours in Rich Environments'. In.

Heiden, Eric, David Millard, Erwin Coumans, Yizhou Sheng and Gaurav S. Sukhatme (May 2021). 'NeuralSim: Augmenting Differentiable Simulators with Neural Networks'. In: *arXiv:2011.04217 [cs]*. URL: http://arxiv.org/abs/2011.04217 (visited on 01/03/2022).

Hinton, Geoffrey E., Simon Osindero and Yee-Whye Teh (July 2006). 'A Fast Learning Algorithm for Deep Belief Nets'. In: *Neural Computation* 18.7, pp. 1527–1554. URL: https://direct.mit.edu/neco/article/18/7/1527-1554/7065 (visited on 15/04/2022).

Honglak Lee, Yirong Shen, Chih-Han Yu, G. Singh and A.Y. Ng (2006). 'Quadruped robot obstacle negotiation via reinforcement learning'. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* Orlando, FL, USA: IEEE, pp. 3003–3010. URL: http://ieeexplore.ieee.org/document/1642158/ (visited on 15/04/2022).

Howell, Taylor A., Simon Le Cleac'h, J. Zico Kolter, Mac Schwager and Zachary Manchester (Mar. 2022). 'Dojo: A Differentiable Simulator for Robotics'. In: *arXiv:2203.00806 [cs]*. URL: http://arxiv.org/abs/2203.00806 (visited on 13/04/2022).

Hwangbo, J., J. Lee and M. Hutter (Apr. 2018). 'Per-Contact Iteration Method for Solving Contact Dynamics'. In: *IEEE Robotics and Automation Letters* 3.2, pp. 895–902.

Hwangbo, Jemin, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun and Marco Hutter (Jan. 2019). 'Learning agile and dynamic motor skills for legged robots'. In: *Science Robotics*. (Visited on 20/05/2019).

Ibarz, Julian, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor and Sergey Levine (Jan. 2021). 'How to Train Your Robot with Deep Reinforcement Learning; Lessons We've Learned'. In: *The International Journal of Robotics Research*, p. 027836492098785. URL: http://arxiv.org/abs/2102.02915 (visited on 08/04/2021).

Innes, Mike, Alan Edelman, Keno Fischer, Chris Rackauckas, Elliot Saba, Viral B. Shah and Will Tebbutt (July 2019). 'A Differentiable Programming System to Bridge Machine Learning and Scientific Computing'. In: *arXiv:1907.07587 [cs]*. URL: http://arxiv.org/abs/1907.07587 (visited on 14/09/2019).

Ivaldi, S., J. Peters, V. Padois and F. Nori (Nov. 2014). 'Tools for simulating humanoid robot dynamics: A survey based on user feedback'. In: *IEEE-RAS International Conference on Humanoid Robots*.

Jaakkola, Tommi, Satinder P Singh and Michael I Jordan (1994). 'Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems'. In: *Advances in neural information processing systems*.

James, Stephen, Marc Freese and Andrew J. Davison (June 2019). 'PyRep: Bringing V-REP to Deep Robot Learning'. In: *arXiv:1906.11176 [cs]*. URL: http://arxiv.org/abs/1906.11176 (visited on 02/05/2022).

Jeong, Hyobin, Inho Lee, Jaesung Oh, Kang Kyu Lee and Jun-Ho Oh (Dec. 2019). 'A Robust Walking Controller Based on Online Optimization of Ankle, Hip, and Stepping Strategies'. In: *IEEE Transactions on Robotics* 35.6, pp. 1367–1386.

Juliani, Arthur, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar and Danny Lange (Sept. 2018). 'Unity: A General Platform for Intelligent Agents'. In: *arXiv:1809.02627 [cs, stat]*. URL: http://arxiv.org/abs/1809.02627 (visited on 03/05/2019).

Kajita, S., F. Kanehiro, K. Kaneko, K. Yokoi and H. Hirukawa (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)*. Vol. 1. Maui, HI, USA: IEEE, pp. 239–246. URL: http://ieeexplore.ieee.org/document/973365/ (visited on 21/05/2022).

Kim, Harin, Donghyeon Seo and Donghan Kim (Feb. 2019). 'Push Recovery Control for Humanoid Robot Using Reinforcement Learning'. In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pp. 488–492.

Kim, Taewoo, Minsu Jang and Jaehong Kim (July 2021). 'A Survey on Simulation Environments for Reinforcement Learning'. In: *2021 18th International Conference on Ubiquitous Robots (UR)*, pp. 63–67.

