Learning and Optimization of the Locomotion with an Exoskeleton for Paraplegic People

Alexis Duburcq

▶ **To cite this version:**

Alexis Duburcq. Learning and Optimization of the Locomotion with an Exoskeleton for Paraplegic People. Artificial Intelligence [cs.AI]. Université Paris sciences et lettres, 2022. English. ⟨NNT : 2022UPSLD061⟩. ⟨tel-04166955⟩

HAL Id: tel-04166955

https://theses.hal.science/tel-04166955v1

Submitted on 20 Jul 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université Paris-Dauphine

Apprentissage et Optimisation de la Locomotion pour un Exosquelette à Destination des Patients Paraplégiques

Learning and Optimization of the Locomotion with an Exoskeleton for Paraplegic People

Soutenue par

Alexis Duburcq

Le 09 Decembre 2022

École doctorale n°543

Sciences de la Décision, des Organisations, de la Société et de l'Échange

Spécialité

Sciences et technologies de l'information et de la communication

Composition du jury :

Sylvain Chevallier Pr., Université de Versailles	*Président du Jury* *Examinateur*
Christine Chevallereau Dir. de recherche, Centrale Nantes	*Rapporteur*
Jean-Baptiste Mouret Dir. de recherche, INRIA	*Rapporteur*
Sylvain Finet Dr., —	*Examinateur*
Guilhem Boéris Ingénieur, Wandercraft	*Co-encadrant*
Nicolas Bredeche Pr., Sorbonne Université	*Co-encadrant*
Yann Chevaleyre Pr., Université Paris-Dauphine	*Directeur de thèse* *Encadrant*

Dauphine
UNIVERSITÉ PARIS

Abstract

This thesis contributes to improving the motion planning and control of bipedal robots. Our concrete goal is restoring natural locomotion for paraplegic people in their daily lives using the medical lower-limb exoskeleton Atalante, notably walking safely and autonomously without crutches. Consequently, the relevance of our methods is determined by their ability to fulfill this goal. In this context, pragmatism through constant confrontation with reality has been the cornerstone of this work. The core idea is to combine traditional robotics and state-of-the-art machine learning to benefit from their respective advantages.

In the first part, we put aside closed-loop control to focus on planning. The objective is to enable online trajectory planning while ensuring safe operation. This is a milestone toward realizing versatile navigation in an unstructured environment and accommodating the preferences of the user. We achieve this by training offline a function approximation of the solution to any given trajectory optimization problem over a continuous task space. We ensure that all the optimal trajectories can be perfectly reproduced by the function approximation regardless of its expressiveness. The computation cost is comparable to generating a finite database of trajectories and scales well to high-dimensional task spaces. Our algorithm is compatible with any motion planning framework and can be used for solving any multi-parametric optimization problem beyond the robotic field. In practice, the function approximation is a neutral network specifically tailored for predicting continuous time series and serves as a 'Memory of motion' that is queried online in no time for the task at hand.

In the second part, we train a policy using reinforcement learning to generalize a predefined set of primitive motions that have been generated and clinically validated for an average user moving around on flat ground. The challenge is not to achieve the best possible performance but rather to ensure transferability and safety. We propose an original formulation closely related to imitation learning, in the sense that trajectories are used to guide and constrain the policy optimization in the same way as expert demonstrations, while giving enough freedom to deal with large external disturbances or modelling discrepancies. Two very different training scenarios have been studied: smoothly transitioning between nominal motions, and reactive stepping for emergency push recovery while standing. Only the latter has been evaluated on Atalante for lack of time. It transfers to reality and attains satisfactory performance without any kind of adaptation, which is very promising.

To support this work, an open-source simulator of poly-articulated robots called Jiminy has been developed. It is heavily optimized for reinforcement learning applications. In particular, several parameters are available to trade-off between realism and regularity of the physics to ease or speed up learning. Internally, it relies on a novel analytical contact formulation that does not involve computing impulse forces. Besides, it takes into account many of the hardware limitations and side effects, among them backlashes, stochastic communication delays, the inertia of the rotors, and the mechanical deformation of the structure.

Résumé

L'objectif de cette thèse est d'exploiter les methodes existantes issu du domaine machine learning afin d'améliorer le planning et control des robots bipèdes. Dès le départ, nous nous sommes fixé comme objectif concret d'aider les paraplégiques à remarcher de façon autonome à l'aide de l'exosquelette de membres inférieurs Atalante. Afin de ne pas perdre de vue cette objective, le pragmatisme et la perpetuelle confrontation à la réalité ont été les pierres angulaires de ce travail. Ce paradigme a eu une importance capitale dans le design des methodes qui ont été proposé dans ce travail, tout en enforçant malgré tout à étendre leur portée au maximum. L'idée centrale est de combiner les methodes issues des domaines du machine learning et de la robotique traditionnelle afin de mutualiser leurs avantages respectifs, plutôt que de substituer l'un à l'autre.

Dans la première partie, nous laissons de côté le contrôle en boucle fermée. L'objectif est de permettre la planification de trajectoires en ligne tout en garantissanun fonctionnement sûr. Il s'agit d'une étape importante vers la navigation en environnement non structuré et la prise en compte des préférences utilisateur. Nous y parvenons en entraînant hors ligne une fonction d'approximation des solutions à un problème d'optimisation de trajectoire quelconque pour un espace de tâche continu. Nous nous assurons que les trajectoires ainsi générées puissent être parfaitement reproduites par la function d'approximation, quelle que soit son expressivité. Le coût de calcul est comparable à la génération d'une base de données de trajectoires et s'adapte bien à un espace de tâches de grande dimension. Notre algorithme est compatible avec n'importe quel outil de planification de mouvement et peut également être utilisé pour résoudre n'importe quel problème d'optimisation multiparamétrique au-delà du domaine de la robotique. En pratique, la fonction d'approximation est un réseau de neurones spécialement conçu pour prédire des séries temporelles continues et sert de "mémoire du mouvement" pouvant être évaluée en ligne presque instantanément.

Dans la deuxième partie, nous entraînons un contrôleur par apprentissage par renforcement afin de généraliser un ensemble prédéfini de mouvements élémentaires qui ont été générés et validés cliniquement avec un utilisateur moyen se déplaçant sur un terrain plat. L'objectif n'est pas d'atteindre la meilleure performance possible, mais plutôt d'assurer la transférabilité et la sécurité. Nous proposons une nouvelle formulation étroitement liée à l'apprentissage par imitation, dans le sens où les trajectoires sont utilisées pour guider et contraindre l'optimisation du contrôleur de la même manière que des démonstrations d'experts, tout en donnant suffisamment de liberté pour compenser de grandes perturbations extérieures ainsi que les erreurs de modélisation. Deux scénarios très différents ont été étudiés : reproduire l'ensemble des mouvements nominaux, et se rattraper lors d'un violent impact dans une posture statique de repos. Seul ce dernier cas de figure a été évalué sur Atalante par manque de temps. La performance du contrôleur sont satisfaisantes sans aucun type d'adaptation en dépit du transfert de la simulation à la réalité, ce qui est prometteur.

Un simulateur open-source de robots poly-articulés appelé Jiminy a été développé afin de rendre ce travail possible. Il est spécifiquement adapté à l'apprentissage par renforcement. En particulier, plusieurs paramètres sont disponibles pour arbitrer entre réalisme et régularité de la physique afin de faciliter ou d'accélérer l'apprentissage. En interne, il s'appuie sur une nouvelle formulation analytique du contact qui ne nécessite pas le calcul de forces impulsionnelles. En outre, il tient compte de nombreuses limitations matérielles et effets secondaires, notamment le jeu articulaire, le délai de communication variable, l'inertie des rotors et la déformation mécanique de la structure.

Table of Contents

Abstract i

Résumé iii

Table of Contents v

List of Figures ix

Acronymes xiii

1 Introduction 1
- 1.1 Context and Motivation . . . 1
  - 1.1.1 Enhancing Motor Recovery and Rehabilitation . . . 1
  - 1.1.2 Restoring Locomotion for Disabled People . . . 2
  - 1.1.3 Atalante: Self-Balanced Medical Lower-Limb Exoskeleton . . . 3
  - 1.1.4 Technical Constraints and Challenges . . . 5
- 1.2 Problem Statement and Contributions . . . 8
  - 1.2.1 Online Trajectory Planning . . . 8
  - 1.2.2 Robust and Safe Control Policy Optimization . . . 10
  - 1.2.3 Realistic Open-Source Simulator for Reinforcement Learning . . . 11

2 Background in Robotics 15
- 2.1 Preliminaries on Rigid Body Dynamics . . . 15
  - 2.1.1 Spatial Vector Algebra . . . 18
  - 2.1.2 Whole-Body Dynamics . . . 26
- 2.2 Planning and Control in Bipedal Robotics . . . 28
  - 2.2.1 Notion of Bipedal Locomotion and Terminology . . . 28
  - 2.2.2 Stability Assessment . . . 33
  - 2.2.3 Classical Control Methods . . . 42

3 Background in Machine Learning 49
- 3.1 Supervised Learning and Neural Networks . . . 49
  - 3.1.1 Introduction to Supervised Learning . . . 49
  - 3.1.2 Fundamentals of Artificial Neural Networks . . . 54
- 3.2 The Reinforcement Learning Problem . . . 65
  - 3.2.1 The Agent-Environment Interaction Loop . . . 66
  - 3.2.2 Key Concepts and Terminology . . . 73
  - 3.2.3 Taxonomy of Algorithms for Continuous Control . . . 86

4 Related work 99

4.1 Related Work on Online Trajectory Planning . . . 99
4.1.1 Model Prediction Control . . . 99
4.1.2 Imitation Learning . . . 102
4.1.3 Trajectory Planning and Function Approximation . . . 106
4.2 Related Work on Robust and Safe Control Policy . . . 110
4.2.1 Classical Control Methods . . . 110
4.2.2 Policy Learning Methods . . . 113
4.2.3 Predictability and Safety in Reinforcement Learning . . . 117

5 Online Trajectory Planning **125**
5.1 Trajectory Planning Problem . . . 126
5.1.1 Optimal Control Formulation . . . 126
5.1.2 Vector Representation of the Solutions . . . 127
5.2 Trajectory Learning . . . 128
5.2.1 Naive Formulation as a Standard Regression . . . 128
5.2.2 Efficient Task Sampling and Certifiability . . . 129
5.2.3 Limitations . . . 132
5.3 Guided Trajectory Learning . . . 133
5.3.1 Unifying Trajectory Planning and Learning . . . 133
5.3.2 Trade-off Between Generalization Ability and Optimality . . . 135
5.4 Iterative Solving . . . 136
5.4.1 Alternating Direction Method of Multipliers . . . 136
5.4.2 Proposed Consensus Optimization Algorithm . . . 138
5.4.3 Theoretical Analysis . . . 138
5.5 Structured Prediction with Neural Network . . . 143
5.5.1 Autoregressive Model . . . 143
5.5.2 Generative Single Forward Pass Model . . . 145
5.6 Experimental Validation . . . 148
5.6.1 Toy Problem: Van der Pol Oscillator . . . 148
5.6.2 Application to Atalante: Flat Foot Walking . . . 151
5.7 Concluding Remarks . . . 158

6 Learning Robust and Safe Policy **159**
6.1 Problem Setup . . . 161
6.1.1 Learning Environment . . . 161
6.1.2 Training Scenarios . . . 165
6.2 Constrained Policy Optimization . . . 166
6.2.1 Implicit Constraints through Early Termination . . . 167
6.2.2 Explicit Constraints through Barrier Functions . . . 170
6.3 Trajectory-Based Imitation using Reinforcement Learning . . . 172
6.3.1 Generalized Space-Time Bounds . . . 173
6.3.2 Scalable Multi-Tasking through Lower-Bound Maximization . . . 182
6.3.3 Task Transitioning via Cross-Initialization . . . 186
6.4 Improving Convergence, Predictability and Safety . . . 187

6.4.1 Reward Engineering . . . 187
6.4.2 Termination Conditions . . . 191
6.4.3 Explicit constraints . . . 192
6.4.4 Smoothness Conditioning . . . 193
6.5 Ensuring Robustness for Bridging the Simulation-Reality Gap . . . 194
6.5.1 Plausible External Disturbances . . . 194
6.5.2 Feasible Initial State Generation . . . 195
6.5.3 Domain Randomization . . . 196
6.6 Results . . . 198
6.6.1 Policy Network Architecture . . . 198
6.6.2 Training Performance . . . 198
6.6.3 Validation in Simulation . . . 198
6.6.4 Standing Push Recovery on Atalante . . . 201
6.7 Concluding Remarks . . . 201

7 Conclusion **203**
7.1 Summary of the Contributions . . . 203
7.2 Discussion . . . 205
7.2.1 Online Trajectory Planning . . . 205
7.2.2 Robust and Safe Control Policy . . . 205
7.3 Perspectives and Future Works . . . 205
7.3.1 Human-Like Locomotion . . . 205
7.3.2 Data Efficiency . . . 206
7.3.3 Global Optimally . . . 206
7.3.4 Transfer to reality . . . 206

Appendices **209**

A Trajectory Planning Using Whole-Body Optimization **211**

B Proximal Splitting Method **227**

C Jiminy – Open-source Simulator for Legged Robots **235**

D Classical Neural Networks Architectures **259**

E Classical Approaches in Reinforcement Learning **269**

Alphabetical Index **297**

Bibliography **299**

List of Figures

1.1 Atalante Product by Wandercraft . . . 3
1.2 Overview of the Hardware of Atalante . . . 5
1.3 Modelling of the Flexibility of the Mechanical Structure of Atalante . . . 6

2.1 Examples of Kinematic Graphs . . . 16
2.2 Canonical Generalized Coordinate for the Cartpole . . . 18
2.3 Description of a Kinematic Chain . . . 25
2.4 Examples of Bipedal Robots . . . 29
2.5 Human Planes of Section . . . 31
2.6 Characterization of Bipedal Locomotion . . . 32
2.7 Motion Primitives for Versatile Locomotion on Flat Ground . . . 32
2.8 Contact Force Measurement . . . 35
2.9 Dynamic Stability Assessment on a Flat Ground . . . 37
2.10 Virtual Support Area . . . 39
2.11 Centroidal Dynamics and Inverted Pendulum Model . . . 40
2.12 Comparison Between True and Projected Support Polygon . . . 41
2.13 Illustration of Early Switch Replanning . . . 44
2.14 Model Predictive Control over a Finite Horizon. . . . 47

3.1 Effect of Function Space Complexity and Number of Samples . . . 53
3.2 Illustration of an Artificial Neuron . . . 55
3.3 Simple Binary Classifier . . . 55
3.4 Illustration of Backpropagation . . . 56
3.5 Graphical Representation of Three Common Activation Functions . . . 60
3.6 Discrete Convolution Acting As a Filter . . . 63
3.7 Agent-Environment Interaction Loop in Reinforcement Learning . . . 67
3.8 Classic Policy and State-Value Network Architecture . . . 78
3.9 Taxonomy of Policy Learning Algorithms . . . 90
3.10 Sample Efficiency of Policy Learning Algorithms . . . 91
3.11 PPO-Clipped Objective for a Single Transition Step . . . 96

5.1 Monte-Carlo Study of the Maximum Inter-Point Distance . . . 131
5.2 Comparison Between Standard Regression and Guided Trajectory Learning 141
5.3 Comparison Between Behavior Cloning and Trajectory Unfolding . . . 143
5.4 Denoising Autoencoder . . . 144
5.5 Trajectory Generation with an Open-Loop Policy . . . 145
5.6 Specialized Network Architecture for Continuous-Time Markov Process . . . 147
5.7 Overview of the Original Optimal Trajectories . . . 148
5.8 Evolution of the Total Prediction Error over GTL Iterations . . . 149
5.9 Prediction Error Distribution for Initial and Final GTL Iterations . . . 149

5.10 Pairwise Distance Between Training Samples . . . 150
5.11 Flat-Foot Walking Domains . . . 151
5.12 Original Optimal Trajectories of the Ankle Joints . . . 153
5.13 Pairwise Distance Between Training Samples . . . 154
5.14 Effect of the Number of Parameters on the Training Error . . . 155
5.15 Effect of Number of Samples on the Testing Error . . . 155
5.16 Norm-Inf Test Error Distribution over Iterations of GTL-0 . . . 156
5.17 Continuity of the Trajectories with Respect to the Task . . . 157
5.18 From Simulation to Reality . . . 158

6.1 Overview of Proposed Control Flow. . . . 162
6.2 Probabilistic Constraint on Motor Position Tracking Error . . . 177
6.3 Probabilistic Constraint on Motor Tracking Error . . . 178
6.4 RBF Kernel for Scaling Reward Components. . . . 188
6.5 Ablation Study . . . 199
6.6 Comparison Between the Predicted Target Velocity, Measured Position, and Command Torque . . . 200
6.7 Strong Impact Kick . . . 200
6.8 Maximum Recoverable Force Magnitude . . . 201
6.9 Scenarios Showing the Robustness of the Push Recovery Policy . . . 202

A.1 Directed Graph of Foot Rolling Walking Pattern . . . 212
A.2 Illustration of the PHZD Periodic Orbit . . . 220
A.3 Illustration of Feedback Linearization . . . 221
A.4 Illustration of the Direct Collocation . . . 224

B.1 Geometric Interpretation of the First-Order Multiplier Iteration . . . 231

C.1 Modelling of the Mechanical Deformation in Simulation . . . 236
C.2 Non-Uniqueness of the Solution in Case of Constraint Redundancy . . . 240
C.3 Ground Reaction Force at Contact Point . . . 241
C.4 Theoretical Evolution of Tangential Velocity of a Contact Point . . . 244
C.5 Theoretical Evolution of Contact Depth over Time . . . 245
C.6 Example of Polyhedral Linear Approximation of Friction Cone. . . . 247
C.7 Pathological Issue with Collision Detection . . . 249
C.8 Illustration of Convex Hull for a Chair . . . 251
C.9 Closest Point to Heightmap Ground Profile . . . 252
C.10 Evaluation of Acceleration-Level Constrained Dynamics Formulation . . . 253
C.11 Pantograph Example of Closed Kinematic Chain Transmission . . . 256
C.12 Transmission of the Ankle of Atalante Exoskeleton . . . 257

D.1 Residual Learning Building Block . . . 261
D.2 Illustration of Encoder-Decoder Architecture for Image Segmentation . . . 263
D.3 Description of Long Short-Term Memory Network . . . 265

E.1 Generalized Policy Iteration 270
E.2 Comparison of the Backup Strategies in Reinforcement Learning 273
E.3 Comparison Between LR and RP Gradients 293

Acronymes

A2C	Advantage Actor-Critic
A3C	Asynchronous A2C
ABA	Articulated Body Algorithm
ADAM	ADAptive Moment estimation
ADMM	Alternating Direction Method of Multipliers
ANN	Artificial Neural Network
BC	Behavior Cloning
BFGS	Broyden-Fletcher-Goldfarb-Shannon
CEM	Cross-Entropy Method
CG	Conjugate Gradient
CGL	Chebyshev-Gauss-Lobatto
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CNN	Convolutional Neural Network
CoM	Center of Mass
CoP	Center of Pressure
CPI	Conservative Policy Iteration
CPO	Constrained Policy Optimization
CRBA	Composite Rigid Body Algorithm
CROP	Certifying RObust Policies
DAGGER	Data AGGERation
DCM	Divergent Component of Motion
DDP	Differential Dynamic Programming
DDPG	Deep DPG
DNN	Deconvolutional Neural Network
DoF	Degree of Freedom
DP	Dynamic Programming
DPG	Deterministic Policy Gradient
DQN	Deep Q Network
DTL	Distributed Trajectory Learning
EA	Evolutionary Algorithms
EGLP	Expected Grad-Log-Prob
EKF	Extended Kalman Filter