Kingma, Diederik P. and Jimmy Ba (Jan. 2017). *Adam: A Method for Stochastic Optimization*. URL: http://arxiv.org/abs/1412.6980 (visited on 26/05/2023).

Kober, Jens, J Andrew Bagnell and Jan Peters (2013). 'Reinforcement Learning in Robotics: A Survey'. In: *International Journal of Robotics Research*, p. 38.

Kober, Jens and Jan Peters (2008). 'Policy Search for Motor Primitives in Robotics'. In: *Advances in Neural Information Processing Systems*. Vol. 21. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2008/hash/7647966b7343c29048673252e490f736-Abstract.html (visited on 15/04/2022).

Koenig, N. and A. Howard (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) (IEEE Cat. No.04CH37566)*. Vol. 3. Sendai, Japan: IEEE, pp. 2149–2154. URL: http://ieeexplore.ieee.org/document/1389727/ (visited on 07/02/2019).

Kohl, N. and P. Stone (2004). 'Policy gradient reinforcement learning for fast quadrupedal locomotion'. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004.* New Orleans, LA, USA: IEEE, 2619–2624 Vol.3. URL: http://ieeexplore.ieee.org/document/1307456/ (visited on 19/04/2022).

Kolter, J Z, Pieter Abbeel and Andrew Y Ng (2007). 'Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion'. In.

Körber, Marian, Johann Lange, Stephan Rediske, Simon Steinmann and Roland Glück (Mar. 2021). 'Comparing Popular Simulation Environments in the

Scope of Robotics and Reinforcement Learning'. In: *arXiv:2103.04616 [cs]*. URL: http://arxiv.org/abs/2103.04616 (visited on 20/04/2022).

Lee, Jeongseok, Michael X. Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S. Srinivasa, Mike Stilman and C. Karen Liu (Feb. 2018). 'DART: Dynamic Animation and Robotics Toolkit'. In: *The Journal of Open Source Software*. (Visited on 09/08/2019).

Lee, Joonho, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun and Marco Hutter (Oct. 2020). 'Learning quadrupedal locomotion over challenging terrain'. In: *Science Robotics* 5.47. URL: https://www.science.org/doi/10.1126/scirobotics.abc5986 (visited on 21/05/2022).

Li, Zhongyu, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth and Koushil Sreenath (Mar. 2021). 'Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots'. In: *arXiv:2103.14295 [cs, eess]*. URL: http://arxiv.org/abs/2103.14295 (visited on 27/05/2021).

Liang, Eric, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan and Ion Stoica (2018). 'RLlib: Abstractions for Distributed Reinforcement Learning'. In: *arXiv*.

Liang, Jacky, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin and Dieter Fox (Oct. 2018). 'GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning'. In: (visited on 19/06/2019).

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver and Daan Wierstra (2016). 'Continuous Control with Deep Reinforcement Learning'. In: p. 14.

Lin, Long-Ji (1993). 'Reinforcement Learning for Robots Using Neural Networks'. PhD thesis. Carnegie Mellon University.

Lopez, Nestor Gonzalez, Yue Leire Erro Nuin, Elias Barba Moral, Lander Usategui San Juan, Alejandro Solano Rueda, Víctor Mayoral Vilches and Risto Kojcev (Mar. 2019). 'gym-gazebo2, a toolkit for reinforcement learning using ROS 2 and Gazebo'. In: *arXiv:1903.06278 [cs]*. URL: http://arxiv.org/abs/1903.06278 (visited on 18/03/2019).

Lovejoy, William S. (Dec. 1991). 'A survey of algorithmic methods for partially observed Markov decision processes'. In: *Annals of Operations Research* 28.1,

pp. 47–65. URL: http://link.springer.com/10.1007/BF02055574 (visited on 07/01/2023).

Lucchi, Matteo, Friedemann Zindler, Stephan Mühlbacher-Karrer and Horst Pichler (Nov. 2020). 'robo-gym – An Open Source Toolkit for Distributed Deep Reinforcement Learning on Real and Simulated Robots'. In: *arXiv:2007.02753 [cs]*. URL: http://arxiv.org/abs/2007.02753 (visited on 02/05/2022).

Maclaurin, Dougal, David Duvenaud and Ryan P Adams (2015). 'Autograd: Effortless Gradients in Numpy'. In.

Maki, B. E. and W. E. McIlroy (May 1997). 'The role of limb movements in maintaining upright stance: the "change-in-support" strategy'. In: *Physical Therapy* 77.5, pp. 488–507.

Makoviychuk, Viktor et al. (Aug. 2021). 'Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning'. In: *arXiv:2108.10470 [cs]*. URL: http://arxiv.org/abs/2108.10470 (visited on 01/03/2022).

Marsden, Jerrold E. and Ratiu, Tudor S. (2013). *Introduction to Mechanics and Symmetry: a Basic Exposition of Classical Mechanical Systems*. Springer Science & Business Media.

Maruskin, Jared (Aug. 2018). *Dynamical Systems and Geometric Mechanics: An Introduction*. De Gruyter. URL: https://www.degruyter.com/document/doi/10.1515/9783110597806/html (visited on 30/03/2022).

McGreavy, Christopher, Kai Yuan, Daniel Gordon, Kang Tan, Wouter J Wolfslag, Sethu Vijayakumar and Zhibin Li (May 2020). 'Unified Push Recovery Fundamentals: Inspiration from Human Study'. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10876–10882.

Metta, Giorgio et al. (2005). 'an open framework for research in embodied cognition'. In.

Miki, Takahiro, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun and Marco Hutter (Jan. 2022). 'Learning robust perceptive locomotion for quadrupedal robots in the wild'. In: *Science Robotics* 7.62, eabk2822. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abk2822 (visited on 21/05/2022).

Mnih, Volodymyr et al. (Feb. 2015). 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540, pp. 529–533. URL: http://www.nature.com/articles/nature14236 (visited on 23/06/2018).

Muratore, Fabio, Fabio Ramos, Greg Turk, Wenhao Yu, Michael Gienger and Jan Peters (Jan. 2022). 'Robot Learning from Randomized Simulations: A Review'. In: *arXiv:2111.00956 [cs]*. URL: http://arxiv.org/abs/2111.00956 (visited on 20/04/2022).

Nashner, Lewis M. and Gin McCollum (1985). 'The organization of human postural movements: a formal basis and experimental synthesis'. In: *Behavioral and brain sciences* 8.1, pp. 135–150.

Natale, Lorenzo, Chiara Bartolozzi, Daniele Pucci, Agnieszka Wykowska and Giorgio Metta (Dec. 2017). 'iCub: The not-yet-finished story of building a robot child'. In: *Science Robotics* 2.13. URL: https://www.science.org/doi/full/10.1126/scirobotics.aaq1026 (visited on 31/05/2022).

Nava, Gabriele, Francesco Romano, Francesco Nori and Daniele Pucci (Oct. 2016). 'Stability analysis and design of momentum-based controllers for humanoid robots'. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 680–687.

Nori, Francesco, Silvio Traversaro, Jorhabib Eljaik, Francesco Romano, Andrea Del Prete and Daniele Pucci (2015). 'iCub Whole-Body Control through Force Regulation on Rigid Non-Coplanar Contacts'. In: *Frontiers in Robotics and AI*. (Visited on 22/07/2020).

Pardo, Fabio, Arash Tavakoli, Vitaly Levdik and Petar Kormushev (Jan. 2022). 'Time Limits in Reinforcement Learning'. In: *arXiv:1712.00378 [cs]*. URL: http://arxiv.org/abs/1712.00378 (visited on 14/02/2022).

Peng, Xue Bin, Pieter Abbeel, Sergey Levine and Michiel van de Panne (July 2018a). 'DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills'. In: *ACM Transactions on Graphics*. (Visited on 24/09/2019).

Peng, Xue Bin, Marcin Andrychowicz, Wojciech Zaremba and Pieter Abbeel (May 2018b). 'Sim-to-Real Transfer of Robotic Control with Dynamics Randomization'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. (Visited on 22/05/2019).

Peng, Xue Bin, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan and Sergey Levine (Apr. 2020). 'Learning Agile Robotic Locomotion Skills by Imitating Animals'. In: *arXiv:2004.00784 [cs]*. URL: http://arxiv.org/abs/2004.00784 (visited on 07/05/2020).

Peng, Xue Bin, Yunrong Guo, Lina Halper, Sergey Levine and Sanja Fidler (July 2022). 'ASE: Large-Scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters'. In: *ACM Transactions on Graphics* 41.4, pp. 1–17. URL: http://arxiv.org/abs/2205.01906 (visited on 30/07/2022).

Peng, Xue Bin and Michiel van de Panne (2017). 'Learning locomotion skills using DeepRL: does the choice of action space matter?' In: *SIGGRAPH*.