ERL	Evolutionary Reinforcement Learning
ES	Evolutionary Strategies
FD	Forward Dynamics
FEM	Finite Element Method
FGSM	Fast Gradient Sign Method
FIM	Fisher Information Matrix
FIP	Floating-base Inverted Pendulum
FK	Forward Kinematics
FNN	Feedforward Neural Network
GAE	Generalized Advantage Estimator
GAN	Generative Adversarial Nets
GEV	Generalized Extreme Value
GJK	Gilbert-Johnson-Keerthi
GMM	Gaussian Mixture Model
GPI	Generalized Policy Iteration
GPS	Guided Policy Search
GRU	Gated Recurrent Unit
GTL	Guided Trajectory Learning
HER	Hindsight Experience Replay
HLC	High-Level Controller
HZD	Hybrid Zero Dynamics
ID	Inverse Dynamics
IK	Inverse Kinematics
IL	Imitation Learning
IMU	Inertial Measurement Unit
IPO	Interior Point Optimization
IS	Importance Sampling
K-FAC	Kronecker-factored Approximate Curvature
KKT	Karush-Kuhn-Tucker
KL	Kullback-Leibler
KNN	K-Nearest Neighbors
LIPM	Linear Inverted Pendulum Model
LLC	Low-Level Controller
LQR	Linear Quadratic Regulator
LR	likelihood-ratio gradient
LSTM	Long Short-Term Memory

LTI	Linear Time-Invariant
MC	Monte-Carlo
MCPO	Masked CPO
MDP	Markov Decision Process
MFCQ	Mangasarian Fromovitz Constraint Qualification
MLCP	Mixed Linear Complementary Problem
MLP	Multi-Layer Perceptron
MPC	Model-Based Predictive Control
MSE	Mean Squared Error
NAS	Neural Architecture Search
NLCP	Non-Linear Complementarity Problem
NLP	Non-Linear Program
OCP	Optimal Control Problem
OffPAC	Off-Policy Actor-Critic
OSIM	Operational Space Inertia Matrix
PCA	Principal Component Analysis
PGD	Projected Gradient Descent
PGS	Projected Gauss-Seidel
PHZD	Partial HZD
PID	Proportional-Integral-Derivative controller
PILCO	Probabilistic Inference for Learning COntrol
PINN	Physics-Informed Neural Network
PLATO	Policy Learning using Adaptive Trajectory Optimization
PPO	Proximal Policy Optimization
QP	Quadratic Program
RBF	Radial Basis Function
RL	Reinforcement Learning
RMSProp	Root Mean Square Propagation
RND	Random Network Distillation
RNEA	Recursive Newton-Euler Algorithm
RNN	Recurrent Neural Network
RP	ReParametrization gradient
SAC	Soft Actor Critic
SARSA	State-Action-Reward-State-Action
SEA	Serial Elastic Actuator
SGD	Stochastic Gradient-Descent

SGLD	Stochastic Gradient Langevin Dynamics
SMILe	Stochastic Mixing Iterative Learning
SOCP	Second-Order Cone Program
SSOR	Successive Over Relaxation
SVIGP	Stochastic Variational Inference for Gaussian Process
SVM	Support-Vector Machines
SVR	Support Vector Regression
TBDP	Trajectory-Based Dynamic Programming
TD	Temporal-Difference
TD3	Twin-Delayed DDPG
TRPO	Trust Region Policy Optimization
TV	Total Variation
VFF	Variational Fourier Features for Gaussian Process
VHIP	Variable-Height Inverted Pendulum
ZMP	Zero-tilting Moment Point
ZO	Zeroth-Order

Chapter 1

Introduction

Contents

1.1 Context and Motivation 1
1.1.1 Enhancing Motor Recovery and Rehabilitation 1
1.1.2 Restoring Locomotion for Disabled People 2
1.1.3 Atalante: Self-Balanced Medical Lower-Limb Exoskeleton 3
1.1.4 Technical Constraints and Challenges 5
1.2 Problem Statement and Contributions 8
1.2.1 Online Trajectory Planning 8
1.2.2 Robust and Safe Control Policy Optimization 10
1.2.3 Realistic Open-Source Simulator for Reinforcement Learning 11

1.1 Context and Motivation

1.1.1 Enhancing Motor Recovery and Rehabilitation

Stroke is a leading cause of motor disabilities. One in 6 people will have a stroke in their lifetime, and 17 million people have had one in 2010 of which 31% aged less than 65 years (Feigin et al., 2014). Of these, 65% are left with severe disability, affecting their capacity to independently carry out activities of daily living according to the Stroke Foundation (2020). In China, which has the highest stroke rate in the world, there are nearly 15 million disabled people with lower-limb motor dysfunctions (Shi et al., 2019). More people are in need of stroke rehabilitation every year, but there is already a lack of therapists. Therefore, lower-limb rehabilitation robots are of great significance as they can reduce the burden on therapists.

It is known that therapy should begin as soon as possible and provide intensive sessions that incorporate multiple sensory mechanisms in a structured way to be effective (Calabrò et al., 2016). In recent years, rehabilitation robots have gained interest for their ability to automate them. Some medical studies came to the conclusion that powered exoskeletons can supplement a therapist in some cases such as stroke patients for which the mobility of the upper body is affected (Conesa et al., 2012; Masiero et al., 2007), but they do not win unanimous support (Kwakkel et al., 2008; Postol et al., 2021; Veerbeek et al., 2017). Similarly, preliminary studies targeting

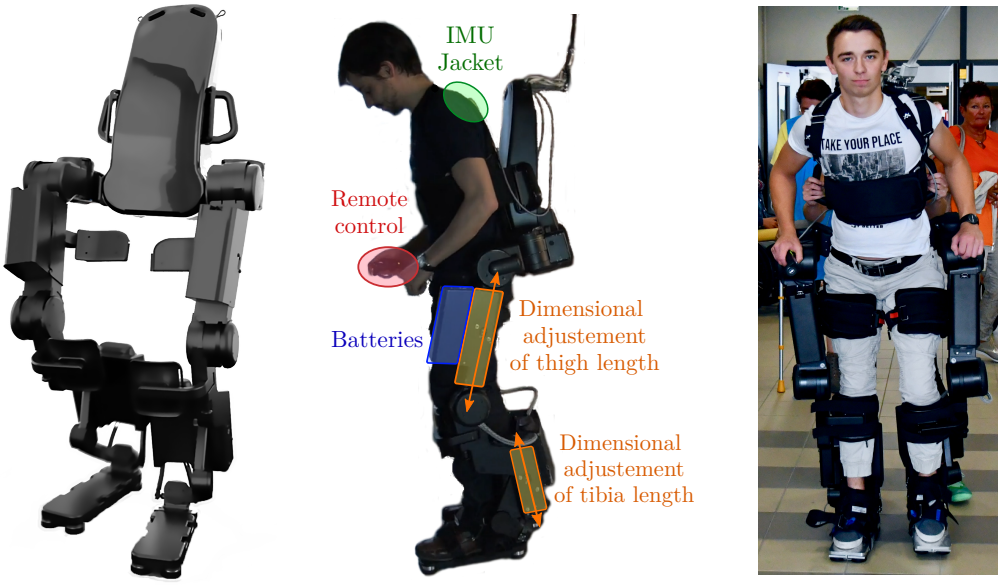
loss of mobility of the lower body for stroke patients show no significant improvement in motor function recovery using lower-limb exoskeletons compared with conventional therapy alone (Hobbs & Artemiadis, 2020; Postol et al., 2021; Shi et al., 2019).

It is primarily the devices used in these studies that are to blame for these mixed results, not the very idea of using powered exoskeletons to supplement conventional therapy. These technologies are rapidly-evolving and were very immature when these inconclusive studies were conducted. In particular, they made use of lower-limb exoskeletons that are not self-balanced and require crutches. The patient must have good motor control of the upper body and a sense of balance to use such devices, making them unusable during the first weeks after the strokes. This issue is a major obstacle to full recovery since motor plasticity quickly decreases after the stroke. Moreover, the patient must involve all his attention in the walking pattern to effectively stimulate brain plasticity and maximize recovery (Behrman et al., 2006; Kleim & Jones, 2008). Having to handle balance with crutches is exhausting, diverting the attention of the patient from their lower body, and eventually leading to pathological gait because the walking pattern is far from being natural. Finally, weight transfer and sense of balance are especially affected in stroke patients having lost lower-limb motor control capability. It is crucial for a powered exoskeleton to emulate these sensations by reacting to patient behavior instead of just blindly tracking its nominal motion, which is out of reach for devices with crutches (Calabrò et al., 2016).

This analysis suggests that making lower-limb powered exoskeleton crutch-less and safe-balanced is a prerequisite for efficient rehabilitation of stroke patients. The locomotion must be natural enough and span the full set of capabilities affected by the stroke in the days right after the event to stimulate brain plasticity and maximize motor recovery (Calabrò et al., 2016). The main advantages of such an exoskeleton over classical therapy are to allow intensive sessions with patients without the need for therapists and the ability to perform exercises that would be otherwise impossible or very complicated such as weight transfer and just standing up. Together, it increases the number of patients that can be treated and may enhance recovery. Furthermore, strong motivation and a positive attitude are known as keys to the efficiency of the therapy (Goodman et al., 2014; Tatla et al., 2013), and studies have demonstrated an increase in motivation from patients using robotic devices for rehabilitation (Lam et al., 2015; Postol et al., 2021).

1.1.2 Restoring Locomotion for Disabled People

Beyond rehabilitation, restoring locomotion for people suffering from spinal cord injuries or neurodegenerative diseases is also a major aspect. Indeed, due to an aging society, the number of people with limb movement disorders is increasing rapidly. Studies show that maximum muscle power is reduced by about 50% on average by natural aging, while the number of falls relative to the number of steps per day is increased by 800% (25 to 90 years) (Grimmer et al., 2019). Age-related diseases, such as Parkinson's disease, as well as the medication itself, can significantly affect walking ability. Furthermore, the World Health Organization reported that over 50



(a) CAD Rendering (b) User interface (c) Clinical trials

Figure 1.1: Atalante product by Wandercraft.

million people suffered non-fatal injuries because of road traffic crashes in total, many of them incurring permanent motor disabilities.

In this regard, walking naturally in an urban environment, while keeping their minds free of any stability concerns and without being quickly exhausting, contributes to giving them back a normal life. So far, no lower-limb medical exoskeleton is being entirely successful at achieving this. On the one hand, balance control is only local at best and devices are not able to perform reactive stepping strategies to recover stability in case of emergency (Goswami & Vadakkepat, 2019, Part VI). On the other hand, every motion is specifically tuned for one situation, and the robot usually has to stop completely for a short time before switching them. It breaks the flow of action and prevents dealing with unknown environments. Not being able to overcome any of these challenges makes it unlikely that such a device will be convenient enough to be widely adopted, as a wheelchair is still a more versatile alternative.

1.1.3 Atalante: Self-Balanced Medical Lower-Limb Exoskeleton

The exoskeleton Atalante, designed by Wandercraft (2021), is an autonomous device, self-balancing and self-supporting. It has 6 actuated revolute joints on each leg,

- 3 joints for the spherical rotation of the hip,
- 1 joint for the flexion of the knee,
- 2 joints for the hinge motion of the ankle.

Atalante is depicted in figure 1.1a. The actuators are powerful enough to sustain the weight of a user heavier than 100kg, with a maximum torque ranging from 100N m to

350N m. The weight of the exoskeleton is about 75kg, making it one of the heaviest devices on the market. This is not a major issue because the exoskeleton supports its own weight in contrast to exoskeletons with crutches, and therefore it is unnoticeable to the user inside. Still, it is more dangerous in case of a fall.

The exoskeleton Atalante has been designed by Wandercraft (2021). It is arguably the most advanced medical lower-limb exoskeleton commercially available. It has the CE marking to be sold in Europe. Wandercraft targets clinical centers in Europe for the time being, but FDA clearance for selling Atalante in the United States is on the way. Concurrently, a few additional units have been sold to universities in the United States for research purposes only. An ongoing medical study seems to confirm that walking on Atalante is having a significant positive impact on rehabilitation in the neurosurgical setting (Apra et al., 2022).

It features dimensional adjustments on the thighs and tibias to fit the morphology of the user. The user is fastened to the exoskeleton using straps on the pelvis, thighs, tibias, and feet. Those straps have to be slack and stretchy for the user's comfort and to reduce the risk of injury. This soft coupling between the user and the exoskeleton makes the whole system harder to control since the user can move inside the robot and disturb the dynamics of the exoskeleton. It is especially true for the upper body (both torso and arms) and the hips in the frontal plane to a lesser extent.

The exoskeleton has only basic proprioceptive sensors figure 1.2,

- 1 *Inertial Measurement Unit* (IMU) in the pelvis, tibia, and foot of each leg
- 4 vertical force sensors under each foot
- 1 encoder for each joint

An IMU integrates an accelerometer and a gyroscope, which provide a noisy measurement of the classical linear acceleration and the angular velocity in local frame respectively. An encoder provides a discrete noise-free measurement of the relative joint position. The velocity is obtained by numerical differentiation and filtering.

There are buttons on the right side of the exoskeleton for the physiotherapist to control the robot and stop it in case of emergency, but the user can also directly interact with it using a remote control and a jacket with an IMU on its back, see figure 1.1b. The user can select the type of gait – i.e. going forward, backward, turning around, or doing side steps – or switch between walking, standing, or exercise mode. In addition, one can use a mobile application to fine-tune in real-time the gait parameters, e.g. step duration and step length. This feature is a direct by-product of the first contribution of this thesis (cf. chapter 5).

Atalante lacks embedded safety apparatus to cushion the shock and hold the user's neck, and falling with it would be dangerous. Thus, it must stay attached to a mobile gantry. A physiotherapist is also requested to prevent repeated falls because the control algorithms have limited robustness to disturbances. Offering some emergency strategy to recover from falling in most cases is a key objective of this thesis (cf. chapter 6). Combined with airbags, it would dispense for any form of external assistance on the next product intended for personal use.

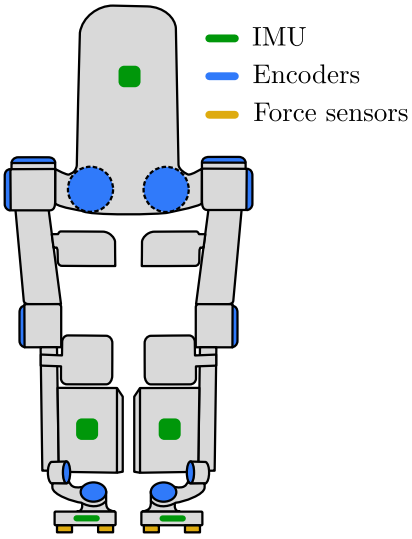


Figure 1.2: Overview of the hardware of Atalante

1.1.4 Technical Constraints and Challenges

Mechanical Flexibility of the Structure

One of the major challenges in robotics is the reality gap. For practical reasons, among them user safety, social acceptance, and agile locomotion, it is necessary to make the device relatively lightweight and slim. A direct consequence of this constraint is that the whole mechanical structure is flexible, including the transmissions.

In general, compliance is a property that is sought in robotics. Notably, it avoids breaking parts by relieving internal mechanical constraints in the structure, and it naturally adapts to the environment by compensating for small discrepancies between expectation and reality. In the particular case of an exoskeleton, it also prevents propagating shockwaves in the patient's bones by absorbing impacts on the ground.

However, controlling systems that are not perfectly rigid is more challenging, especially when it lacks an accurate dynamic model. It is usually the case when compliance was not introduced on purpose as for Atalante. Still, Seok et al. (2015) demonstrated that well-characterized mechanical compliance could be desirable since it relaxes hardware and software requirements for emulating compliance through control. For instance, Pratt and Williamson (1995) specifically developed a type of compliant actuators called *Serial Elastic Actuator* (SEA). This technology enables some real robots to perform highly dynamic motions such as jumping or running, e.g. the commercially available quadrupedal robot ANYmal (Hutter et al., 2016).

We observe on Atalante that the mechanical deformation of the structure only marginally disturbs the position of the *Center of Mass* (CoM) but significantly affects the swing foot figure 1.3. The latter is about 2cm lower than expected, and it touches the ground 20% earlier, which is enough to make the exoskeleton fall without early impact handling in the control loop. Currently, this is done by switching to the next

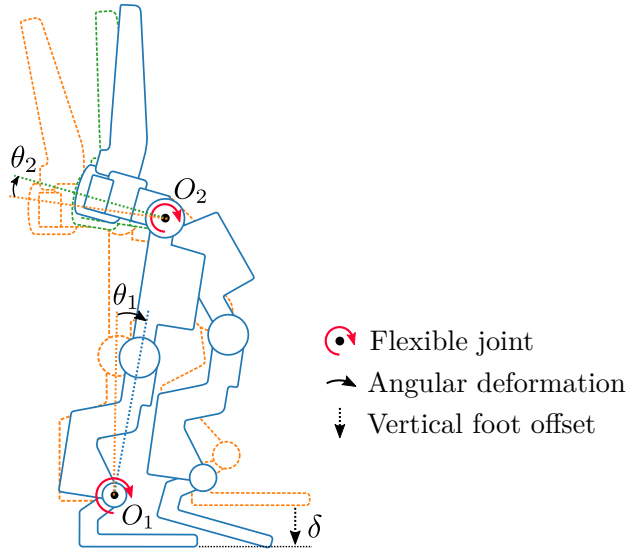


Figure 1.3: Modelling of the flexibility of the mechanical structure of Atalante as localized deformation at hip and ankle joints. The deformation is about 1deg for each flexible joint, lowering the vertical position of the flying foot by about 2cm. The dashed orange and green robots are respectively the theoretical configurations as expected by offline planning algorithms and after applying ankle deformation only. The solid blue one in front is the real configuration under mechanical deformation.

step as soon as the flying foot touches the ground. This strategy is fairly easy to implement, at the cost of further increasing the gap between the trajectory that was planned offline and the one performed in reality by the exoskeleton. This issue could be disregarded for a classic bipedal robot, but not for a rehabilitation exoskeleton for which it is critical to reproduce physiological walking.

Vigne et al. (2020b) have shown that modelling the mechanical deformation of Atalante is extremely difficult. The lack of a model means that the phenomenon cannot be explicitly taken into account in classical trajectory planning methods. Besides, the dynamics of the mechanical deformation must be decoupled from that of the controllers to ignore completely the former without rendering the whole feedback loop unstable. This restriction translates into an upper bound for tracking accuracy, amplifying indirectly the effect of the deformation at the kinematic level.

One option consists in estimating the mapping from rigid to flexible state by leveraging experimental data. First, the true nominal trajectory is executed on the real robot. Then, the difference between the observed and nominal trajectories is extracted from the data, and the corresponding feedforward time-dependent offset is subtracted from the command sent to each motor. It results in a new observed trajectory that should be closer to the original nominal one. It keeps going back and forth until convergence, and the whole process must be repeated for each nominal motion individually. In practice, simple linear ramps for the frontal and sagittal hip

joints of the swing leg work reasonably well on Atalante.

Alternatively, Vigne (2021) compensate online the observed deformation using a classical observer-controller structure. Their results are competitive with the previous method without the need for tedious and time-consuming manual tuning, which is promising. However, it cannot perfectly cancel the effects of the mechanical deformation due to hardware and modelling limitations. Relying too much on a biased observation in the feedback loop may cause vibrations (Vigne et al., 2020a). Thus, there remains a strong interest in developing robust control methods capable of handling poorly modeled phenomenons as a complementary approach. The design of such a controller – and the simulation environment to benchmark it – is an open question that we partially address in this thesis using the *Reinforcement Learning* (RL) paradigm (cf. appendix C and chapter 5). Randomizing the unknown parameters and making the task more difficult than necessary are some key components. For legged locomotion, an uneven ground profile could be used in simulation during training even though the robot is only supposed to walk on flat ground in reality.

Multi-Agent System with Partially Observable State

The exoskeleton Atalante with a user inside behaves quite differently from a classical humanoid robot. Indeed, it is not an actual bipedal robot but rather multiple independent agents interacting with each other and the external environment. More precisely, the user can only interfere with the exoskeleton, while the exoskeleton interacts directly with both the user and the environment.