Peters, Jan and Stefan Schaal (Oct. 2006). 'Policy Gradient Methods for Robotics'. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Beijing, China: IEEE, pp. 2219–2225. URL: http://ieeexplore.ieee.org/document/4058714/ (visited on 15/04/2022).

Peters, Jan, Sethu Vijayakumar and Stefan Schaal (2003). 'Reinforcement Learning for Humanoid Robotics'. In: *Proceedings of the third IEEE-RAS international conference on humanoid robots*, p. 20.

Pratt, Jerry, John Carff, Sergey Drakunov and Ambarish Goswami (Dec. 2006). 'Capture Point: A Step toward Humanoid Push Recovery'. In: *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pp. 200–207.

Pucci, Daniele, Francesco Romano, Silvio Traversaro and Francesco Nori (Nov. 2016). 'Highly dynamic balancing via force control'. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 141–141.

Puterman, Martin L. (2005). *Markov decision processes: discrete stochastic dynamic programming*. Wiley series in probability and statistics. Hoboken, NJ: Wiley-Interscience.

Qiao, Yi-Ling, Junbang Liang, Vladlen Koltun and Ming C. Lin (Sept. 2021). 'Efficient Differentiable Simulation of Articulated Bodies'. In: *arXiv:2109.07719 [cs]*. URL: http://arxiv.org/abs/2109.07719 (visited on 13/04/2022).

Rackauckas, Christopher, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan and Alan Edelman (Nov. 2021). 'Universal Differential Equations for Scientific Machine Learning'. In: *arXiv:2001.04385 [cs, math, q-bio, stat]*. URL: http://arxiv.org/abs/2001.04385 (visited on 20/04/2022).

Raffin, Antonin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus and Noah Dormann (2021). 'Stable-Baselines3: Reliable Reinforcement Learning Implementations'. In: *Journal of Machine Learning Research* 22.268,

pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html (visited on 26/05/2023).

Ramos, Fabio, Rafael Carvalhaes Possas and Dieter Fox (June 2019). 'BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators'. In: (visited on 09/08/2019).

Reda, Daniele, Tianxin Tao and Michiel van de Panne (Oct. 2020). 'Learning to Locomote: Understanding How Environment Design Matters for Deep Reinforcement Learning'. In: *Motion, Interaction and Games*. Virtual Event SC USA: ACM, pp. 1–10. URL: https://dl.acm.org/doi/10.1145/3424636.3426907 (visited on 30/07/2022).

Rodriguez, Diego and Sven Behnke (June 2021). 'DeepWalk: Omnidirectional Bipedal Gait by Deep Reinforcement Learning'. In: *arXiv:2106.00534 [cs]*. URL: http://arxiv.org/abs/2106.00534 (visited on 13/04/2022).

Rohmer, Eric, Surya P. N. Singh and Marc Freese (Nov. 2013). 'V-REP: A versatile and scalable robot simulation framework'. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1321–1326.

Romualdi, Giulio, Stefano Dafarra and Daniele Pucci (July 2021). 'Modeling of Visco-Elastic Environments for Humanoid Robot Motion Control'. In: *IEEE Robotics and Automation Letters* 6.3, pp. 4289–4296. URL: http://arxiv.org/abs/2105.14622 (visited on 26/07/2022).

Rudin, Nikita, David Hoeller, Philipp Reist and Marco Hutter (Oct. 2021). 'Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning'. In: *arXiv:2109.11978 [cs]*. URL: http://arxiv.org/abs/2109.11978 (visited on 13/04/2022).

Rummery, G. A. and M. Niranjan (1994). *On-Line Q-Learning Using Connectionist Systems*. Tech. rep.

Sabne, Amit (2020). *XLA: Compiling Machine Learning for Peak Performance*. Tech. rep.

Salvato, Erica, Gianfranco Fenu, Eric Medvet and Felice Andrea Pellegrino (2021). 'Crossing the Reality Gap: A Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning'. In: *IEEE Access* 9.

Schaal, Stefan (1996). 'Learning from Demonstration'. In: *Advances in Neural Information Processing Systems*. Vol. 9. MIT Press. URL: https://proceedings.

neurips.cc/paper/1996/hash/68d13cf26c4b4f4f932e3eff990093ba-Abstract.html (visited on 15/04/2022).