The user is not fully controllable. Their behavior determines whether the system will fall or walk properly, no matter the theoretical stability of the nominal trajectory. It is fine as long as they assist the robot during its motion or at least do not disturb it. Yet, one cannot expect this assumption to hold true because a typical user is disabled. Most of them have fairly bad control of the upper body and sense of balance, in such a way that their behavior can be sometimes adversarial or helpful. Moreover, they tend to hold themselves using their arms by pushing on the batteries, which is also disturbing the dynamics of the robot.

There is no sensor on the user except one IMU on the back, so their state cannot be directly observed. Still, assuming the dynamics of the robot is perfectly known, its proprioceptive sensors are sufficient to estimate the state of the user at any point in time. Additionally, if a kinematic model of the user is available, then it may even be possible to deduce their full state from the history of past measurements. One can expect machine learning techniques to be able to learn it and therefore to be more robust than classic control approaches.

Once the state of the user is estimated, it can be perceived as something else than one external disturbance to be rejected among others. It should make it easier to maintain and recovery balance. More importantly, it enables counteracting the specific pathologies of the patients and adapting the behavior of the robot to the evolution of their condition. It should enhance motor recovery, adding up to the existing benefits of the platform for rehabilitation.

Offline Trajectory Planning

The exoskeleton is basically tracking pre-computed motions, disregarding early impact handling and admittance control to compensate locally for small disturbances (cf. section 2.2.3). A set of trajectories corresponding to primitive motions are generated offline once per user, relying on external computational resources hosted by Wandercraft. Notably, walking in a straight line, turning round, or doing side steps.

Although this approach was proven successful, it breaks the system's autonomy. This is acceptable from a product used in clinical centers, where access to the internet can be requested. Moreover, the process is repeated systematically, without being able to leverage the result of previous generations to speed up the next ones, which is a waste of resources for Wandercraft. Even more concerning is the potential waste of time for the physiotherapist if the delay induced by the generation process was not anticipated. It takes about 5 minutes for each first-time user, which is significant during 30 minutes sessions and may slow down the adoption of the product by clinical centers. In addition, it lacks versatility since only a finite discrete set of gaits is available on the device. This is fine for rehabilitation where the set of exercises is limited, but it makes it very difficult to navigate in a real environment. Finally, it lacks robustness as there is no guarantee for the optimization to convergence, even though it is very unlikely to fail in practice.

1.2 Problem Statement and Contributions

This thesis aims at solving some blocking points preventing to restore locomotion for disabled people in their daily life. Several challenges have been tackled in particular,

- embedding a continuous manifold of nominal trajectories on the exoskeleton to make it possible to navigate in unstructured environments (cf. chapter 5),
- ensuring smooth and natural transitions between nominal trajectories without having to go back to stand-still systematically (cf. chapter 6),
- providing a robust control policy to guarantee the safety of the user by maintaining balance at all costs, while actively compensating for discrepancies between planning and reality (cf. chapter 6).

Hereafter, we summarize our contributions.

1.2.1 Online Trajectory Planning

Online trajectory planning enables robots to deal with a real-world environment that may change suddenly and to carry out sequences of tasks in unknown orders and contexts. For instance, walking robots must be able to change direction or adapt their speed, but also to consider stairs of different heights or the position and size of obstacles. Conventionally, online trajectory planning capability is achieved by solving trajectory optimization in real time. This can be done by either relying on embedded resources or streaming data through the network to a dedicated computation unit. A hybrid mix of both is also relevant as it enables continuous improvement over the

whole fleet of devices already deployed by sharing experience globally. For instance, Boston Dynamics and Tesla are streaming data related to computer vision processing whenever a network connection is available. In classical robotics, mainly two different planning approaches are used for bipedal robots: whole-body optimization by solving directly over the full-body state (Dalibard et al., 2013), or reduced model optimization which is a two-stage method consisting in first planning the *centroidal dynamics* – i.e. the angular momentum and the position of the center of mass – according to a highly simplified model, and then finding a consistent full-body state (Ahn et al., 2021; Apgar et al., 2018; Caron, 2020).

Offline trajectory planning based on whole-body optimization is already very challenging for complex systems that may involve hybrid dynamics, under-actuation, redundancies, balancing issues, or a need for high accuracy such as bipedal robots. Although efficient methods exist to solve most trajectory optimization problems such as Differential Dynamic Programming and Direct Collocation Transcription (Budhiraja et al., 2019b; Hereid & Ames, 2017; Hereid et al., 2018; Huynh et al., 2021), there is no guarantee of convergence and finding solutions is computationally demanding, preventing their uses online. Hereid et al. (2016b) get around these issues by running the optimization in the background and updating the trajectory periodically, e.g. between each step for bipedal robots. However, it remains hard to meet such computational performance, and this still provides a poor reaction time. Reduced model approaches can be used to speed up the calculations and ensure convergence, for example by linearizing the dynamics. Nevertheless, it does not have any guarantee to be feasible in practice because it does not take into account the actual dynamics of the system, and the overall motion is less natural (Boer, 2012; Kajita et al., 2003).

A workaround to avoid online trajectory optimization consists of using a function approximation, i.e. performing trajectory learning over a set of trajectories generated beforehand. This requires no simplification of the model since the optimizations are carried out offline. Moreover, once training has been done, it operates at a fraction of the cost of the previous methods. Two distinct approaches can be considered: policy learning, i.e. training a controller, and trajectory learning, i.e. predicting nominal state sequences. While the potential of policy learning is impressive, it implies a complete overhaul of the control architecture and new requirements in terms of embedded and on-premise computational resources, not to mention certifiability concerns. Trajectory learning has the advantage of being effortless to integrate into robotic systems for which there already exists control strategies that ensure robust tracking of trajectories generated through optimization: it comes down to replacing a finite set of trajectories with the function approximation. A naive approach would be to train a function approximation on a database of solutions to the optimization problem. Although it may work in practice, this is sensitive to overfitting and does not offer any guarantee to really perform the desired task or to be feasible. This is nonetheless state-of-the-art in trajectory learning as this field is still largely unexplored.

Our contribution is the *Guided Trajectory Learning* (GTL) algorithm, which makes trajectory optimization adapt itself, so that it only outputs solutions that can be perfectly represented by a given function approximation. The idea is to make

the trajectory optimization problem adapt itself wherever the function approximation fails to fit. Our method is readily applicable to any complex robotics system with a high-dimensional state for which offline trajectory optimization methods and satisfactory control strategies are already available and efficient. It makes online trajectory planning based on function approximations more accurate and reliable by guaranteeing the feasibility of the predictions, thereby being practical for systems where failure is not an option. We demonstrate it on flat-foot walking with the exoskeleton Atalante (cf. chapter 5).

This contribution has been the subject of the following publications and patents:

- *Online Trajectory Planning Through Combined Trajectory Optimization and Function Approximation: Application to the Exoskeleton Atalante* in the **International Conference on Robotics and Automation**
- *Procédés d'apprentissage de paramètres d'un reseau de neurones, de génération d'une trajectoire d'un exosquelette et de mise en mouvement de l'exosquelette* as **international patent under the Patent Cooperation Treaty (No. WO2021058918A1)**

1.2.2 Robust and Safe Control Policy Optimization

Achieving dynamic stability for bipedal robots is one of the most difficult challenges in robotics. Continuous feedback control is required to keep balance since the vertical posture is inherently unstable. However, hybrid high-dimensional dynamics, kinematic redundancy, model and environment uncertainties, and hardware limitations make it hard to design robust embedded controllers. Trajectory planning for bipedal robots has been solved successfully through whole-body optimization. In particular, Gurriet et al. (2018) achieved stable walking on flat ground and without disturbances on the exoskeleton Atalante by using state-of-the-art traditional methods. Yet, robust tracking of reference motions and emergency recovery is still an open problem. Classic control approaches require a lot of expert knowledge and effort in tuning because of discrepancies between approximate models and reality. Solutions are mainly task-specific, and improving versatility is usually done by stacking several estimation and control strategies in a modular hierarchical architecture (Herzog et al., 2016; Kim et al., 2020; Lohmeier et al., 2009; Moro & Sentis, 2018). Despite being efficient in practice, it makes the analysis as well as tuning increasingly challenging and thereby limits its capability. Machine learning methods, such as deep RL, in contrast, require expert knowledge and extensive efforts to design the agent and the reward rather than structuring explicit controllers and defining good approximate models. RL aims at solving observation, planning, and control as a unified problem by training an end-to-end control policy. Tackling the problem globally substantially increases its potential, however, state-of-the-art algorithms still face difficulties to converge and obtain satisfying behavior for practical applications. Moreover, a ubiquitous problem of controllers trained with deep RL is the lack of safety and

smoothness. This is a problem for real-life deployment as human beings aren't able to predict future motions. Without special care, the command varies discontinuously like a bang-bang controller, which can result in a poor transfer to reality, high power consumption, loud noise, and system failures (Mysore et al., 2021). Despite these potential limitations, a robust gait and push recovery for the bipedal Cassie robot was recently learned in simulation using deep RL and then transferred successfully to the real device (Castillo et al., 2021; Li et al., 2021). Concurrently, several works on standing push recovery for humanoid robots trained in simulation suggest that the approach is promising (Ferigo et al., 2021; Melo et al., 2020), although the same level of performance has not been achieved on real humanoid robots.

Our main contribution is the development of a purely reactive controller for standing push recovery on legged robots using RL, which is used as the last resort fallback in case of emergency. Precisely, we design an end-to-end policy featuring a variety of human-like balancing strategies from the latest proprioceptive sensor data, while guaranteeing predictable, safe, and smooth behavior. The resulting policy greatly expands the set of recoverable states in comparison to classical model-based controllers on similar systems. Moreover, the policy can be directly transferred to a real robot. We demonstrate agile push recovery behavior for strong perturbations on the self-balanced medical exoskeleton Atalante (cf. chapter 6).

This contribution has been the subject of the following publications and patents:

- *Reactive Stepping for Humanoid Robots using Reinforcement Learning: Application to Standing Push Recovery on the Exoskeleton Atalante* in the **International Conference on Intelligent Robots and Systems**
- *Methods for training a neural network and for using said neural network to stabilize a bipedal robot* as **European patent** (under examination)

1.2.3 Realistic Open-Source Simulator for Reinforcement Learning

Simulation is a critical tool in robotics. Robots are no longer limited to operating in structured environments and performing scripted actions. In this context, being able to compare and analyze different software and hardware solutions in a virtual playground can help to bring out the physical constraints, planning challenges, and control limitations that the robot may face before actually building it. Thus, it reduces the cost and accelerates the engineering design cycle by quickly discarding unsuccessful solutions. Furthermore, artificial intelligence techniques such as RL are now mature for real applications (cf. section 3.2). It promises to endow the next generation of robots with locomotion and decision-making skills, but those algorithms are usually extremely data-hungry. Simulation enables generating a large amount of training data in a fraction of the real-time, without safety concerns or wearing out the device. Finally, while physically testing robots before deployment is mandatory, building a controlled environment for every specific task to perform ensures thorough testing, which avoids regressions over time and enables benchmarking of new solu-

tions. In parallel, data-driven approaches consisting in training a surrogate model to replace simulators with an oracle are slowly emerging. One of their main use cases is speeding up calculations that would otherwise be very costly, e.g. fluid and quantum mechanics. Ha and Schmidhuber (2018) train a function approximation of a dataset of observations without any additional requirement, which they called *world model*. This approach is condemned by the robotic community, mainly because of a lack of understanding of its internal workings following its black box design, limited theoretical validation, and unreliable accuracy. Raissi et al. (2019) introduce *Physics-Informed Neural Networks* (PINNs) to overcome most of these shortcomings. To do so, they enforce in addition that the model respects any given law of physics described by general nonlinear partial differential equations, possibly comprising unknown physical constants. This approach can be applied to obtain either a discrete or continuous model while ensuring arbitrary accuracy in both cases. Yet, if the model is trained from synthetic data, then it has little advantage over an actual physics engine which is already cheap to evaluate for legged robots, and providing enough real data to identify the unknown parameters is challenging.

Optimizing control policies using RL for legged robots requires a fast and realistic simulator. Furthermore, the dynamics must be smooth to minimize the signal-to-noise ratio and thereby reduce the number of samples necessary to accurately estimate the gradient of the problem. Several physical simulators were already available when we started investigating this topic, among them Mujoco (Todorov et al., 2012), Dart (Lee et al., 2018), Drake, ODE, Simbody, and Bullet. Mujoco is ubiquitous in the Machine Learning community. Yet, relying on it for anything else than synthetic benchmarking is not acceptable due to its unrealistic contact model. Moreover, at that time, it was commercial software, closed-source, and poorly documented. Dart, Drake, or Simbody are not great options either because they are notably slow. ODE is based on Newton dynamics instead of Lagrangian dynamics, which is known to lead to non-smooth physics with poor signal-to-noise ratio, bad numerical stability, and non-repeatable results. On its side, Bullet was a promising option but not thoroughly tested nor widely adopted by the machine learning community. Ultimately, all these simulators were showing limitations regarding the physics and the set of features readily available. For instance, considering the gantry, the physiotherapist or the user inside the exoskeleton was theoretically possible but not supported out-of-the-box. More importantly, none of them were modeling the mechanical deformation of the structure. Besides, none of these tools were sharing a common interface with the libraries already developed at Wandercraft internally, substantially hindering the transfer of knowledge from the existing codebase to the simulation environment. Typically, being able to export simulation log files in the same format as the real robot would enable using the analysis toolchain already available.

A new simulator specifically tailored for legged robots has been developed during this thesis. This simulator, called Jiminy, is open-source and freely available. The minimal features to perform policy optimization are provided. Notably, it implements the standard interface required by all RL libraries, and examples of learning environments are available for a few commercially available legged robots. And last but not

least, Jiminy comes with basic tools to visualize and analyze the results conveniently. Its performance is fairly competitive against other available libraries, with a real-time factor being close to 50 on a single thread for the exoskeleton Atalante. Under the hood, Jiminy leverages the rigid body dynamics library Pinocchio (Carpentier et al., 2019). This library is heavily used by the research teams at Wandercraft, and hence having a simulator based on it greatly eases internal adoption (cf. appendix C).

This contribution had been released in open-source under MIT license:

- *Jiminy: a Fast and Portable Python/C++ Simulator of Poly-Articulated Systems with OpenAI Gym Interface for Reinforcement Learning* at **https://github.com/duburcqa/jiminy**
- *Tianshou: a Highly Modularized Deep Reinforcement Learning Library* in the **Journal of Machine Learning Research (Machine Learning Open Source Software Paper)**

Outline of the Thesis

This thesis is organized into six chapters. First, chapter 2 defines the theoretical model of the system patient-exoskeleton, then it introduces classic planning and control methods of bipedal robots. Next, chapter 3 gives an overview of supervised learning and neural networks, then presents reinforcement learning. These two preliminary chapters cover the necessary technical tools to fully comprehend the related works and our contributions. Chapter 4 is dedicated to discussing the state-of-the-art approaches related to our contributions. The chapter is divided into two parts. First, we review existing results about combining trajectory optimization and function approximation, followed by policy learning approaches from which our contribution takes inspiration. The second part focuses on presenting some techniques to ensure robustness and safety of control policy using reinforcement learning and how to do the sim-to-real transfer. Chapter 5 and chapter 6 constitute our main contributions. Chapter 5 introduces a new algorithm combining trajectory optimization and function approximation to enable online trajectory planning. Chapter 6 presents results on learning versatile locomotion skills for legged robots while ensuring robustness and safety of the control policy. Finally, Chapter 7 proposes a discussion and some perspectives on our contributions. Appendix A formalizes mathematically the trajectory optimization problem and illustrates it for walking in a straight line on flat ground with the exoskeleton Atalante. Appendix C gives some details about the physics modeling and features available in the Jiminy Simulator that was developed as part of this thesis. Appendix D summarizes some prominent network architectures in machine learning. Appendix E reviews in depth the algorithms that led to major breakthroughs in RL from its foundation almost 50 years ago up to this day while drawing connections between them.

Chapter 2

Background in Robotics

Contents

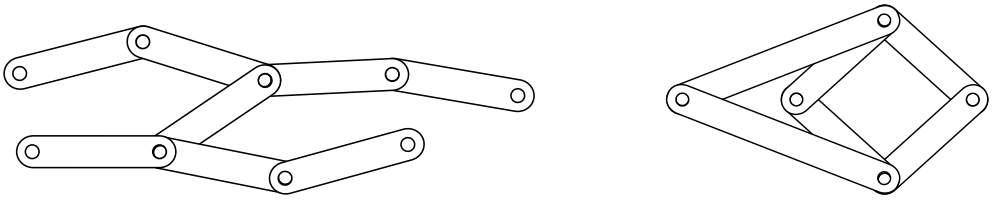
2.1 Preliminaries on Rigid Body Dynamics 15
2.1.1 Spatial Vector Algebra 18
2.1.2 Whole-Body Dynamics 26
2.2 Planning and Control in Bipedal Robotics 28
2.2.1 Notion of Bipedal Locomotion and Terminology 28
2.2.2 Stability Assessment 33
2.2.3 Classical Control Methods 42

2.1 Preliminaries on Rigid Body Dynamics

The *Rigid Body Dynamics* refers to the study of the kinematics and dynamics for *rigid poly-articulated system*. Great progress has been made in this field since the availability of cheap computational resources. Nowadays, algorithms to compute any kinematics and dynamics quantities with the lowest possible complexity have been found (Goswami & Vadakkepat, 2019, Part IV; Featherstone, 2008; Carpentier & Mansard, 2018). They are fast enough to enable online control for complex systems such as humanoid robots using embedded computational resources. Nevertheless, how to efficiently model and integrate over time the interaction with the environment is still an active research topic. In line with this statement, we offer our very own contact solver with our open-source simulator Jiminy in appendix C.

Kinematic Trees

A *rigid poly-articulated system* is mathematically represented as a directed graph (Goswami & Vadakkepat, 2019, Part IV). The vertices of this graph are rigid *bodies*, also called *links*. A rigid body is not supposed to deform or change shape, which is an idealization. Any assembly of multiple mechanical parts fixed relative to one another is viewed as one rigid body. Each body is fully characterized by a small set of constant physical parameters (cf. section 2.1.1). The edges are *joints*. A joint defines the relative motion that is permitted between its parent and child bodies. It is a mathematical abstraction of one physical mechanism as well as possible its



(a) Open kinematic tree (b) Closed kinematic graph

Figure 2.1: Examples of kinematic graphs

transmission system doing the interface with actuators if any. Thus, it gives you limited information about the actual design of the robot: infinitely many mechanisms would be modeled the same way. Each type of joint characterizes one particular set of independent movements among the 3 translations (*X, Y, Z*) and the 3 rotations (*Roll, Pitch, Yaw*) plus any combination between them (cf. section 2.1.1). These independent movements are termed *Degrees of Freedom* (DoFs). Their number cannot exceed 6 in our 3D space. The most common type of joint is the revolute joint which only enables a single rotation. More complex types exist, such as the helicoidal joint, allowing a screwing motion coupling a translation and a rotation and therefore still having a single DoF. Simple directed paths in a kinematic graph are known as *sub-chains*. They are the assembly of several rigid bodies articulated by distinct joints. A sub-chain is said to be closed if cyclic, and open otherwise. Robots having closed kinematic chains are referred to as parallel, serial otherwise (see figure 2.1).

The most famous example of a parallel robot is the hexapod positioning system called Stewart platform. Parallel kinematic chains combine high rigidity with small mass and inertia of the manipulator relative to the load. Thus, they allow high precision and high speed at the same time without having to compromise between the two, unlike serial robots. However, the workspace tends to be limited, in part because there may be singular configurations for which set the system in motion would require infinitely large efforts for the actuators, leading to the destruction of the mechanical structure or hardware if reached.

The determination of the singularities is an open problem in the general case. Moreover, the structure of a parallel robot is said to be *hyperstatic* or *statically indeterminate*: the static equilibrium equations are insufficient for determining the internal forces acting on that structure. Further information, such as material properties and mechanical deformations, must be taken into account to find out which is the unique physically meaningful solution among all the feasible ones (Matheson, 1959). It is necessary to anticipate which parts will break first and when, but that is all. Indeed, any feasible solution is suitable to simulate the system since all of them would lead to the same temporal evolution of the system.