Schaal, Stefan (June 1999). 'Is imitation learning the route to humanoid robots?' In: *Trends in Cognitive Sciences* 3.6, pp. 233–242. URL: https://linkinghub.elsevier.com/retrieve/pii/S1364661399013273 (visited on 15/04/2022).

– (2006). 'Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics'. In: *Adaptive Motion of Animals and Machines*. Tokyo: Springer-Verlag, pp. 261–280. URL: http://link.springer.com/10.1007/4-431-31381-8_23 (visited on 15/04/2022).

Schulman, John, Sergey Levine, Philipp Moritz, Michael I. Jordan and Pieter Abbeel (Apr. 2017a). 'Trust Region Policy Optimization'. In: *arXiv:1502.05477 [cs]*. URL: http://arxiv.org/abs/1502.05477 (visited on 22/07/2020).

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan and Pieter Abbeel (Oct. 2018). 'High-Dimensional Continuous Control Using Generalized Advantage Estimation'. In: *arXiv:1506.02438 [cs]*. URL: http://arxiv.org/abs/1506.02438 (visited on 28/02/2022).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford and Oleg Klimov (2017b). 'Proximal Policy Optimization Algorithms'. In: *arXiv*. (Visited on 07/05/2020).

Selig, Jon M. (2005). *Geometric Fundamentals of Robotics*. Vol. 128. Springer. URL: https://link.springer.com/book/10.1007/b138859 (visited on 23/06/2022).

Shafiee, Milad, Giulio Romualdi, Stefano Dafarra, Francisco Javier Andrade Chavez and Daniele Pucci (Oct. 2019). 'Online DCM Trajectory Generation for Push Recovery of Torque-Controlled Humanoid Robots'. In: *arXiv:1909.10403 [cs]*. URL: http://arxiv.org/abs/1909.10403 (visited on 21/05/2022).

Silver, David, Satinder Singh, Doina Precup and Richard S. Sutton (Oct. 2021). 'Reward is enough'. In: *Artificial Intelligence* 299, p. 103535. URL: https://www.sciencedirect.com/science/article/pii/S0004370221000862 (visited on 09/02/2022).

Singh, Shubham, Ryan P. Russell and Patrick M. Wensing (Apr. 2022). 'Efficient Analytical Derivatives of Rigid-Body Dynamics using Spatial Vector Algebra'. In: *IEEE Robotics and Automation Letters* 7.2, pp. 1776–1783. URL: http://arxiv.org/abs/2105.05102 (visited on 20/04/2022).

Smith, Laura, J. Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan and Sergey Levine (Oct. 2021). 'Legged Robots that Keep on Learning: Fine-Tuning Locomotion Policies in the Real World'. In: *arXiv:2110.05457 [cs]*. URL: http://arxiv.org/abs/2110.05457 (visited on 18/10/2021).

Sola, Joan (2017). 'Quaternion kinematics for the error-state Kalman filter'. In.

Solà, Joan, Jeremie Deray and Dinesh Atchuthan (Nov. 2020). 'A micro Lie theory for state estimation in robotics'. In: *arXiv:1812.01537 [cs]*. URL: http://arxiv.org/abs/1812.01537 (visited on 26/11/2021).

Stephens, Benjamin (Nov. 2007). 'Humanoid push recovery'. In: *2007 7th IEEE-RAS International Conference on Humanoid Robots*. Pittsburgh, PA, USA: IEEE, pp. 589–595. URL: http://ieeexplore.ieee.org/document/4813931/ (visited on 21/05/2022).

Suh, H. J. Terry, Max Simchowitz, Kaiqing Zhang and Russ Tedrake (Feb. 2022). 'Do Differentiable Simulators Give Better Policy Gradients?' In: *arXiv:2202.00817 [cs]*. URL: http://arxiv.org/abs/2202.00817 (visited on 13/04/2022).

Sutton, Richard S. (Aug. 1988). 'Learning to predict by the methods of temporal differences'. In: *Machine Learning* 3.1, pp. 9–44. URL: http://link.springer.com/10.1007/BF00115009 (visited on 15/04/2022).

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, MA: The MIT Press.

Tan, Jie, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez and Vincent Vanhoucke (2018). 'Sim-to-Real: Learning Agile Locomotion For Quadruped Robots'. In: p. 10.

Tesauro, Gerald (Mar. 1994). 'TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play'. In: *Neural Computation* 6.2, pp. 215–219. URL: https://direct.mit.edu/neco/article/6/2/215-219/5771 (visited on 15/04/2022).