Legged robots may comprise closed-kinematic sub-chains. For most of them, it is possible to completely ignore these sub-chains without modifying the overall dynamics of the system by virtually relocating the associated motors on some passive joints. For the others, kinematic constraints must be added, but this specific case is

ignored for simplicity. Therefore, we will focus on serial robots in the following, and all kinematic chains will be assumed open unless stated otherwise.

Formally, such an acyclic kinematic graph is an arborescence (directed root tree) but commonly referred to simply as a *tree*. Its leaves are the bodies that are not the parent of any joint and are called *end-effectors*. Typical examples are the feet of a legged robot. Conversely, a kinematic tree has only one root. It is the unique body that is not the child of any joint and is called *base*. The base can be fixed relative to the world, which is the case for robotic arms, or free-floating for robots moving capability of locomotion such as legged robots. Free-floating base are usually called *freeflyer*. For a consistent formalism between fixed and free-floating robots, a virtual fixed body representing the world is systematically prepended to the kinematic tree of free-floating robots, and the world is connected to the actual freeflyer of the robot through a joint allowing all the 6 DoFs without restriction. Accordingly, it will always be assumed that the base is fixed in the following, while still referring to robots as either fixed or free-floating, whichever is appropriate.

Since the bodies of a kinematic chain are connected in series, exploiting the sparsity pattern along the whole structure of the robot is crucial. The canonical rigid body dynamics algorithms (cf. section 2.1.2) developed by Featherstone (2008) are widely accepted as state-of-the-art in terms of algorithmic complexity. Basically, it consists in recursively computing physics quantities by going back and forth (possibly several times) along the chain using the *Divide-and-Conquer paradigm* (Cormen et al., 2009). Recently, Carpentier and Mansard (2018) introduced efficient algorithms to compute their analytical derivatives, along with some additional quantities. Having access to these derivatives is fundamental in appendix A for whole-body trajectory planning, which is at the heart of our first contribution (cf. chapter 5).

Generalized Coordinates

The *Generalized coordinates* are a set of parameters that determines the configuration of a physical system at any point in time. Formally, the generalized coordinates q form a *coordinate chart*, i.e. it is a homeomorphism from a *topological space* $\mathcal{Q}$ called *coordinate manifold* to a subset of the euclidean space $\mathbb{R}^n$. For poly-articulated robots, the coordinate manifold gathers the cartesian positions plus orientations of all its moving bodies (cf. section 2.1.1). Infinitely many charts can be defined for the same manifold. The coordinates do not even have to be mutually independent. For instance, the configuration of a wheel around a fixed axis can be described by its rotation angle θ. This choice is the most common but flawed: either the angle presents discontinuities or grows unboundedly. The alternative parameterization $(\cos(\theta), \sin(\theta))$ is somewhat harder to manipulate but does not face such issues.

The generalized velocities $\dot{q}$ and accelerations $\ddot{q}$ are the time derivatives of the generalized coordinates q. Notations such as v or a are dismissed to avoid confusion with classical body velocity or acceleration. If the coordinates manifold is differentiable, then they locally belong to the *tangent space* $T_q\mathcal{Q}$, which is a real vector space of $\mathbb{R}^n$, where n is the number of DoFs of the system.

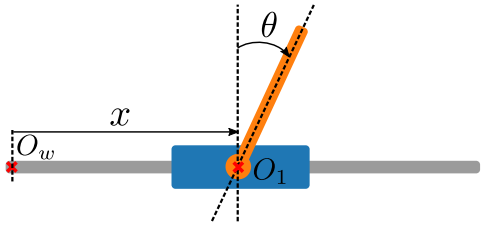


Figure 2.2: Canonical generalized coordinate for the Cartpole. It corresponds to the vector $[x, \theta]$, where x is the linear position of the cart along the rail and θ is the angle of the pole relative to the vertical axis.

For a poly-articulated robot, the canonical generalized coordinates are obtained by concatenating the parameterization of the state of all the joints. The parameterization for a given joint is not unique to its type. It is said to be *minimal* if its dimensional matches the number of DoFs of the corresponding joint, and any value within range maps to a valid state of the joint. Otherwise, one additional constraint per extra dimension must be enforced. For spherical joints, the quaternion is the most common parameterization although not minimal (its norm must be 1) as opposed to axis-angle (cf. section 2.1.1), mainly because operations involving rotations are numerically the cheapest when specialized for quaternions. Similarly, for revolution joints, the scalar relative angle θ is ubiquitous, but the vector $[cos(\theta), sin(\theta)]$ is sometimes preferred if unbounded for numerical stability.

The generalized coordinates are at the heart of Lagrangian mechanics, which aims to express the dynamics of the system as a function of the generalized coordinates. This formulation prevents the constraints between bodies due to joints and the corresponding internal forces from appearing explicitly in the equations of motion while ensuring they are satisfied exactly. This property not only speeds up physics computations but also improves the computational accuracy and stability of rigid body dynamics algorithms compared to Newtonian mechanics. In the latter case, the configuration of the system is rather specified by the placement of all the bodies, and the resultant forces acting on each of them are derived to obtain independent equations of motion of each body. See section 2.1.2 for details.

2.1.1 Spatial Vector Algebra

Spatial Motion

The dynamic equations of motion are usually expressed by keeping separated the linear and angular parts of physical quantities, e.g. linear vs. angular velocities or forces vs. moments. This unnecessary distinction artificially doubles the number of terms and equations (although the actual number of scalar equations is obviously unchanged) and hence the dynamics looks more complicated than it is. This often leads to mistakes because it is harder to understand and manipulate these terms, and it impedes the numerical efficiency of some computations. Besides, it prevents from considering the coupling between the translation and rotation part of transforms (cf.

section 2.1.1), which has a huge impact when computing distances for instance (cf. section 6.3.1). In the following, 6D notations called *spatial vectors* are introduced. Although non-intuitive, it benefits from solid mathematical foundations that are helpful to infer important properties.

The *spatial motion* vector space $\mathcal{M}^6$ is used to describe quantities homogeneous to velocities or any higher order time derivative. More specifically, the spatial velocity for a rigid body B is called *twist*. Given a fixed point O anywhere in space, the twist is fully specified by a pair of 3D vectors: the linear velocity vector $v_O \in \mathcal{R}^3$ of the point O, and the angular velocity vector $\omega \in \mathcal{R}^3$. The angular one is independent of the application point O, and as such it has no subscript. Indeed, its direction and magnitude characterize respectively the axis passing through the origin O around which the body B rotates and at what speed. In contrast, the linear velocity vector v_O truly depends on the application point. The definition of the spatial velocity and its subsequent properties generalize naturally to any spatial motion vector.

The origin must be a body-fixed point for a valid vector algebra over coordinate vectors to be defined, namely linear transformations plus scalar and cross products. Nevertheless, this origin can still be chosen differently over time. Notably, it can correspond to a virtual point that happens to coincide with any other point of interest at the current time only. It is essential to keep in mind this subtle distinction to avoid any mistakes when differentiating coordinate vectors.

Let us consider a *cartesian frame* O_{xyz} with orthonormal basis $\{i, j, k\}$ and origin O. It defines a *cartesian coordinate system*, that we identify to the Cartesian frame itself for simplicity. In this Cartesian frame, we have $v_O = v_{O_x} i + v_{O_y} j + v_{O_z} k$ and $\omega = \omega_x i + \omega_y j + \omega_z k$. The so-called Plücker basis is defined as $\{d_x, d_y, d_z, d_{O_x}, d_{O_y}, d_{O_z}\} \subset \mathcal{M}^6$, where d_x, d_y, d_z denote the unit translations in the directions x, y, z, and $d_{O_x}, d_{O_y}, d_{O_z}$ are the unit rotations about the directed lines O_x, O_y, O_z. This basis and its coordinate system on $\mathcal{M}^6$ are used to write down the linear and angular velocity vectors v_O, ω as one unified 6D vector called a spatial velocity vector and denoted $\hat{v}$. The spatial velocity vector $\hat{v}$ in this basis and its corresponding coordinate vector $\underline{\hat{v}}_O$ are given by,

$$\hat{v} = v_{O_x} d_x + v_{O_y} d_y + v_{O_z} d_z + \omega_x d_{O_x} + \omega_y d_{O_y} + \omega_z d_{O_z}, \tag{2.1}$$

$$\underline{\hat{v}}_O = [\underline{v}_O, \underline{\omega}] = [v_{O_x}, v_{O_y}, v_{O_z}, \omega_x, \omega_y, \omega_z], \tag{2.2}$$

where $\underline{v}_O = [v_{O_x}, v_{O_y}, v_{O_z}]$ and $\underline{\omega} = [\omega_x, \omega_y, \omega_z]$.

There is no mention of the Cartesian frame and application point in the notation of the spatial velocity vector $\hat{v}$ to emphasize that it does not depend on them. Indeed, the spatial velocity defines a vector field (the linear velocity field of body B), which is a quantity intrinsic to the body as a whole rather than a property of individual body-fixed points. This vector field $\mathbf{V}$ can be derived from the linear velocity v_O at a given body-fixed point O and the angular velocity ω,

$$\mathbf{V}(P) = v_O + \omega \times \overrightarrow{OP}, \tag{2.3}$$

where $\overrightarrow{OP} \in \mathcal{R}^3$ is the relative position of P with respect to O, and $\mathbf{V}(P)$ is the value of the vector field $\mathbf{V}$ at a body-fixed point P, namely the linear velocity vector

of the body B of the point P. This is easy to check that the vector field is invariant to the application point O despite appearing in its definition. On the contrary, its coordinate vector $\underline{\hat{v}}_O$ can be interpreted as a measure of a flow passing through the point O rather than the velocity of one particular body-fixed point.

Interestingly, the spatial velocity of a body i in kinematic chain with N joints is related to the joint-space velocity $\dot{q}$ of the system through a $6 \times N$ matrix J_i called *body Jacobian* for body i,

$$v_i = J_i \dot{q}. \tag{2.4}$$

The structure of the Jacobian matrix is sparse. Indeed, the velocity of a body only depends on the previous joints in a kinematic tree, and it is not affected by the other sub-chains. Its sparsity pattern also depends on the type of joints that are involved. Being able to take advantage of the sparsity is critical for the scalability of rigid body algorithms as it can significantly lower their algorithmic complexity.

Spatial Forces

The same reasoning applies to the *spatial force* vector space, denoted $\mathcal{F}^6$. Given a rigid body B and a fixed point O, the spatial force, also called *wrench*, consists of a linear force f acting along a line that passes through O, together with a couple τ_O equal to the total moment about O. The total moment about any other point P can be calculated using the force analogous equation of equation (2.3),

$$\tau(P) = \tau_O + f \times \overrightarrow{OP}. \tag{2.5}$$

It shares analogous properties to the spatial motion, e.g. being invariant to the location of the application point O.

In the same way as before, the Plücker basis $\{e_{O_x}, e_{O_y}, e_{O_z}, e_x, e_y, e_z\} \subset \mathcal{F}^6$ is used to obtain a spatial force vector $\hat{f}$ gathering the linear force f and torque τ_O. $e_{O_x}, e_{O_y}, e_{O_z}$ are unit torques in directions x, y, z, and e_x, e_y, e_z are unit linear forces acting along the lines O_x, O_y, O_z. In this basis, the spatial force vector $\hat{v}$ is given by

$$\hat{f} = f_x e_{O_x} + f_y e_{O_y} + f_z e_{O_z} + \tau_{O_x} e_x + \tau_{O_y} e_y + \tau_{O_z} e_z, \tag{2.6}$$

with corresponding coordinate vector,

$$\underline{\hat{f}}_O = [f_x, f_y, f_z, \tau_{O_x}, \tau_{O_z}, \tau_{O_z}]^T = [\underline{f}, \underline{\tau}_O]^T, \tag{2.7}$$

where $\underline{f} = [f_x, f_y, f_z]^T$ and $\underline{\tau}_O = [\tau_{O_x}, \tau_{O_z}, \tau_{O_z}]^T$.

Duality Properties

Formally, the vector spaces $\mathcal{M}^6, \mathcal{F}^6$ associated with spatial motions and forces are dual. As such, many important properties can be deduced. The hat symbol over spatial vectors and underlining of coordinate vectors will be dropped in the following if already implicit. Conversely, if there is any ambiguity, a subscript on a spatial

vector will specify to which body or moving frame it is related, and a prescript on a coordinate vector will denote the coordinate system.

The scalar product that takes one element from each vector space is properly defined. In any event, it is homogeneous to an energy or some higher-order time derivative such as a power. Yet, it must involve quantities related to the same body to be physically meaningful. Given $m \in \mathcal{M}^6$ and $f \in \mathcal{F}^6$ and their dual coordinate representations $\underline{m}, \underline{f}$, the scalar product is defined as follows,

$$m \cdot f = f \cdot m = \underline{m}^T \underline{f} = \underline{f}^T \underline{m}. \tag{2.8}$$

Therefore, one can interpret the scalar product as applying the operator $a\cdot = a^T$ that maps b to $a \cdot b$.

Two different cross products are defined on spatial vectors: one takes two motion vectors and returns a motion vector, and the other one takes a motion vector as left-hand and a force vector as right-hand to produce a force vector. Let A be a Cartesian frame that is moving with a spatial velocity of v_A, and ${}^A m, {}^A f$ be two coordinates vectors representing the spatial vectors $m \in \mathcal{M}^6, f \in \mathcal{F}^6$ in Cartesian frame A respectively. The coordinate vectors ${}^A \dot{m} \in \mathcal{M}^6$, ${}^A \dot{f} \in \mathcal{F}^6$ that represents their time derivative in the Cartesian frame A are given by

$${}^A \dot{m} = {}^A\left(\frac{\mathrm{d}m}{\mathrm{d}t}\right) = \frac{\mathrm{d}\,{}^A m}{\mathrm{d}t} + {}^A v_A \times {}^A m, \qquad {}^A \dot{f} = \frac{\mathrm{d}\,{}^A f}{\mathrm{d}t} + {}^A v_A \times^* {}^A f, \tag{2.9}$$

where $\times$ is the cross product operator and $\times^*$ can be regarded as its dual. These two cross-products are involved in the temporal differentiation of spatial vectors.

Time Derivative

$\frac{\mathrm{d}\,{}^A m}{\mathrm{d}t}, \frac{\mathrm{d}\,{}^A f}{\mathrm{d}t}$ are the element-wise time derivative of the coordinate vectors representing the spatial vectors m, f in Cartesian frame A, respectively. It is called *apparent derivative* since it can be regarded as the apparent rates of change of m, f as perceived by an observer attached to the moving frame A hence having a velocity of v_A. If the Cartesian frame A is fixed, then the spatial and actual derivatives match. Conversely, if the spatial vector is constant in the Cartesian frame A and the latter is moving, then the first term of the left-hand of the equalities vanishes. As for the scalar product, one can define the operators $\hat{v}\times$ and $\hat{v}\times^*$ as 6×6 skew-symmetric matrices (Siciliano & Khatib, 2008, Part A, Chapter 2),

$$\hat{v}\times = \begin{pmatrix} v_O \times_3 & \omega \times_3 \\ 0 & \omega \times_3 \end{pmatrix}, \qquad \hat{v}\times^* = -(\hat{v}\times)^T, \tag{2.10}$$

where $\times_3$ is the skew matrix representation of the cross-product for 3D vectors,

$$a\times_3 = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & -a_x & 0 \end{pmatrix}. \tag{2.11}$$

The matrix operators $a\times, a\times^*$ are not efficient from a numerical point of view but are convenient to derive more complex operators in compact forms. They are used without distinction from the actual cross-product operators $\times, \times^*$ in the following.

As a side note, the *classical acceleration* people are accustomed to is the apparent derivative of the spatial velocity in a Cartesian frame with an orientation fixed in space but an origin attached to the body. Whereas the spatial acceleration preserves the structure of the vector space in it, it is not the case for the classic acceleration. For instance, the addition of spatial accelerations is well-defined, just like for spatial velocities. Moreover, there is no fictitious force coming from working in a non-inertial reference frame to worry about, which is a major advantage. The classical acceleration $\hat{\underline{a}}'$ can be obtained from equation (2.9),

$$\hat{\underline{a}}' = \hat{\underline{a}} - \begin{pmatrix} v_O \\ 0 \end{pmatrix} \times \begin{pmatrix} v_O \\ \underline{\omega} \end{pmatrix} = \hat{\underline{a}} + \begin{pmatrix} \underline{\omega} \times v_O \\ 0 \end{pmatrix}. \tag{2.12}$$

Frame Placement

The pair position plus orientation of a frame in the world is called *pose*. Formally, the position corresponds to the cartesian coordinates in the 3D euclidean space $\mathbb{R}^3$, but it is more complicated for the rotation. The set of all possible rotations about the origin of the 3D euclidean space form a *Lie group*. As such, special operators must be used to perform linear operations, integration, or differentiation. The rotation group is called 3D Special Orthogonal group and denoted $SO(3)$ because it is homeomorphic to the set of orthogonal matrices of size 3 with determinant $+1$, i.e. rotation matrices. The group of poses is homeomorphic to $\mathbb{R}^3 \times SO(3)$ since translations and rotations are independent. It is called the 3D Special Euclidean group and is denoted $SE(3)$.

Being able to represent rotations using a coordinate system, called chart on $SO(3)$, instead of rotation matrices is important for several reasons. First, it is much easier to comprehend. Secondly, it is more compact and therefore numerical implementations can be more efficient. It exists many coordinate systems, but all of them are facing multiple-value issues, i.e. different coordinates can represent the same rotation. Moreover, some of them have singularities making them only valid locally. For instance, the *Euler angles* representation $(\text{roll}, \text{pitch}, \text{yaw})$ is widely used because it is very intelligible, but the uniqueness property breaks down for some specific coordinates, which is referred to as gimbal lock. Another famous parameterization are unit quaternions (x, y, z, w), sometimes called *versors*. It can be used to represent 3D rotations up to sign, so the unit quaternion group is a double covering map of $SO(3)$. This is less an issue that the gimbal lock problem in practice, and therefore it is often preferred over Euler angles, at least for internal implementations. Although it is hard to interpret unit quaternions directly, it is easy to relate them to their axis-angle representation $(\hat{e}, \theta)$,

$$q = [x, y, z, w]^T = e^{\theta/2\,\hat{e}} = \left[sin\left(\frac{\theta}{2}\right)\hat{e}, cos\left(\frac{\theta}{2}\right) \right]^T, \tag{2.13}$$

where $\hat{e}$ is the axis of the rotation, and θ is the angle. The typical axis-angle coordinates are the product of the angle by the axis $\theta\hat{e}$. It can be obtained from a unit quaternion through the inverse of the exponential map of its Lie Algebra. It means that the axis-angle coordinates can be interpreted as coordinates on its tangent space at 1, that is an angular velocity. Thus, it is often involved in the integration and differentiation of rotations.

Coordinate Transforms

Being able to define the pose of a frame and use it to perform operations on motion and force vectors is fundamental for poly-articulated system. To this end, coordinate transforms are introduced. They are called this way because they map spatial vectors computed in a given Cartesian frame to another one. The transformation rule is different according to whether it operates on spatial motion or force vectors. Let A and B be two Cartesian frames. The coordinate transform from A to B coordinates for a motion vector is written ${}^{B}X_{A}$, while the same transform for a force vector is denoted ${}^{B}X_{A}^{*}$ to highlight a kind of duality between them. It follows,

$$\hat{m}_B = {}^{B}X_{A}\ \hat{m}_A, \qquad\qquad \hat{f}_B = {}^{B}X_{A}^{*}\ \hat{f}_A, \tag{2.14}$$

where $\hat{m}_A, \hat{m}_B, \hat{f}_A, \hat{f}_B$ are the coordinate vectors representing the spatial vectors $\hat{m} \in \mathcal{M}^6$ and $\hat{f} \in \mathcal{F}^6$ in Cartesian frames A and B respectively.

The transform ${}^{B}X_{A}$ is very important because it also represents the relative position and orientation of frame A in B coordinates. Suppose that the position and orientation of frame A in B coordinates is described by a position vector ${}^{B}p_{A}$ and a rotation matrix ${}^{B}R_{A}$. The transform ${}^{B}X_{A}$ can be formulated as a 6x6 matrix. It can be easily obtained by decomposing the transformation ${}^{B}X_{A}$ into a pure translation ${}^{B}p_{A}$ followed by a pure rotation ${}^{B}R_{A}$,

$${}^{B}X_{A} = \underbrace{\begin{pmatrix} 1 & {}^{B}p_{A}\times \\ 0 & 1 \end{pmatrix}}_{\text{pure translation}} \underbrace{\begin{pmatrix} {}^{B}R_{A} & 0 \\ 0 & {}^{B}R_{A} \end{pmatrix}}_{\text{pure rotation}} = \begin{pmatrix} {}^{B}R_{A} & {}^{B}p_{A} \times {}^{B}R_{A} \\ 0 & {}^{B}R_{A} \end{pmatrix}. \tag{2.15}$$

The 6x6 matrix associated with the inverse transform ${}^{A}X_{B} = {}^{B}X_{A}^{-1}$ can be computed easily by simply inverting the decomposition. Moreover, from the duality of the spatial motion and force spaces, it comes ${}^{B}X_{A}^{*} = {}^{B}X_{A}^{-T}$. Similarly, the time derivative of a transform is given by,

$${}^{B}\dot{X}_{A} = {}^{B}(v_A - v_B) \times {}^{B}X_{A}. \tag{2.16}$$

Another key property at the heart of rigid body algorithms is the *chain rule*. Let us consider a set of k cartesian frames $\{A^i\}_{i=1}^{k}$, then

$${}^{A^k}X_{A^1} = {}^{A^k}X_{A^{k-1}}\ {}^{A^{k-1}}X_{A^{k-2}}\ \cdots\ {}^{A^2}X_{A^1}. \tag{2.17}$$

Rigid Body Description

Important properties of a rigid body B are its total mass m and the relative position p_G of its *Center of Mass* (CoM) G. It is defined as the unique point where the weighted relative position of the distributed mass over its volume sums to zero,

$$p_G = \frac{1}{m} \iiint_{V_B} \rho(p) p \, \mathrm{d}V, \tag{2.18}$$

where $\rho(p)$ is the local density at point p. If the density is uniform, then it boils down to the centroid of the solid.

Another key property is its inertia I_G expressed at the CoM. It maps the angular velocity ω of a body to its angular momentum h_G. The angular momentum is central in analytical mechanics since it relates the acceleration of a body to the external forces applied to it. It will be presented in more detail in the following. Formally, it is defined as follows,

$$I_G = \iiint_{V_B} \rho(p)(\|p\| I_3 - p \otimes p) \, \mathrm{d}V, \tag{2.19}$$

where $\otimes$ denotes the outer products in $\mathbb{R}^3$ and I_3 is the identity matrix of size 3.

A reference frame attached to the body in which to compute the aforementioned physics properties must be specified. This reference frame must be chosen carefully to minimize coordinate changes when carrying out computations along the kinematic tree and speed up rigid body algorithms subsequently. By convention, it is the frame associated with the parent joint of the body.

The spatial inertia $\hat{I}$ aggregates some of these properties to operate directly on spatial vectors. It is the 6x6 matrix

$$\hat{I} = \begin{pmatrix} I_G & 0 \\ 0 & mI_3 \end{pmatrix} \tag{2.20}$$

mapping a spatial velocity vector $\hat{v} \in \mathcal{M}^6$ to the spatial momentum $\hat{h} \in \mathcal{F}^6$,

$$\hat{h} = \hat{I}\hat{v}. \tag{2.21}$$

From this definition, it stands out that the spatial inertia $\hat{I}$ is a symmetric dyadic tensor, which means that it can be expressed as the sum of six symmetric outer products of vectors $g_i \in \mathcal{F}^6$,

$$\hat{I} = \sum_{i=1}^{6} g_i \otimes g_i. \tag{2.22}$$

From this and knowing that the spatial inertia is a quantity that is fixed in the body frame, its time derivative can be derived from equation (2.9),

$$\dot{\hat{I}} = \hat{v} \times^* \hat{I} - \hat{I} \times \hat{v}. \tag{2.23}$$

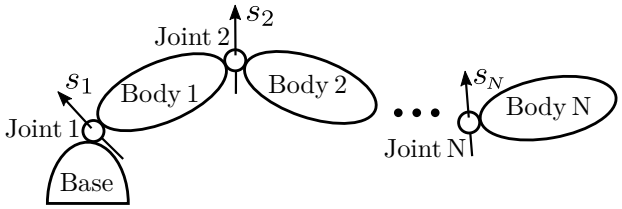


Figure 2.3: Kinematic chain with N bodies. Each joint is characterized by its spatial axis vector s_i assuming a single degree of freedom.

To wrap up, all the physic properties that are needed to compute the dynamics of a rigid body are the relative position of the CoM and the spatial inertia matrix $\hat{I}$ expressed at the CoM. They are estimated by the CAD software used to design the robot. The accuracy is reasonable but far from perfect because many elements are neglected, i.e. cables, electronics, protection foam, glue, drawing simplifications (threading, deburring...), and engineering tolerance (material, assembly...). If more accuracy is needed, typically for spatial applications, then it exists devices to measure these properties directly for rigid sub-assemblies.

Joint Description

A joint constrains the relative motion between its *parent* and *child* bodies in the kinematic tree. A joint is fixed relative to its parent body. Thus, it is sufficient to attach a frame to each of them and specify the relative frame placement with respect to their parent body to carry out computations along the kinematic tree.

By convention, the velocity v_{J_i} of a joint i is defined as the spatial velocity of the child body v_i relative to the parent body v_{i-1}, both expressed in the reference frame of the parent body of the joint. Formally, the joint restricts the velocity v_{J_i} to a subspace $\mathcal{S}_i \subseteq \mathcal{M}^6$ at the current time. If the joint allows n_f DoFs, then $dim(\mathcal{S}) = n_f$ and $S(q)$ is a $6 \times n_f$ matrix. Similarly, the number of constraints enforced by the joint is $n_c = 6 - n_f$ and the constraint internal forces lie in the n_c-dimensional orthogonal subspace $\mathcal{S}^\perp \in \mathcal{F}^6$. Indeed, joints do not generate nor consume power since friction and actuation are handled separately, therefore the scalar product between motion and force vectors must be zero, which is exactly the orthogonality condition.

The joint constraint is most often *scleronomic*, i.e. it can be written as equality only involving positions and velocities without explicit time dependency. It yields,

$$v_{J_i} = S(q_i)\dot{q}_i, \tag{2.24}$$

where q_i is the subset of the generalized coordinates associated with the joint i, and $S(q_i)$ is a matrix that may depend on the joint configuration. The matrix $S(q_i)$ is specific to each type of joint. It is usually invariable, one notable exception being the universal joint or Cardan joint. The latter is a compound joint of two revolute joints whose axes intersect orthogonally. For joints allowing a single DoF such as prismatic but also helicoidal joint, $S(q_i)$ is a constant motion vector $S_i \in \mathcal{M}^6$ and

${}^0X_{i-1}S_i$ is the spatial axis vector in base frame for the current configuration $s_i(q)$ (see figure 2.3). It is always possible to model a complex joint as a serial chain of 1-DoF coincident joints separated by massless bodies. Such a compound joint is said to be kinematically equivalent.

Let ${}^jv_i$ be the spatial velocity of a body i in the frame of a body j and ${}^jv_{J_i}$ be the spatial velocity of a joint J_i in the frame of a body j. The spatial velocity ${}^0v_i$ of a body i in the base frame and its Jacobian ${}^0J_i$ are computed using the chain rule:

$$\begin{aligned} {}^0v_i &= {}^0X_{i-1}\,{}^{i-1}v_i = {}^0X_{i-1}\left(S(q_i)\dot{q}_i + {}^{i-1}v_{i-1}\right) = {}^0X_{i-1}S(q_i)\dot{q}_i + {}^0v_{i-1} \\ &= \cdots = \sum_{j=1}^{i} {}^0X_{j-1}S(q_j)\dot{q}_j = \left[{}^0X_{i-1}S(q_i), {}^0X_{i-2}S(q_{i-1}), \cdots, S(q_1)\right]^T \dot{q} \\ &= {}^0J_i\dot{q}. \end{aligned} \tag{2.25}$$

2.1.2 Whole-Body Dynamics

The dynamic equation of a poly-articulated robot can either be obtained using *Newton-Euler formulation* and *Lagrange formulation*. The algorithmic complexity of a rigid body algorithm is directly related to the formulation that is used. One or the other will be more appropriate depending on the kinematic structure of the robot and the quantity to compute. Therefore, it is essential to present both. Newton-Euler is more intuitive and well-known but Lagrange formulation is more commonly used in robotics to describe the dynamics of a system.

For a single rigid body, the dynamic equation of motion is straightforward to obtain from the Newton-Euler formulation. The rate of change of the spatial angular momentum $\hat{h}$ equals the total spatial force acting on it,

$$f = \dot{\hat{h}} = \hat{I}\hat{a} + \dot{\hat{I}}\hat{v} = \hat{I}\hat{a} + \hat{v} \times^* \hat{h}, \tag{2.26}$$

where $f \in \mathcal{F}^6$ is a sum of the spatial forces applied on the body, $\hat{I}$ is its spatial inertia, and $\hat{v}, \hat{a} \in \mathcal{M}^6$ are its spatial velocity and acceleration respectively. One can obtain the dynamic equation of the whole system by simply stacking the ones for each body individually. At this point, f comprises the actual external forces to the whole system such as the ground reaction forces and motor torques, but also the internal forces consequent to kinematic constraints and the effect of gravity. It is possible to get rid of those internal forces by taking into account those constraints explicitly and solving them using the Lagrangian multiplier method. Let T_i be the matrix that spans $S_i^{\perp}$, where S_i the motion subspace for joint i,

$$T_i^T v_{J_i} = 0. \tag{2.27}$$

By differentiating this relation, concatenating it for each joint, and using the duality between motion and force vectors it yields,

$$T^T a_J + \dot{T}^T v_J = 0, \qquad f = \tau + f_g + T\lambda, \tag{2.28}$$

where T is a block diagonal matrix having T_i in its j-th diagonal block, τ is the sum of the actual external forces and λ is the vector of internal forces resulting from the joint constraints. One can compute the internal forces λ explicitly by jointly solving equations (2.26) and (2.28).

The Lagrange formulation proceeds via the Lagrangian of the system,

$$L = T - U, \tag{2.29}$$

where T, U are the total kinetic and potential energy respectively. The total kinetic energy is the sum of the kinematic energy of the n individual bodies,

$$T = \sum_{i=1}^{n} \frac{1}{2} \hat{v} \cdot \hat{h} = \sum_{i=1}^{n} \frac{1}{2} \hat{v} \hat{I} \hat{v}. \tag{2.30}$$

This expression can be written in matrix form using generalized coordinates based on equation (2.25),

$$T = \frac{1}{2} \dot{q}^T H(q) \dot{q} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \dot{q}_i^T H_{i,j} \, \dot{q}_j, \tag{2.31}$$

where I_i^c is the aggregated inertia of the subtree rooted at body i treated as a single composite rigid body and $H_{i,j} = S_i^T I^c_{\max(i,j)} S_j$. $H(q)$ is called *inertia matrix* or *mass matrix* and only depends on the configuration q of the system.

The dynamic equations of motion can then be developed using Lagrange's equation of the first kind,

$$\frac{\mathrm{d}}{\mathrm{d}t} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q} \cdot \left(\frac{\partial g}{\partial p_i} (g^{-1}(q)) \right) = \tau_i \tag{2.32}$$

where τ are the generalized external forces, $p_i = \int \dot{q}_i$ is the element-wise primitive of the generalized velocity locally and $g \in \mathcal{C}^1$ is the mapping from the local chart p to the generalized coordinates q. If the generalized coordinates are all independent, which is usually the case except for the spherical joint associated with the freeflyer, then g is identity. The canonical matrix form of the dynamics of poly-articulated robots is obtained by replacing equation (2.31) in equation (2.32),

$$H(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau = Bu + \sum_i J_i^T(q) f_i, \tag{2.33}$$

where H, C, G are the joint-space inertia, Coriolis, and gravity matrices respectively, B is a selection matrix that determines how the controls u take effect on the joints, $f_i \in \mathcal{F}^6$ is the i-th external force applied on body B_i and J_i is the jacobian of B_i.

This general form is important to infer mathematical properties about the system but is never computed explicitly in practice. Recursive algorithms going back and forth through the subchains of the kinematic tree are used instead because they have

much lower algorithmic complexity. The main algorithms are: *Recursive Newton-Euler Algorithm* (RNEA) for the *Inverse Dynamics* (ID), *Articulated Body Algorithm* (ABA) for the forward dynamics, and *Composite Rigid Body Algorithm* (CRBA) for calculating the joint-space inertia matrix (Featherstone, 2008). *Forward Dynamics* (FD) refers to the calculation of the acceleration response of a given rigid-body system to a given applied force $\ddot{q} = \text{FD}(\text{model}, q, \dot{q}, \tau)$, and the inverse dynamics refers to the calculation of the force that must be applied to a given rigid-body system in order to produce a given acceleration response $\tau = \text{ID}(\text{model}, q, \dot{q}, \ddot{q})$.

It must be distinguished from the *Forward Kinematics* (FK), which consists of computing the pose, spatial velocity and spatial acceleration of the joints based on the generalized position and its time derivatives $\hat{p}_J, \hat{v}_J, \hat{a}_J = \text{FK}(\text{model}, q, \dot{q}, \dot{q})$. This relation is said to be of n-th order according to the highest order derivative involved. Any spatial feature up to the same order can be inferred from the joint information by linear transformation (cf. section 2.1.1), e.g. the position of the CoM or the angular velocity of *Inertial Measurement Unit* (IMU) sensors.

Broadly speaking, *Inverse Kinematics* (IK) is the reverse operation. It is more challenging because the FK is most often not invertible due to over-constrained kinematics. It is formulated as an optimization problem that is solved via an iterative gradient descent method. At this point, the spatial features of interest are considered in place of the joints, eventually with different priority levels for weighting their respective contributions hierarchically. The problem is said to be *whole-body* if it operates on the generalized coordinates and derivatives as a whole instead of each motor individually. The problem can be written as a *Quadratic Program* (QP) with inequality constraints, which can be solved efficiently with arbitrary precision. In theory, it has exponential time complexity, but it is closer to polynomial time in practice. Still, it is much larger than constant time for recursive algorithms.

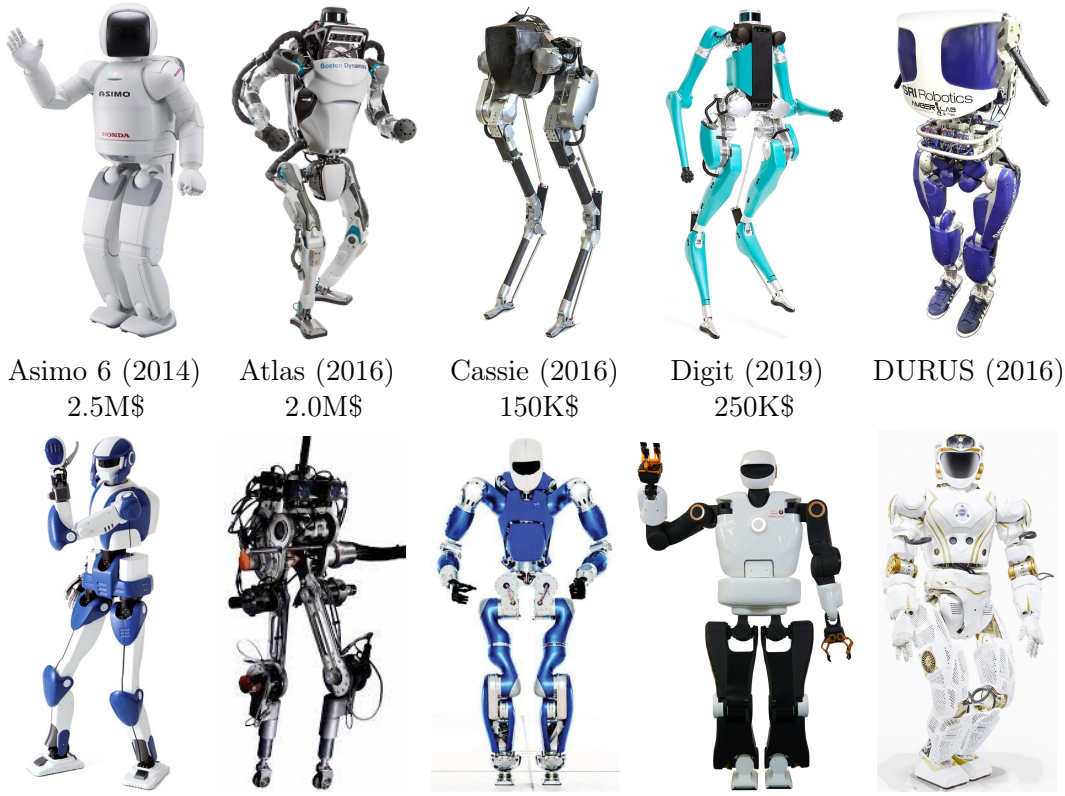
2.2 Planning and Control in Bipedal Robotics

2.2.1 Notion of Bipedal Locomotion and Terminology

Description of Bipedal Robots

Bipedal robots fall into the category of rigid poly-articulated systems. A bipedal robot is a kinematic tree comprising two sub-chains called *legs* and another one called *torso*, all connected at a common body called *pelvis*. The torso may have two additional sub-chains for *arms*. In that case, it is labeled as a humanoid robot. Cassie robot by Agility Robotics is an example of a bipedal robot that is not humanoid. Their other product Digit – basically Cassie with the addition of an upper body – is said to be humanoid, even though it looks like an ostrich (see figure 2.4). It has a very short femur and foot-like toes, which gives the impression that the knee is bending backward while in fact, it is its ankle.

A robot is *fully-actuated* if one can command an arbitrary instantaneous acceleration for any state of interest. This assumption holds for any bipedal system with



Asimo 6 (2014) 2.5M$ | Atlas (2016) 2.0M$ | Cassie (2016) 150K$ | Digit (2019) 250K$ | DURUS (2016)

HRP-4 (2010) 325K$ | RABBIT (2003) | TORO (2013) | Talos (2017) 1.0M$ | Valkyrie (2016) 2.0M$

Figure 2.4: Examples of bipedal robots (not at scale) with price tags if available.

human-like feet and all joints actuated, as long as at least one foot is in contact with the ground where it remains flat and does not slip. Being fully-actuated enables standing still indefinitely without falling, otherwise stepping in place would be necessary. It is usually expected from a bipedal robot but not necessary. RABBIT is a biped that is *under-actuated* since it has point feet. In such a case, most of the classical theory about the stability analysis of legged robots is not applicable. The concept of *Hybrid Zero Dynamics* (HZD) has been developed to overcome this limitation and enable both planning and control on this kind of platform (Finet, 2018).

Most legged robots only feature revolute joints (Goswami & Vadakkepat, 2019, Part II). However, it can get more complex if the transmissions between the actual actuators and the mechanical joints are nonlinear, e.g. the ankles of Atalante and Cassie (cf. appendix C.3). The mechanical structure is supposed to be rigid, or more precisely, only flexible at specific locations, typically the joints (cf. appendix C.1).

Humanoid robots have to perform reactive motions to keep balance, which requires actuators that can generate high torques and high speeds at the same time. Atlas robot uses hydraulic actuators powered by an electric pump. It delivers high

torques with high bandwidth, but it creates loud sounds. Moreover, hydraulic systems need security features to avoid safety issues due to high pressure, which are both difficult to implement and certify. With the advent of electric motors in many industries during the last decade, the technologies behind it have undergone several breakthroughs (Goswami & Vadakkepat, 2019, Part III). Electric motors can deliver much higher torque than they used to. *Strain wave gearing*, also called Harmonic Drive, are very compact mechanical gears with high reduction ratios that became increasingly popular. Coupled together with this kind of motor, it can deliver high mechanical power at the appropriate speed for a small volume and weight. Regarding the mechanical design, new lighter materials are now available, e.g. carbon fibers. All these technologies make it possible to design robots capable of very agile and dynamic motions such as jumping or flipping. Atalante relies on them like many others.