Theodorou, Evangelos, Jonas Buchli and Stefan Schaal (May 2010). 'Reinforcement learning of motor skills in high dimensions: A path integral approach'. In: IEEE, pp. 2397–2403. URL: http://ieeexplore.ieee.org/document/5509336/ (visited on 17/06/2018).

Theodorou, Evangelos A, Jonas Buchli, Stefan Schaal and Buchli Org (2010). 'A Generalized Path Integral Control Approach to Reinforcement Learning'. In: *The Journal of Machine Learning Research* 11, p. 45.

Todorov, E., T. Erez and Y. Tassa (Oct. 2012). 'MuJoCo: A physics engine for model-based control'. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033.

Todorov, Emanuel (May 2014). 'Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo'. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China: IEEE, pp. 6054–6061. URL: http://ieeexplore.ieee.org/document/6907751/ (visited on 25/05/2023).

Traversaro, Silvio (2017). 'Modelling, Estimation and Identification of Humanoid Robots Dynamics'. PhD thesis. URL: https://github.com/traversaro/traversaro-phd-thesis (visited on 05/01/2020).

Traversaro, Silvio, Daniele Pucci and Francesco Nori (2017). 'A Unified View of the Equations of Motion used for Control Design of Humanoid Robots'. In.

Traversaro, Silvio and Alessandro Saccon (2019). *Multibody dynamics notation*. Tech. rep., p. 24. URL: https://pure.tue.nl/ws/portalfiles/portal/139293126/A_Multibody_Dynamics_Notation_Revision_2_.pdf.

Tsounis, Vassilios, Mitja Alge, Joonho Lee, Farbod Farshidian and Marco Hutter (Jan. 2020). 'DeepGait: Planning and Control of Quadrupedal Gaits using Deep Reinforcement Learning'. In: *arXiv:1909.08399 [cs]*. URL: http://arxiv.org/abs/1909.08399 (visited on 28/07/2020).

Viceconte, Paolo Maria, Raffaello Camoriano, Giulio Romualdi, Diego Ferigo, Stefano Dafarra, Silvio Traversaro, Giuseppe Oriolo, Lorenzo Rosasco and Daniele Pucci (Apr. 2022). 'ADHERENT: Learning Human-like Trajectory Generators for Whole-body Control of Humanoid Robots'. In: *IEEE Robotics and Automation Letters* 7.2, pp. 2779–2786.

Vousten, Laurens C.J.M. (2022). 'Simulating Box Impact Dynamics in MuJoCo'. PhD thesis. Eindhoven University of Technology. URL: https://pure.tue.nl/ws/portalfiles/portal/212923254/1462253_Impacts_in_MuJoCo.pdf (visited on 25/05/2023).

Vukobratovic, Miomir and Davor Juricic (Jan. 1969). 'Contribution to the Synthesis of Biped Gait'. In: *IEEE Transactions on Biomedical Engineering* BME-16.1, pp. 1–6.

Vukobratović, Miomir and Branislav Borovac (Mar. 2004). 'Zero-moment point — thirty five years of its life'. In: *International Journal of Humanoid Robotics* 01.01, pp. 157–173. URL: https://www.worldscientific.com/doi/abs/10.1142/S0219843604000083 (visited on 21/05/2022).

Warner, Frank W. (1983). *Foundations of Differentiable Manifolds and Lie Groups*. Vol. 94. Springer Science & Business Media. URL: https://link.springer.com/book/10.1007/978-1-4757-1799-0 (visited on 23/06/2022).

Watkins, Christopher (1989). 'Learning from Delayed Rewards'. PhD thesis. King's College.

Werling, Keenon, Dalton Omens, Jeongseok Lee, Ioannis Exarchos and C. Karen Liu (June 2021). 'Fast and Feature-Complete Differentiable Physics for Articulated Rigid Bodies with Contact'. In: *arXiv:2103.16021 [cs, eess]*. URL: http://arxiv.org/abs/2103.16021 (visited on 13/04/2022).

Wieber, Pierre-brice (Dec. 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*, pp. 137–142.

Williams, Ronald J. (May 1992). 'Simple statistical gradient-following algorithms for connectionist reinforcement learning'. In: *Machine Learning* 8.3, pp. 229–256. URL: https://doi.org/10.1007/BF00992696 (visited on 15/04/2022).

Xie, Zhaoming, Patrick Clary, Jeremy Dao, Pedro Morais, Jonathan Hurst and Michiel van de Panne (Mar. 2019). 'Iterative Reinforcement Learning Based Design of Dynamic Locomotion Skills for Cassie'. In: (visited on 05/04/2019).