Some legged robots including Valkyrie use *Serial Elastic Actuators* (SEAs). It is a special actuator block that introduces physical compliance by inserting an elastic element between the motor and the load to store and release part of the mechanical energy. They serve mainly three purposes: filtering external forces, regulating applied forces, and measuring the torques at the joint level. More precisely, they reduce the magnitude of force impulses and spreads them out over time. In principle, it enables completely canceling them out through control, which would be impossible otherwise. First, the reflected inertia of geared motors limits the response time of the transmissions even in an ideal world. Secondly, the real hardware stack has even more limited bandwidth, the surplus energy would be dissipated mainly by overheating, and the motors have maximum torques. The downside is making the control significantly more complex as the dynamics of the actuators must be taken into account (Paine et al., 2015). Torque-controlled robots like DURUS (Hereid et al., 2018) are very promising. Grasping a glass requires accurate planning based on its exact shape to avoid breaking it if position control is used without compliance, whereas knowing its exact shape is not even necessary when controlling the force applied to it. Thus, torque control not only enables safer interaction with the world but also alleviates the need of planning ahead actions precisely.

Position control remains relevant to accurately reproduce some nominal trajectory. It has demonstrated its robustness over the years in many applications. For this reason, Talos allows for both position and torque control (Stasse et al., 2017).

Bipedal Locomotion

Formally, bipedal locomotion consists in translating the center of mass or rotating the principal axes of inertia by moving the legs. During locomotion, the altitude of the center of mass is always above a certain level which is used to characterize falling. In particular, the motion of the center of mass is almost sinusoidal for humans during walking (Kuo, 2007). Human locomotion has been heavily studied for more than one century, but its underlying mechanisms are still misunderstood. Many works agree to say that human walking results in the optimal motion of the center of mass (Charalambous, 2014; Kuo, 2007). Collins et al. (2009) claim that the motion of

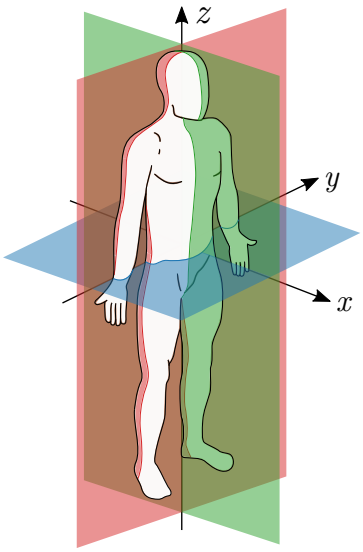


Figure 2.5: Human planes of section. The sagittal plane divides the body into right and left sections, the frontal plane splits the body into front and back portions, and the transverse plane separates the upper body from the lower body.

arms plays a role in the reduction of energetic consumption and the risk of slippage. Yet, it is not clear which cost function human walking minimizes and how it ensures robustness and emergency recovery (Bretl et al., 2010).

Human locomotion on flat ground is a cyclic motion. It can be divided into two phases: the single support phase where only one foot is in contact with the ground, and the double support phase where both feet are in contact. A leg is said to be in *stance phase* when it is in contact with the ground, in *swing phase* otherwise (see figure 2.6). The tip of a leg is called *foot*, no matter if it constitutes an actual foot. During the stance phase, the *flying foot* first impacts the ground with the heel. The foot rotates about the heel. Next, the foot lays flat on the ground. Then, the heel lifts from the ground and the foot rotates about the toe. The stance phase finishes when the toe lifts from the ground. In the particular case of walking, when the speed increases, the duration of the double support phase diminishes until it disappears altogether. It is called running, and the double support phase is replaced by the *flight phase*. The human planes of sections are commonly used in robotics and the medical field to describe locomotion (see figure 2.5).

Only a small set of primitive periodic motions is necessary to enable versatile locomotion on flat ground (see figure 2.7). These primitive motions can be combined to move in complex environments and adapt the pace if necessary. It is natural for humans, but the underlying mechanism is still unclear. It is not enough to combine them linearly or to change the speed by time dilation, otherwise, the resulting motion is not guaranteed to be stable.

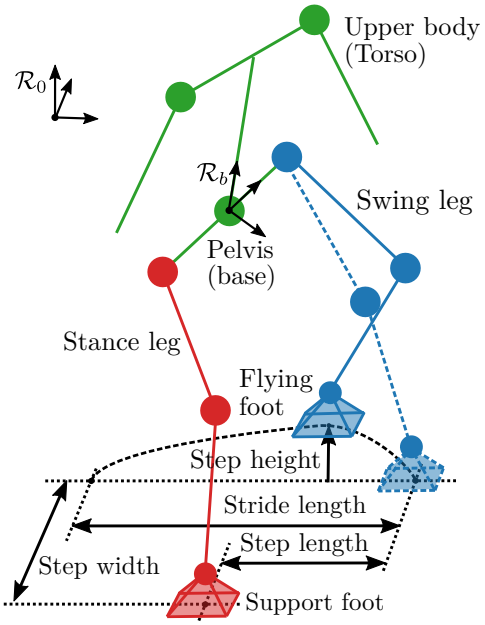


Figure 2.6: Characterization of bipedal locomotion

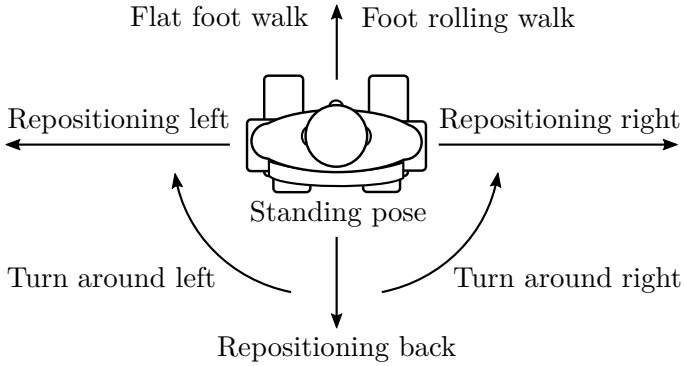


Figure 2.7: Motion primitives for versatile locomotion on flat ground

Modelling of the System Patient-Exoskeleton

We assume that each link of the user is rigidly fastened to the exoskeleton. In this regard, the system exoskeleton-patient can be viewed as a humanoid robot after aggregating their respective mass distributions. Although this simplifying assumption is rather realistic for the lower body, it is questionable for the upper body. Indeed, the fastening of the hip and torso to the exoskeleton is slack, and the arms are completely free. Nevertheless, trying to simulate the coupling between the patient and the exoskeleton is unlikely to bring any improvement for several reasons,

- the straps are made of deformable composite materials and doing a proper identification of their physical properties would be very difficult,
- the coupling is never the same as the way users are fastened is not repeatable,

- simulating the dynamics of the robot and the exoskeleton separately and interacting through coupling forces at several locations is computationally intensive.

In reality, paraplegic people apply involuntary resistive forces on their own joints as a result of muscle spasticity – a condition in which muscles stiffen or tighten – and spams. These forces would be regarded as disturbances from the perspective of the robot, and as such, should be taken into consideration. However, simulating these effects requires a muscle model, which involves unknown patient-specific parameters. Besides, patients may also apply forces intentionally or by reflex through the motion of their upper body. These behaviors are highly subjective and not directly related to a given pathology. How to model them is an open question.

Under the previous assumption of rigid coupling between the user and the exoskeleton, the mathematical expression of the dynamics of the system is the same as any other poly-articulated robot. Therefore, most of the theoretical background developed for bipedal robots can be translated nearly effortlessly (Goswami & Vadakkepat, 2019). Still, some dynamic properties that would be constant for classical robots are not for Atalante. Especially, the morphology of the user modifies the mass distribution of the robot and the length of its links featuring dimensional adjustments. The planning and control algorithms must be adapted to handle such variability.

2.2.2 Stability Assessment

Mammals keep balance while moving seemingly effortlessly. Yet, reproducing this behavior on legged robots is a challenging task for both planning and control. This problem is already quite well understood for quadrupedal robots, but much remains to be done for bipedal robots. Indeed, it gets harder as the number of legs decreases. First and foremost, it is essential to come up with some criteria to assess whether the robot is falling. There are two types of local stability: static and dynamic. A trajectory is said to be statically stable if the robot could stop and hold in place at any point in time without falling. Although statically stable humanoid locomotion is generally possible, it often looks unnatural as it is slow-paced and lacks efficiency. It also limits the set of motions that can be performed, which is even more detrimental and often prohibitive. For all these reasons, it is no longer an active research topic. The notion of dynamic stability is more interesting but much harder to define rigorously. Roughly speaking, it relates to the ability to keep moving without falling. Typically, human walking can be viewed as constantly falling forward, slowing the fall by landing the flying foot at the right location and starting again on the opposite leg. It is not statically stable as it would be impossible to stay in place in the middle of a step. More generally, mammal locomotion is a classic example of dynamically stable motion that is not statically stable. The classical criteria involved in the assessment of dynamic stability are reviewed in this section. More details can be found in *Humanoid robotics: a reference*, Part VI or Boer (2012, Chapter 7).

In the particular case of periodic locomotion for legged robots, at least two different notions of stability can be defined: a global one through the analysis of the convergence to a limit cycle, and a local one only interested in what is happening

at the current time. Global stability is what matters in practice. However, global stability criteria are very challenging to define and evaluate. The Poincaré map relates the attractiveness of a limit cycle over successive steps to the stability analysis of a fixed point. It is mathematically powerful but is limited to the theoretical analysis of the stability of periodic nominal motions. Thus, it may be helpful in planning but has no value in control. The Lyapunov stability theory is comparatively more versatile, but how to apply this approach to complex systems such as legged robots is still an active research topic, as discussed in section 4.2.3. Thus, we set aside the question of global stability in the following to focus exclusively on its local counterpart. Local stability covers mainly *contact stability* and *capturability*. The former is about preventing the robot from slipping or losing contact with the ground, while the latter refers to the ability to stop the motion completely in any number of steps. Guaranteeing contact stability is essential in planning (cf. appendix C.3.2), while capturability is prematurely useful in control. These two aspects have been widely studied for generating motion on legged robots, with very impressive results in simulation (Caron et al., 2020; Caron et al., 2017). Indeed, motions involving slipping on purpose such as sharp turns heavily rely on the friction model and hence would hardly work in practice. Consequently, the contact sequence is systematically planned so as to avoid it, with some safety margin to provide leeway and mitigate model uncertainties. Then, those local stability criteria are usually involved in the feedback loop to actively compensate for unexpected disturbances.

Classic planning and control methods have been outperformed recently by end-to-end policy learning approaches (Castillo et al., 2021; Li et al., 2021). The learning agent usually discovers all by itself its very own stability criterion. Still, it remains beneficial to guide the agent by providing reward components from classical stability metrics because it tends to improve the robustness of the convergence, lead to more human-like behaviors, and ease sim-to-real transfer. Beyond this, black box end-to-end approaches raise interpretability concerns. Having in mind the various classical stability criteria is very helpful to analyze results both in simulation and in reality.

Center of Pressure

Many attempts were made in the past to find a good stability criterion for legged locomotion. We limit ourselves to the analysis of the contact stability here. Consequently, it is assumed that there is at least one contact point with the ground the whole time. It is definitely restrictive, as it excludes running or jumping for instance, but it is nonetheless sufficient for natural locomotion.

Until the last decade, the *Center of Pressure* (CoP) was ubiquitous in the literature. It provides an intuitive and easy-to-compute necessary condition for contact stability but is only defined for coplanar contacts on flat ground. The CoP is the point C in the ground plane where the momentum of the resultant force exerted by the pressure field equals zero (Vukobratović & Stepanenko, 1972). It yields,

$$p_C = \frac{\tau_{n,c}}{f_{n,c}}, \tag{2.34}$$

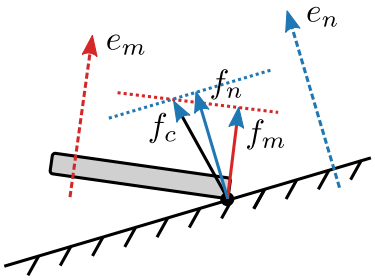


Figure 2.8: Pressure force estimation. The sensor axis is not aligned with the ground normal. The relative angle between them could be observed by placing an additional IMU sensor in the foot but is usually neglected, leading to measurement error.

where e_n denotes the normal axis of the ground, $f_{n,c}, \tau_{n,c}$ are the force and momentum resulting from the pressure forces (Caron et al., 2017; Sardain & Bessonnet, 2004). If the ground interaction consists of m contact points $\{p_i\}_{i=1}^m$, then $\tau_{n,c} = \sum_{i=1}^m f_{i,n}(p_i \times e_n)$ and $f_{n,c} = \sum_{i=1}^m f_{i,n}$.

Equivalently, the CoP can be reformulated as the barycenter of the pressure forces,

$$p_C = \frac{1}{f_{n,c}} \sum_{i=1}^{m} f_{i,n}\, p_i. \tag{2.35}$$

As a property of the barycenter, the CoP is always in the closed convex hull of the contact points, the so-called *support polygon*.

It follows from these definitions that the CoP must be an interior point of the support polygon for the contact to be stable. This geometric condition for contact stability is necessary: if the CoP is on the boundary of the support polygon, then the robot will start tilting around the corresponding edge and lose contact. Still, it is not sufficient because the robot might well slip if the friction with the ground is not large enough no matter the CoP and the support polygon. The distance from the CoP to the boundary of the support polygon is nonetheless an informative metric of stability, also called stability margin. It is a numerical indicator of the risk of tipping-over since the robot may withstand strong disturbances without breaking contact stability if the CoP is further away from the boundary.

Having a stability metric that is independent of the friction forces is beneficial for control as it can be estimated from the pressure forces alone. The latter are usually measured contact sensors only capable of measuring the force along their own vertical axis, ignoring all the other components. They are almost unbreakable, while extremely cheap and reliable. However, their vertical axis only matches the true normal of the ground when the foot on which they are attached is flat on the ground (see figure 2.8). It is not always the case, notably during the double support phase in foot rolling (see figure A.1). This discrepancy affects the accuracy of the CoP estimate. Furthermore, the CoP estimate is confined to the convex hull of the contact sensors. The latter are never placed at the actual vertices of the feet but further inside their footprint due to mechanical design considerations (see figure 2.9). Thus, the

CoP estimate would saturate before ridging an edge of the support polygon. In theory, sensor fusion and model-based approaches could compensate for such biases while being robust to intrinsic measurement noise and bias. Yet, this is very challenging in practice, mainly because of modelling errors. It is also possible to add extra sensors or switch with more expensive technologies, e.g. relying on a 6-axis force sensor at each end-effector like Talos, but it would substantially increase the production and maintenance costs. Using glsrl methods for training observers or end-to-end control policies directly is a promising research direction to overcome issues related to limited sensor capabilities and partial state observability.

Zero-tilting Moment Point

Being able to walk on uneven ground or use hands to keep balance by pushing on surfaces is important. However, this is out of the scope of the original contact stability criteria based on the CoP, which is only applicable to coplanar contact points. Moreover, taking into account is helpful to avoid planning feasible motions. New contact stability criteria emerged during the last decade to overcome these limitations, but they lack an intuitive geometric interpretation and do not provide a stability metric for robustness analysis (Caron et al., 2015; Hirukawa et al., 2006). More recently, Caron et al. (2017) managed to generalize the original contact stability criteria based.

Let us define the *gravito-inertial wrench* $\hat{f}^{gi}$ and the *contact wrench* $\hat{f}^{c}$ as follows,

$$\hat{f}^{gi} = \begin{pmatrix} f^g - \dot{P} \\ p_G \times (f^g - \dot{P}) - \dot{L}_G \end{pmatrix}, \qquad \hat{f}^{c} = \begin{pmatrix} f^c \\ \tau^c_O \end{pmatrix} = \sum_i \begin{pmatrix} f^{c,i} \\ p_{C_i} \times f^{c,i} \end{pmatrix}, \tag{2.36}$$

where $\hat{h} = (P, L_G)$ is the spatial momentum of the system at its CoM G in world frame, f_G denotes the gravity force, and f_i is the contact force exerted by the environment on the robot at the i-th contact point C_i in world frame (Caron et al., 2017). Notably, the force at a given contact point i can be decomposed in pressure $f_{i,n}$ and friction $f_{i,t}$ components are that respectively normal and tangential to the ground plane (cf. appendix C.2.2).

While the contact wrench only requires going through all the bodies in contact to be computed, the gravito-inertial wrench aggregates the individual gravity and inertial forces induced by all the bodies in the kinematic tree. The spatial momentum $\hat{h} = (P, L_G)$ of the robot taken at its CoM G in world frame is given by,

$$P = \sum_{\text{body } k} m_k \dot{p}_{G_k}, \qquad L_G = \sum_{\text{body } k} \Big\{ m_k (p_{G_k} - p_G) \times \dot{p}_{G_k} + R_k I_k \omega_k \Big\}, \tag{2.37}$$

where, m_k the mass of a given body k, p_{G_k} the absolute position of its CoM G_k, R_k its orientation matrix in world frame, ω_k its angular velocity in local frame, and I_k its inertia matrix expressed at its CoM in local frame.

The gravito-inertial wrench augmented with the external forces and the contact wrench are related via the dynamic equations of motion (Sardain & Bessonnet, 2004),

$$\hat{f}^{gi} + \hat{f}_{ext} = -\hat{f}^{c}, \tag{2.38}$$

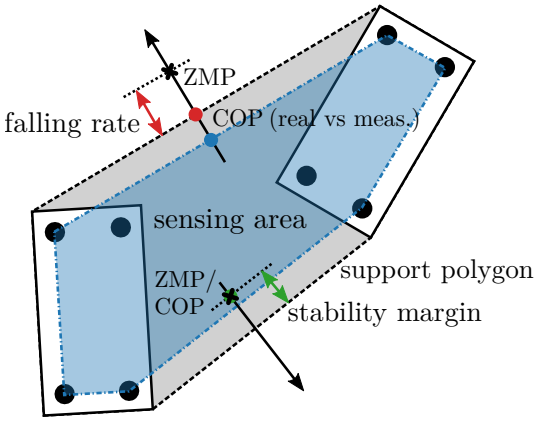


Figure 2.9: Dynamic stability assessment on flat ground using the ZMP. If the ZMP is within the support polygon, then the stability margin corresponds to the distance from the closest edge. The larger the margin the stronger the force it can withstand without having to take action to avoid falling.

where $\hat{f}^{ext}$ is the resultant wrench exerted by all external forces except the ground reaction. These include the controls through the transmissions.

The *Zero-tilting Moment Point* (ZMP) is defined as the point Z in a given plane where the tilting momentum acting on the system due to augmented gravito-inertial wrench equals zero. The tilting momentum is the component that is tangential to the supporting surface, hence the normal component of the momentum is not zero in general. Namely, $Z \in \Pi(O, e_n)$ s.t. $e_n \times (\tau_Z^{gi} + \tau_Z^{ext}) = 0$, where $\Pi(O, e_n)$ is the virtual plane $\Pi(O, e_n)$ with origin O and normal axis e_n. It yields,

$$p_Z^{\Pi} = \frac{e_n \times \tau_O^c}{e_n \cdot f^c}. \tag{2.39}$$

This expression is well-posed mathematically since $e_n \cdot f^c$ is about equal to the weight of the robot during walking. It is easy to check that the absolute position of the ZMP is independent of the origin of the virtual plane $\Pi(O, e_n)$, which is expected. Note that the friction forces have an impact on the ZMP for contact points not contained in the virtual plane.

According to equation (2.38), the augmented gravito-inertial wrench has been replaced by the contact wrench in equation (2.39). Indeed, the gravito-inertial wrench involves the full state of the system and its time derivative, which cannot be reliably estimated on the real robot. Moreover, some external forces may not be unobservable, typically the effort of the physiotherapist on the handles in the particular case of a medical exoskeleton. On the contrary, the contact wrench can be measured almost directly by placing 6-axis force sensors in the end-effectors. Its computation implicitly involves the relative position of each end-effector with respect to the CoM of the robot. The latter can be estimated fairly accurately using a state observer taking into account the mechanical deformation Vigne et al. (2020b).

On flat ground, the virtual plane $\Pi(O, e_n)$ can be chosen to be the ground itself. In that case, the ZMP matches exactly the CoP no matter what (Sardain & Bessonnet, 2004). Any virtual plane would be equally acceptable, so that the ZMP can be any point on the *zero moment axis*, which is defined as the line between the CoM and the CoP. Thus, ZMP can be seen as an extension of the CoP to uneven ground.

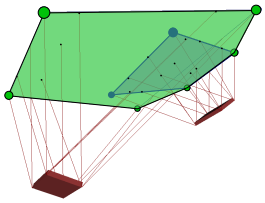
It is a common misbelief that the ZMP is not confined to the support polygon and differ from the CoP as soon as the robot starts rotating about an edge of the support polygon. Such a discrepancy would imply that the dynamic equation of motion is not verified, which is obviously impossible for the real robot. Still, this may happen during motion planning if consistency is not strictly enforced. If the ZMP derived from the augmented gravito-inertial wrench is outside the support polygon during planning, then the generated motion would not be feasible (not dynamically stable) and hence impossible to realize experimentally: the contact will break and the robot will fall if the controller keeps tracking the nominal trajectory.

Caron and Kheddar (2017) refined the original definition of the support area to take into account the friction for coplanar contacts. The actual support area $\mathcal{Z}$ is the intersection between the convex hull of the contact points $\mathcal{S}$ and the friction cone $\mathcal{C}$ rooted backward at the CoM,

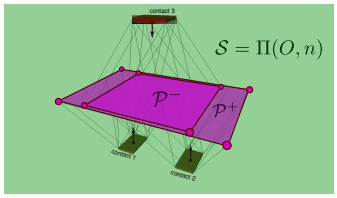
$$\mathcal{Z} = \mathcal{S} \cap (p_G - \mathcal{C}). \tag{2.40}$$

Having the ZMP strictly within the support area $\mathcal{Z}$ is more restrictive than originally but still does not provide a sufficient condition for contact stability. It only states that there exists at least one instantaneous acceleration for which the contact is stable. This acceleration is only guaranteed to be realizable if the system is fully-actuated and subsequently unbounded controls (see definition in section 2.2.1). Indeed, the true support area is a subset of $\mathcal{Z}$. Nevertheless, this simplifying assumption is much weaker than infinite friction and usually not limiting.

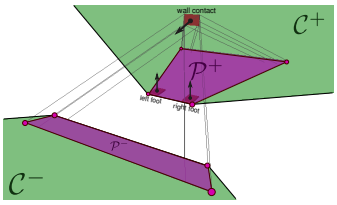
This enhanced definition of the support area is still limited to coplanar contact points. Generalizing it to uneven ground is significantly more challenging than taking into account friction. First, the friction cone associated with each contact point is projected in this virtual plane. The pressure can be either positive or negative depending on the orientation of contact normals relative to the plan normal. Next, the convex hull corresponding to both positive and negative pressure is extracted separately. It will result in two separated polygons that can each be empty and are denoted $\mathcal{P}^+, \mathcal{P}^-$ respectively. Assuming there is at least one contact point, there are three different cases at this point figure 2.10: one of the convex hulls is empty, they intersect, or they are disjoint. If one of them is empty, then the (virtual) support area $\mathcal{S}$ is a polygon and corresponds to the single non-degenerated convex hull. It is the most common scenario, for instance walking on a constant slope or on stairs without using hands to push on walls. It would also be the case if the ground profile is not too steep and the friction coefficient is bounded. The maximum slope must not exceed $\pi/2 - \arctan(1/\alpha)$; where α is the friction coefficient. It corresponds to about 60 degrees for typical friction $\alpha = 0.5$. If the two polygons are non-empty and



(a) Either $\mathcal{P}^+$ or $\mathcal{P}^-$ is empty. $\mathcal{S}$ is the only non-empty polygon.



(b) The intersection of $\mathcal{P}^+$ and $\mathcal{P}^-$ is not empty. $\mathcal{S}$ is the whole virtual plan.



(c) The intersection of $\mathcal{P}^+$ and $\mathcal{P}^-$ is empty. $\mathcal{S}$ is the reunion of $\mathcal{C}^+$ and $\mathcal{C}^-$.

Figure 2.10: Virtual Support area in the three cases. $\mathcal{P}^+, \mathcal{P}^-$ denote the positive and negative pressure polygons, and $\mathcal{S}$ is the virtual support area. (Caron et al., 2017)

intersect, then the contact forces can generate any resultant wrench, which means that the ZMP can be anywhere while maintaining dynamic stability. For instance, it will happen if the robot is pushing on the ceiling in a room with flat ground. Finally, if the two polygons are non-empty and disjoint, then it gets more complicated. The support area $\mathcal{S}$ is the reunion of two polygonal cones $\mathcal{C}^+, \mathcal{C}^-$ extending the two convex hulls $\mathcal{P}^+, \mathcal{P}^-$ respectively,

$$\mathcal{C}^+ = \left\{ p_{Z^+} + \lambda \overrightarrow{Z^- Z^+}, \lambda \geq 0, Z^\pm \in \mathcal{P}^\pm \right\} \tag{2.41}$$

$$\mathcal{C}^- = \left\{ p_{Z^-} + \lambda \overrightarrow{Z^+ Z^-}, \lambda \geq 0, Z^\pm \in \mathcal{P}^\pm \right\}. \tag{2.42}$$

Capture Point

How to assess the dynamic stability of legged robots was put aside for now. Intuitively, it requires some kind of forecasting capability, which is out of reach of the previous criteria that are all about instantaneous contact stability. Pratt et al. (2006) has introduced the concept of *Capture Point* to address this question, as intermediate quantities to maximize the basin of attraction around a nominal trajectory. It is assumed in the following that contact with the environment is limited to the ground. It holds true for bipedal locomotion on uneven ground as long as pushing on walls using the hands is prohibited.

The Capture Point is a point on the ground where the robot must step to bring itself to a complete stop (Hirukawa et al., 2006). It can be interpreted as an anticipation of the future as viewed from the current point in time. Moving the flying foot to the Capture Point is enough to maintain balance indefinitely. Yet, it is essential to make sure it is always reachable, eventually in several steps if it is too far away to be reached in a single step. The Capture Point is closely related to the notion of N-step capturability, which is thoroughly studied in "Foot placement in robotic bipedal locomotion", Chapter 3. If the Capture Point is reachable, then the robot is 1-step capturable. By extension, 0-step capturability consists in keeping balance without moving the feet by only changing the posture in double support.

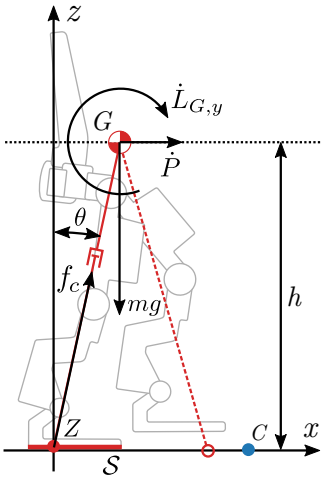


Figure 2.11: Centroidal Dynamics and Inverted Pendulum Model. $\mathcal{S}$ denotes the support polygon, which is the footprint of the stance leg here. Z is the ZMP while C is the DCM. The height h may be variable depending on the model. The angular momentum along y-axis $\dot{L}_{G,y}$ is usually assumed to be zero but not necessarily.

Fast computation of the Capture Point is not possible for complex systems such as legged robots. It does not have any closed-form solution in the general case, and the solution may not be unique. To circumvent this limitation, Kajita et al. (2001) suggested using an approximate model called *Linear Inverted Pendulum Model* (LIPM) for bipedal robots. In this model, the height of the CoM is constant and there is no angular momentum around the CoM. The first assumption implies that the robot is walking with bent knees. The second one entails that the upper body must be always straight with locked arms if any, and the inertia of each leg is negligible.

The relation between the temporal evolution of the CoM and the ZMP can be derived easily for the LIPM,

$$\ddot{p}_G = \omega_0^2(p_G - p_Z), \tag{2.43}$$

where $\omega_0 = \sqrt{g/h}$ is called *natural frequency*, g is the gravity constant and $h = z_G - z_Z$ is the height of the CoM. Many less restrictive variants of the original inverted pendulum model have emerged to allow more natural and versatile locomotion. In particular, Caron (2020) has introduced the *Variable-Height Inverted Pendulum* (VHIP) to enable climbing stairs. The natural frequency ω is no longer constant but satisfies a *Riccati equation*. Regardless of the specific model, the second-order dynamics can always be decoupled into two first-order nonlinear systems. It follows

$$\xi = p_G + \frac{\dot{p}_G}{\omega} + \frac{g}{\omega^2}, \tag{2.44}$$

$$\dot{\xi} = \omega(\xi - p_Z) + \frac{g}{\omega}, \tag{2.45}$$

$$\dot{p_G} = \omega(\xi - p_G), \tag{2.46}$$

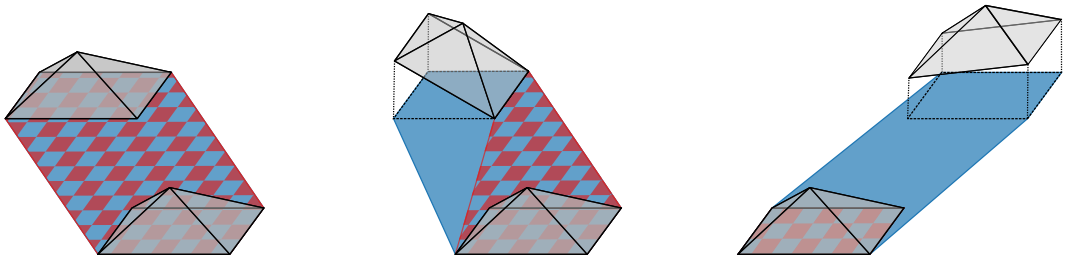


Figure 2.12: Comparison between true and projected support polygon on 3 different configurations of the feet for a bipedal robot. The projected support polygon is depicted by the blue area and the projected one by the red chessboard pattern. As opposed to the true support polygon, the projected one is smoothly morphing without discontinuities. They match if and only if both feet are flat.

where the extraneous variable ξ is called *Divergent Component of Motion* (DCM) because it is repelled by the ZMP (see figure 2.11). Indeed, the DCM is diverging over time if the position of the ZMP is not regulated. On the contrary, the CoM is attracted to the DCM. Consequently, regulating the ZMP suffices to prevent the full state from diverging and hence avoid falling. It is easy to show that the DCM matches the Capture Point if the inverted pendulum model is exact: moving the ZMP to the DCM stops the evolution of the DCM, so that the CoM exponentially converges to the DCM until the whole motion ends completely.

Let us define the projected support polygon as the virtual support area associated with the horizontal plane passing through the DCM and involving all the candidate contact points including those not touching the ground (see figure 2.12). The Capture Point is not confined to the projected support area unlike the ZMP. In this perspective, an appropriate dynamic stability metric would be the distance of the DCM from the border of the projected support polygon.

During single support phases, moving the swing leg will reshape the projected support polygon, thereby increasing the proposed stability metric. The legs of most bipedal robots have powerful actuators to withstand the full weight of the robot and limited inertia, allowing very fast motion of the swing leg. Therefore, the DCM is likely to end up in the projected support polygon before impact and subsequently the true one at impact. At this point, regulating the ZMP would be sufficient to keep balance assuming the system is fully-actuated with unbounded controls. This foot placement strategy is natural for humans when falling backward (Goswami & Vadakkepat, 2019, Part VI; Boer, 2012, Chapter 5). Another strategy would be to lower the center of mass. It is also permitted by the proposed metric since it would limit the divergence of the DCM and make it easier to catch up. Mesesan et al. (2021) rely on this strategy to maintain balance if ankle control is not sufficient. They show that it roughly doubles the maximum force impulse that the robot can withstand. They were able to recover stability for pushes up to $0.5 N.s.kg^{-1}$ in simulation on the humanoid robot TORO, which is competitive against optimal control methods.

The projected support polygon is morphing continuously over time and the DCM

is evolving continuously. Therefore, the distance of the DCM from the border of the projected support polygon is also continuous. It makes this metric very appropriate as an objective function for trajectory planning and policy learning since it provides a well-defined gradient valid whatever the configuration of the robot and the contact state. However, estimating the DCM may be harder than the ZMP on the real platform since it requires estimating the position and velocity of the center of mass, which is not directly observable. Moreover, it relies on the assumption that the robot can be approximated by a pendulum, which may not be valid in practice. Besides, it implicitly assumes that the robot is always in contact with the ground, but the contact does not have to be stable.

2.2.3 Classical Control Methods

Trajectory-Based Model-Free Motor Control

Proportional-Integral-Derivative controllers (PIDs) are simple yet effective feedback controller (Siciliano & Khatib, 2008). They are ubiquitous in robotics and are mainly used for position and velocity control of actuated joints (Boer, 2012; Goswami & Vadakkepat, 2019). PIDs at the joint level have been studied thoroughly both in theory and reality over the past century (Åström & Hägglund, 1995). They mimic the dynamics of spring-damper mechanisms whose stiffness and damping can be adjusted. If tuned properly, they are moderately robust to model uncertainties, sensor noise, and communication delay.

It takes as input the target position and velocity computed by a higher-level controller running at a lower frequency. The objective of the PID is to reach this target exponentially in time, at a given precision,

$$u_j^*(t) = K_{P,j}\left((q_j(t) - q_j^*(t)) + K_{I,j}\max\left(-I_e, \min\left(I_e, \int_0^t (q_j(\tau) - q_j^*(\tau))\,\mathrm{d}\tau\right)\right) + K_{D,j}(\dot{q}_j(t) - \dot{q}_j^*(t))\right), \quad (2.47)$$

where $q_j(t), q_j^*(t)$ are the relative current and target position of joint j at time t respectively, $u_j^*(t)$ is the target torque, $K_{P,j}, K_{I,j}, K_{D,j}$ are the proportional, integral and derivative gains respectively, and I_e is used to clamp the maximum absolute value of the integral term. Clamping is necessary to limit the command torque if a joint got stuck after some unexpected event. This phenomenon is referred to as *integral windup*. It would also occur in the situation where the PID target undergoes a large change. The integral term would accumulate a significant error during the rise, to finally overshoot the target and continue to increase because the integral term is unwound. However, the integral term is important as it compensates static errors at rest to enable perfectly reaching the target, which would be impossible otherwise. The discrete-time version is obtained by replacing $q_j(t)$, $q_i^*(t)$, $\dot{q}_j(t), \dot{q}_i^*(t)$ by their discrete-time counterparts, and the integral term by the clamped sum over

all iterations since startup of the robot. The output torque is held constant until the next update. Note that it would be equally valid to output the target acceleration instead of torque. It has the advantage to render the temporal response agnostic to the load of the motor. In the end, computing the target torque is still necessary. Doing so requires a model as the torque and acceleration are related by the apparent inertia of each joint, i.e. their composite subtree inertia.

Finet (2018) has proven that basic high-gain PIDs at the joint level form a good approximation of the optimal input-output feedback linearization (cf. appendix A.2.2) if the coupling between the actuators is not too strong. Notably, there are strictly equivalent if the actuators are perfectly decoupled. In practice, the coupling is weak for robots having transmissions with high reduction ratios, which means that the apparent inertia of the joints is large. It is typically the case for Atalante, and therefore good performances can be expected using high-gain PIDs.

Low-gains PIDs emulate compliance. It accommodates discrepancies between the real and planning environment instead of fighting against it and thereby alleviates the need for accurate planning. For instance, the stance foot will stay flat on the ground regardless of the ground profile. However, the tracking error will increase and the robot may fall because of it. A time-dependent feedforward term precomputed offline could be added to further improve the tracking accuracy. This term corresponds to the theoretical torque to realize the nominal trajectory and is computed by FD. Thus, the efficiency of the approach heavily depends on the validity of the theoretical model. Moreover, the theoretical torque may be discontinuous, causing loud noise and discomfort for the user inside. For these reasons, high-gains PIDs without feedforward term are often preferred to ensure accurate tracking and reduce compliance to a bare minimum. Tuning high-gain PIDs is not straightforward and usually involves heuristics. If the coupling between actuators can be neglected, then Ziegler-Nichols method (1993) can be used to tune each motor individually. Otherwise, it gets much tougher. As a sub-optimal workaround, each actuator can still be tuned independently, reducing the proportional gains causing vibrations in closed-loop in a second stage.

Tracking the trajectory in joint space does not require any model but has several limitations. Regarding humanoid robots, in single support, the tracking error for the pose of the flying foot is going to be much worse than any other body because of the compounding of errors along the whole kinematic chain. Similarly, PIDs at the joint level have no direct control over the position of the CoM nor the state of the freeflyer, so they cannot prevent these quantities from diverging. For humanoid robots, the swing foot strikes the ground earlier than expected on flat ground because of the mechanical deformation of the structure. If nothing is done, then it would push on the ground to complete the step and fall backward in the process. One solution is to switch directly to the next as soon as impact is detected, though doing so is not straightforward as it would introduce a discontinuity in the target position and velocity sent to the controller. It will result in a burst of acceleration of the motors to cancel the tracking error as fast as possible which would destabilize the system. Optimal control approaches could be used to prevent such behavior. Yet,

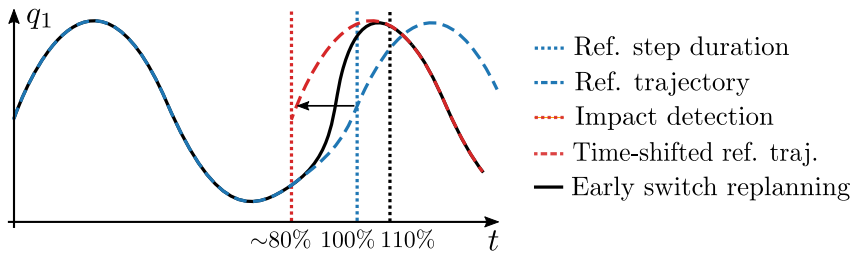


Figure 2.13: Illustration of early switch replanning. The impact is detected earlier than expected. Switching directly to the next step is necessary, but discontinuous PID targets must be avoided. A model-free heuristic is used to transition smoothly from the past to the next nominal step.

it may be hazardous to use model-based feedback control on real robots because of the modelling uncertainties. A more conservative model-free heuristic is used on the exoskeleton Atalante. It is depicted in figure 2.13. The objective is to find the shortest path to catch up with the original nominal trajectory in a fixed amount of time while enforcing continuity and bounds on position, velocity, acceleration, and jerk to get a smooth transition. It has no guarantee to generate a dynamically stable gait even if perfectly tracked on the device. Nevertheless, the swing foot is only a few centimeters lower than expected and touches the ground at about 80% of the expected step duration, such that the nominal trajectory is only marginally modified by the heuristic and the effect on stability remains limited. Gurriet et al. (2018) has realized stable walking on flat ground with the exoskeleton Atalante using this technique. Tracking kinematic features such as the pose of the flying foot and pelvis alleviates this issue. It requires a model, but it is fairly reliable since it does not involve the dynamics (Apgar et al., 2018). However, it is still not robust enough to handle unexpected disturbances, and it does not take into account any of the stability criteria presented in section 2.2.2.

Trajectory-Based Model-Based Reactive Control

Early impact handling is effective to avoid falling backward systematically but does not help to maintain stability during the whole motion. It is sufficient for almost statically stable slow-paced motions such as flat foot walking, but not for dynamic motion such as foot rolling or stair climbing. As soon as the robot starts to lose balance, it will quickly fall if nothing is done to recover stability. Let us assume a nominal trajectory has been computed offline for the theoretical whole-body model to ensure stability without disturbances in simulation. The objective is to maintain stability while tracking this trajectory on the real device, rather than recovering stability once the robot has already started falling. Mathematically, the basin of attraction of the nominal trajectory must be maximized.