Xu, Jie, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg and Miles Macklin (Apr. 2022). *Accelerated Policy Learning with Parallel Differentiable Simulation*. URL: http://arxiv.org/abs/2204.07137 (visited on 28/07/2022).

Yang, C., K. Yuan, W. Merkt, T. Komura, S. Vijayakumar and Z. Li (Nov. 2018). 'Learning Whole-Body Motor Skills for Humanoids'. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 270–276.

Yoon, Jaemin, Bukun Son and Dongjun Lee (Jan. 2023). 'Comparative Study of Physics Engines for Robot Simulation with Mechanical Interaction'. In: *Applied Sciences* 13.2, p. 680. URL: https://www.mdpi.com/2076-3417/13/2/680 (visited on 25/05/2023).

Zamora, Iker, Nestor Gonzalez Lopez, Victor Mayoral Vilches and Alejandro Hernandez Cordero (Feb. 2017). *Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo*. Tech. rep. arXiv:1608.05742. arXiv. URL: http://arxiv.org/abs/1608.05742 (visited on 31/05/2022).

Zhao, Wenshuai, Jorge Peña Queralta and Tomi Westerlund (Sept. 2020). 'Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey'. In: *arXiv:2009.13303 [cs]*. URL: http://arxiv.org/abs/2009.13303 (visited on 02/10/2020).

Part III

APPENDIX

A CENTER OF PRESSURE

In this section, we provide the intuition and equations for computing the CoP of a rigid body. A practical example is the computation of the CoP of a robot foot modelled as a body having the shape of a box, as illustrated in Figure A.1. Let us assume the following properties:

- We assume a flat terrain with a normal $\hat{n} = (0, 0, 1) \in \mathbb{R}^3$. Therefore, the normal remains constant over the entire surface supporting the body.
- We consider the existence of k external 6D forces $\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_k \in \mathbb{R}^6$ applied to different points of the body.
- We introduce an unknown frame $C = ({}^W\mathbf{p}_{CoP}, [W])$ corresponding to the CoP with ${}^W\mathbf{p}_{CoP}$ belonging to the supporting surface. The flat terrain assumption implies that the z axis of the C frame is normal to the terrain.

Definition A.0.1 (Center of Pressure)**.** The CoP is the point ${}^W\mathbf{p}_{CoP} \in \mathbb{R}^3$ belonging to the rigid body's supporting surface to which a pure linear force ${}_W\boldsymbol{f}_{CoP} \in \mathbb{R}^3$ can be applied to produce the equivalent effects along the normal direction of all the external 6D forces. □

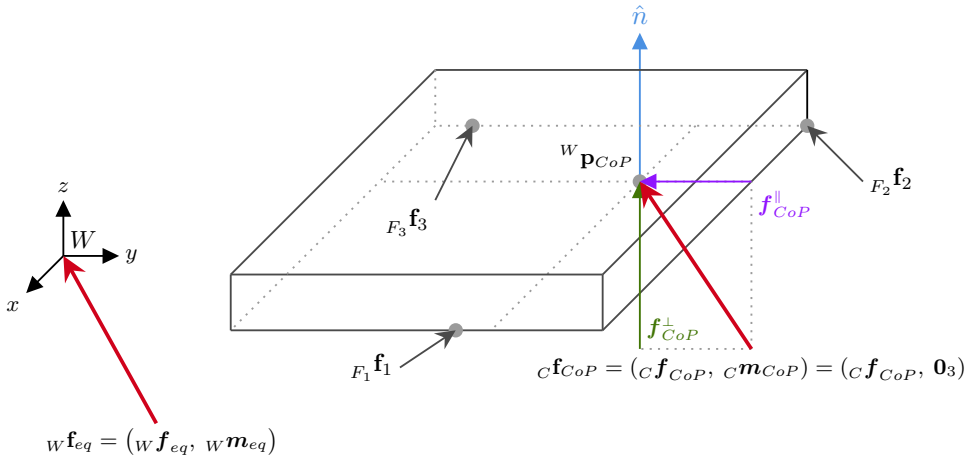


Figure A.1: Illustration of the setting in which the CoP is calculated.

As first step, we need to combine all the external forces, computing a single *equivalent* 6D force ${}_W\mathbf{f}_{eq} \in \mathbb{R}^6$. If the external forces are expressed in W, the equivalent force is just the sum of external forces ${}_W\mathbf{f}_{eq} = \sum_i {}_W\mathbf{f}_i$. Instead, if they are are expressed in a local frame $F_i = ({}^W\mathbf{p}_i, [W])$, assuming the knowledge of the application point, we can use the transform introduced in Equation (2.11):

$${}_W\mathbf{f}_{eq} = \begin{bmatrix} {}_W\boldsymbol{f}_{eq} \\ {}_W\boldsymbol{m}_{eq} \end{bmatrix} = s \sum_i {}_W\mathbf{X}^{F_i} \, {}_{F_i}\mathbf{f}_i.$$

We can project the equivalent 6D force into the normal and tangential components with respect ground:

$${}_W\mathbf{f}_{eq} = \begin{bmatrix} {}_W\boldsymbol{f}^{\perp}_{eq} + {}_W\boldsymbol{f}^{\parallel}_{eq} \\ {}_W\boldsymbol{m}_{eq} \end{bmatrix} = \begin{bmatrix} {}_W\boldsymbol{f}^{\perp}_{eq} \\ {}_W\boldsymbol{m}_{eq} \end{bmatrix} + \begin{bmatrix} {}_W\boldsymbol{f}^{\parallel}_{eq} \\ \mathbf{0}_3 \end{bmatrix} = {}_W\mathbf{f}^{\perp}_{eq} + {}_W\mathbf{f}^{\parallel}_{eq},$$

where we grouped the equivalent angular component ${}_W\boldsymbol{m}_{eq}$ with the perpendicular term. The linear component of the equivalent perpendicular force can be computed as:

$${}_W\boldsymbol{f}^{\perp}_{eq} = ({}_W\boldsymbol{f}_{eq} \cdot \hat{n})\, \hat{n} = \begin{bmatrix} 0 \\ 0 \\ f^{\perp z}_{eq} \end{bmatrix}.$$

The point associated to the CoP can now be obtained by solving for ${}^W\mathbf{p}_{CoP}$ the following equation that expresses the equivalent 6D force into the CoP and enforces it to be a pure linear force by setting the resulting angular component to zero:

$$\begin{aligned} {}_C\mathbf{f}_{CoP} = \begin{bmatrix} {}_C\boldsymbol{f}_{CoP} \\ \mathbf{0}_3 \end{bmatrix} &= {}_C\mathbf{X}^W \, {}_W\mathbf{f}^{\perp}_{eq} = \begin{pmatrix} \mathbf{I}_3 & \mathbf{0}_3 \\ -{}^W\mathbf{p}^{\wedge}_{CoP} & \mathbf{I}_3 \end{pmatrix} \begin{bmatrix} {}_W\mathbf{f}^{\perp}_{eq} \\ {}_W\boldsymbol{m}_{eq} \end{bmatrix} \\ &= \begin{bmatrix} {}_W\mathbf{f}^{\perp}_{eq} \\ {}_W\mathbf{f}^{\perp}_{eq} \times {}^W\mathbf{p}_{CoP} + {}_W\boldsymbol{m}_{eq} \end{bmatrix}. \end{aligned}$$

where we used the relation ${}^{C}\mathbf{o}_{W} = -{}^{W}\mathbf{p}_{CoP}$. We can rearrange the second equation as follows:

$$ {}_{W}\boldsymbol{m}_{eq} = \begin{bmatrix} m^{x}_{eq} \\ m^{y}_{eq} \\ m^{z}_{eq} \end{bmatrix} = {}_{W}\boldsymbol{f}^{\perp}_{eq} \times {}^{W}\mathbf{p}_{CoP} = \begin{pmatrix} 0 & -f^{\perp z}_{eq} & 0 \\ f^{\perp z}_{eq} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{bmatrix} p^{x}_{CoP} \\ p^{y}_{CoP} \\ p^{z}_{CoP} \end{bmatrix} $$

and finally resolve for ${}^{W}\mathbf{p}_{CoP}$:

$$ {}^{W}\mathbf{p}_{CoP} = \begin{bmatrix} m^{y}_{eq}/f^{\perp z}_{eq} \\ -m^{x}_{eq}/f^{\perp z}_{eq} \\ 0 \end{bmatrix}. $$

This result can be extended to different terrain normals through geometrical transformations. The most impacting difference would regard the frame corresponding to the CoP, defined as $C = ({}^{W}\mathbf{p}_{CoM}, [W])$. If the terrain has a different normal, the orientation of the frame should still be considered a known value but, this time, obtained by the properties of the surface.