Hirai et al. (1988) have demonstrated that it was possible to regulate the position of the ZMP without any model. It is based on the observation that adding a tipping

momentum to the pelvis proportionally to the error is sufficient. They define the scaling factor as the theoretical vertical inertia of the upper body in the standing pose, but it would be easy to tune it manually. This approach improves stability, but it does not bring any guarantee regarding the temporal response. Chevallereau et al. (2008) introduce a whole-body model-based approach that addresses this shortcoming. It has strong mathematical guarantees and allows foot tilting during walking. It is based on the zero dynamics theory presented in appendix A.2.3. Although effective in simulation on a simple 2D walker, it is unlikely to work in practice as it requires an accurate dynamic model of the system. Englsberger et al. (2011) has proven that ideally the ZMP must react proportionally to deviations of DCM relative to this reference. This solution maximizes the basin of attraction among linear feedback controllers. On the downside, it requires accurate control of the ground contact forces, which is beyond the capability of model-free approaches.

Englsberger et al. relies on textitAdmittance control at CoM level without naming it explicitly. It consists in using position-based control at the joint level to realize the desired contact forces. A simplified model is used to estimate the current position of the DCM and compute the target from the reference trajectory. The most common approximate model for bipedal locomotion on flat ground is the LIPM but more complex inverted pendulum models are also found in the literature. The admittance controller computes target positions and velocities at the joint level that is slightly off the reference to cancel out the tracking error of the DCM and consequently maintain stability. PIDs are then used to compute the target accelerations in a feature space gathering the pose of the feet, pelvis, and CoM. Finally, a weighted task-based second-order IK solver computes the motor accelerations. Later, Caron et al. (2019) introduces complementary regularization strategies based on the positions of the end-effectors, so-called whole-body admittance control. This approach is fast enough for online control. It has been demonstrated on real humanoid robots, e.g. Atlas (Englsberger et al., 2015) and HRP-4 (Caron, 2020; Caron et al., 2019).

Although efficient in practice, admittance control can only handle small disturbances as it relies on local modifications around a reference at the current time. Once stability is lost, anticipating the fall is necessary to prevent it. For example, apart from lowering the position of his CoM, it is natural for a human to transfer all the weight on a single foot to free the other and move the latter in the direction of the Capture Point. This kind of *foot placement* strategy implies anticipation but not necessarily planning ahead of time by integrating explicitly the equation of motion (Goswami & Vadakkepat, 2019, Part VI; Boer, 2012, Chapter 4). Typically, the Capture Point carries some information about what will happen next based on the LIPM model and where to land the foot to stop the fall. So, one solution is to rely on the projected support polygon to maximize the stability over time instead of tracking the DCM. Contrary to admittance control, maximizing this objective can exhibit natural recovery strategies such as foot placement. It is providing much more freedom than admittance control as it allows not only to regulate the contact forces to move the DCM closer to the projected support polygon but also to modify the future support polygon to anticipate impact with the ground and next steps. However, it is incon-

sistent with tracking the reference trajectory, so it is likely to significantly alter the reference trajectory. Nevertheless, a weighted task-based inverse kinematics solver can be used to trade-off between tracking and stability. Very recently, Mesesan et al. (2021) have validated this approach of the humanoid robot TORO.

Trajectory-Free Model-Based Optimal Control

Anticipating future events without planning ahead explicitly is possible. For instance, the Capture Point determines where the flying foot is going to touch down in the future. However, to predict temporal information such as impact time, it is necessary to rely on a model and integrate the dynamics. Model-based optimal control is going one step further: instead of predicting what is going to happen over a time horizon, it is looking for how to perform the desired task optimally. Formally, model-based optimal control consists in maximizing the future return based on a dynamic model and starting in a given state. There is no feedback controller involved as it outputs the optimal command sequence directly, thereby avoiding relying on suboptimal PIDs at the joint level. The resulting *Optimal Control Problem* (OCP) is very similar to the trajectory planning problem (A.29) presented in appendix A. Planning well in advance allows for performing more efficient and versatile motions by being less conservative. Indeed, 2-steps capability is less restrictive than 1-step capability.

Even though the entire optimal command sequence over the planning horizon could be readily applied to the system, it would not work in practice because of the various discrepancies between simulation and reality. First, the theoretical model is always approximate so that the future state is getting less and less reliable as time goes by. Secondly, even if the model is perfectly known, the actual trajectory would diverge as soon as anything does not go exactly as planned due to local disturbances. The ensuing error tends to increase exponentially for systems associated with stiff differential equations such as legged robots. Finally, hazards such as external pushes may occur in the meantime, instantly invalidating the planned state and command sequences. To get around these limitations, only the next optimal command is applied, or a very short window. Then, the new state is observed, and the optimization process is solved once again in accordance. This approach is more reasonable as it curbs the propagation of errors and increases the reactivity of the system. This approach is termed *Model-Based Predictive Control* (MPC) since computing the next command involves predicting the future by integrating the differential equations of motion for a given model (Grüne & Pannek, 2011). It is illustrated in figure 2.14.

The main drawback of MPC is its computational cost. For complex systems such as legged robots, the original OCP takes the form of a *Non-Linear Program* (NLP). The cost of solving such problems scales poorly with the dimensionality of the state and command spaces, when it does not fail completely. It follows that naively trying to solve the OCP online in a timely manner is impracticable using limited embedded computational resources. Warm-starting partially addresses these issues but is rarely sufficient (cf. section 4.1.1). On top of that, the OCP is simplified to make computations more tractable, starting with the model of the system. Kajita et al.

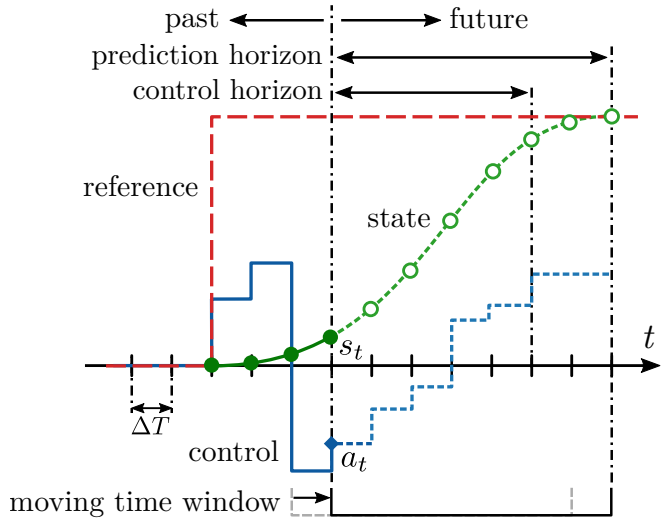


Figure 2.14: Model Predictive Control over a finite horizon.

(2003) were the first to regulate the ZMP using model-based optimal control for the LIPM. In their case, the time is discretized, the model is linearized, and the objective function is quadratic. Because of these properties, it admits a closed-form solution in the unconstrained case, which is very cheap to evaluate compared to solving an optimization problem. If there is any inequality constraint, then it does not admit a closed-form solution, but it can be solved quickly and reliably since the problem is convex. This particular type of MPC is called *Linear Quadratic Regulator* (LQR) (Anderson & Moore, 2007). A LQR can be viewed as a PID controller with feedforward whose gains are nearly-optimal in the close vicinity of the nominal trajectory. Thus, the LQR formulation is one way to tune analytically centralized PID gains to achieve the desired response, providing that a model is available (He et al., 2000). Wieber (2006) have slightly extended Kajita et al.'s method to use a linear MPC. It can handle moderate external disturbances in simulation.

Added to that, the original problem is broken up into smaller ones treated hierarchically. It speeds up computation even further by decoupling the whole optimization into independent smaller problems that are much faster to solve. Traditionally, for legged locomotion, it is broken up in three stages (Apgar et al., 2018; Caron & Kheddar, 2017; Dai et al., 2014; Herdt et al., 2010):

1. adjusting the position and timing of the next footprints to handle capturability and navigation concerns if necessary
2. finding a stable trajectory for the centroidal dynamics that is consistent with the footprint constraints
3. computing the motor torques using weighted task-based second-order IK

The first two stages generally rely on an inverted pendulum model while the last one necessitates an accurate whole-body model. It is challenging to guarantee that the sub-problems are compatible with each other, such that it exists at least one feasible

solution at a given stage that satisfies the constraints enforced by the previous one. The failure rate of the whole optimization may be very high because of this lack of consistency. It reaches 40% for humanoid locomotion on rough terrain (Caron & Kheddar, 2017). Budhiraja et al. (2019a) overcome this issue by enforcing a consensus between all stages, at the cost of a moderate increase in the computation burden. Their approach is very similar to the one we present in chapter 5. State-of-the-art control methods based on MPC for legged robots are reviewed in section 4.1.1.

Chapter 3

Background in Machine Learning

Contents

3.1 Supervised Learning and Neural Networks 49
3.1.1 Introduction to Supervised Learning 49
3.1.2 Fundamentals of Artificial Neural Networks 54
3.2 The Reinforcement Learning Problem 65
3.2.1 The Agent-Environment Interaction Loop 66
3.2.2 Key Concepts and Terminology 73
3.2.3 Taxonomy of Algorithms for Continuous Control 86

3.1 Supervised Learning and Neural Networks

3.1.1 Introduction to Supervised Learning

Supervised Learning consists in fitting a function mapping an input to an output based on example input-output pairs. The input-output pairs that will be used for training are called *training examples* or *training samples*, while the whole dataset of examples is the *training set*. The inputs are called *instances* and are characterized by a set of quantifiable properties known as *explanatory variables* or *features*.

Two class of problems falls in this category, *classification* and *regression*. In classification, the outputs are called *labels* and uniquely identify each instance. Those labels are elements from a finite set of categories of interest for a given application. A typical scenario would be to infer the labels of unseen instances based solely on the training set. This key capability of leveraging available data to predict properties of new instances is referred to as *generalization ability*. One famous example is handwriting recognition. In this case, the features are the gray-scale value of each pixel of the handwritten character – i.e. a scalar value representing an amount of light – and the label is the corresponding letter or digit. In regression, the output is rather called *dependent variable* or *response*. The objective here is to model the relationship between the features and the response. Contrary to classification, both the features and the response are continuous quantities, which means the input and output spaces are an uncountable set. Regression is relevant when generating new input-output pairs is costly or even impossible. It can also be used to remove noise

from data, analyze the causal relationship between the features and the response, or simply compress the training set. Using the regression model to predict values within the range of values of the available features is commonly termed *interpolation*, while predicting values outside this range is termed *extrapolation*. Regression is mainly used for interpolation because extrapolation relies on strong modelling assumptions. One example of regression frequently used in robotics is system identification, which consists in finding optimal values for unknown parameters of the physically-based model, such as the dry and viscous friction of an actuator.

In the following, we present the rigorous formulation of the learning problem described above, with a focus on regression since it is central in chapter 5. Let us denote $\mathcal{X}$ the input space and $\mathcal{Y}$ the output space. We define the training set comprising N samples as the finite set $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{N} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ s.t. $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. Supervised learning aims at finding a function $f : \mathcal{X} \to \mathcal{Y}$ mapping inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$. This function f is called *classifier* or *function approximation* for classification and regression problems respectively.

The function f can be either *parametric* or *non-parametric*, depending on the algorithms. Such non-parametric functions are deterministic and only depend on the training set without involving any extrinsic parameter. The resulting function approximation does not take any predetermined form but is constructed from the information derived from the data. Formally, a scalar *scoring function* $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ conditioned by the training set $\mathcal{B}$ can be used to represent the function f, such that $f(x)$ corresponds to the value of y associated with the highest score $f(x) = \arg\min_{y \in \mathcal{Y}} g(x, y)$. *K-Nearest Neighbors* (KNN) are probably the most famous non-parametric models, along with *local regression*, which is a generalization of moving average (Ruppert & Wand, 1994) and smoothing splines. Eventually, basic pre-processing steps remove unnecessary data or reduce the dimensionality. It helps to compress information and improve prediction accuracy. One of the major concerns about non-parametric methods is their scalability to large datasets. This is conditioned by the ability to efficiently prune unnecessary samples to prevent the cost of evaluating f from growing unbounded. This procedure is an optimization process by itself. Despite pruning, the accuracy of non-parametric models increases very slowly with the amount of data available, which is not surprising because the data must supply both the model structure and the estimates. Nevertheless, they are sometimes used in robotics, notably for picking a good initial guess for complex optimization problems (cf. sections 4.1.1 and 5.2.3).

Conversely, a parametric function is fully specified by a vector θ of size m and is denoted g_θ to highlight this particularity. The vector θ gathers variables that are usually continuous but may well be discrete. The number of parameters m should reflect the complexity of the mapping. It implies that the cost of evaluating the function approximation g_θ is invariant to the number of training samples N. The challenge here is to choose an appropriate parameterization achieving a good trade-off between its *expressive power*, defined as its capability to approximate with arbitrary accuracy a large variety of continuous functions, and the number of parameters to ensure high generalization ability. Widespread examples of such parameterization are

the monomial basis, which is widely used in many fields where the amount of data available is limited and very noisy including robotics, and artificial neural networks, which will be presented soon after.

From now on, let us focus only on the parametric case. First, a loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ function that measures the distance between elements of $\mathcal{Y}$ must be specified. It will be used to evaluate the *prediction error* $l(y, \hat{y})$ between the true outputs y and the values $\hat{y}$ returned by the function for each training sample $f(x)$ and thereby quantify how well the function f fits the training set.

Given the joint probability distribution $\mathcal{D}(\mathcal{X} \times \mathcal{Y})$ of the samples, the objective is to minimize the expectation of the prediction error, which is called the *true risk* or *approximation error* for classification and regression respectively,

$$R_{\mathcal{D}}(f) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[l(y, f(x)) \right]. \tag{3.1}$$

Let $\mathcal{F}$ denote the set of all possible functions, such that each element of $\mathcal{F}$ is a function f mapping from $\mathcal{X}$ to $\mathcal{Y}$. An optimal function f^* minimizes the approximation error $R_{\mathcal{D}}$ over the set of possible functions $\mathcal{F}$ and the joint distribution $\mathcal{D}$,

$$f^* = \arg\min_{f \in \mathcal{F}} R_{\mathcal{D}}(f). \tag{3.2}$$

For classification, the usual loss is based on the binary indicator function,

$$l(y, \hat{y}) = \mathbb{1}_{\{y\}}(\hat{y}), \tag{3.3}$$

where $\mathbb{1}_{\mathcal{Y}'}(y) = 1, \forall y \in \mathcal{Y}'$, 0 otherwise. This loss is non-convex, and finding a near-optimal solution is an NP-hard problem (Ben-David et al., 2003). Instead, a common approach is to consider a convex approximation of the original problem. The model is no longer deterministic but probabilistic so that the function approximation $f(x)$ is a (log-concave) probability density function, e.g. a Gaussian distribution. Then, the loss is then replaced by the negative log-likelihood of the true data according to the probabilistic model. It yields,

$$l(y, y') = -\log \mathbb{P}(y|y') = -\log \left[f(x) \right]_y. \tag{3.4}$$

For regression, assuming the output space is Euclidean, the L^p-norm is typically used,

$$l(y, \hat{y}) = \|y - \hat{y}\|_p^p = \sum_{i=1}^{m} (y_i - \hat{y}_i)^p. \tag{3.5}$$

In practice, the joint distribution $\mathcal{D}$ is unknown. Therefore, the true error cannot be computed directly and one has to replace with a *surrogate*. Given that training samples are available, a natural candidate is the sample mean over the whole training set $\mathcal{S}$. We define the *empirical risk* or *estimation error* for classification and regression respectively as follows,

$$R_{\mathcal{S}}(f) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{N} l(y_i, f(x_i)). \tag{3.6}$$

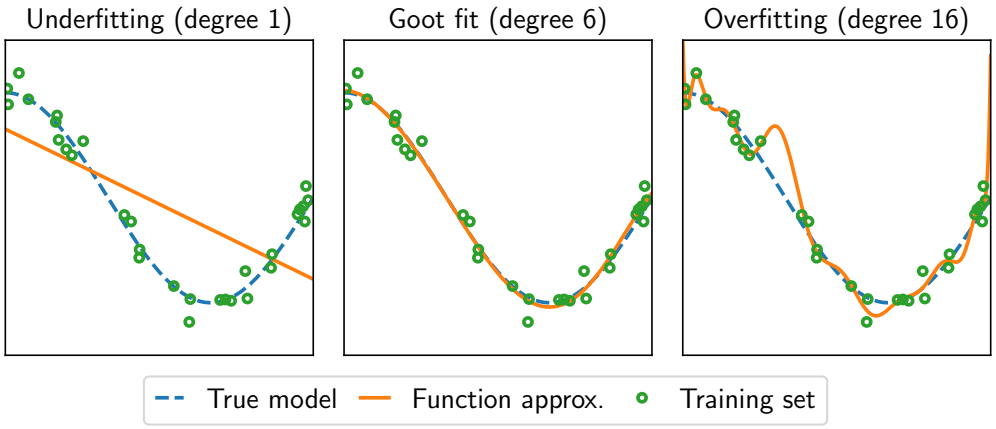
Since the sample mean is an unbiased estimate, a function $\tilde{f}$ minimizing the estimation error is guaranteed to minimize the *approximation error* if and only if the number of samples N is infinite and the samples are drawn independently and identically according to $\mathcal{D}(\mathcal{X} \times \mathcal{Y})$ where $\mathcal{X} \times \mathcal{Y}$ is a compact space. These conditions never hold in practice. Still, intuitively, increasing the number of samples or decreasing the complexity of the function space $\mathcal{F}$ both decrease the ensuing extra error, which is called *complexity penalty*. An upper bound to the approximation error by the estimation error and a complexity penalty is called *generalization bound*.

The complexity of a function space $\mathcal{F}$ is also termed capacity, richness, or expressiveness. This notion is difficult to define rigorously in the general case. Yet, one can reckon it depends on the regularity of the function f, which is quantified by its Lipschitz constant for continuous functions, and the size of the function space $\mathcal{F}$, namely its cardinality $|\mathcal{F}|$ when countable. It exists multiple metrics to assess the complexity of an uncountable function space $\mathcal{F}$ depending on problems. For instance, the *Vapnik-Chervonenkis dimension* has been proposed for binary classification and is defined as the largest training set that the algorithm can learn to perfectly fit. Similarly, the *Rademacher complexity* measures the richness of a class of real-valued functions with respect to a given probability distribution. These metrics are then used to derive the complexity penalty. Alternatively, assessing the complexity penalty directly based on the Lipschitz constant has drawn a lot of attention during the last few years, especially in the field of adversarial robustness of neural networks. In such a case, evaluating the Lipschitz constant can be extremely costly. Providing a tight upper bound that is scalable and efficient to compute is an active research topic.

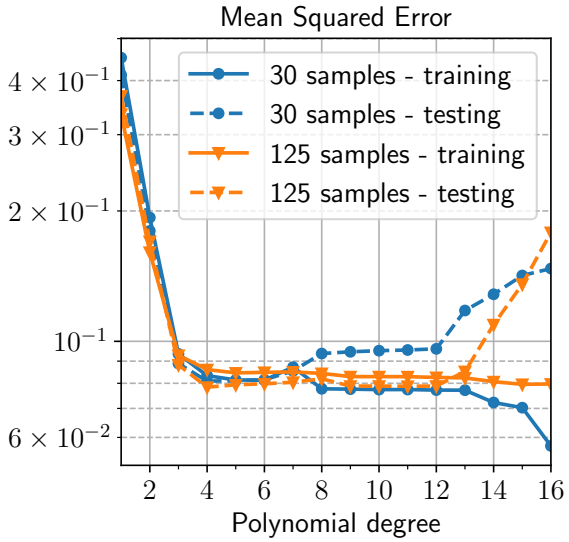
The true error can get arbitrarily large if the function f can be discontinuous. For instance, consider the function to fit perfectly the training set and infinite otherwise,

$$f(x) = \begin{cases} y_i & \text{if } \exists (x_i, y_i) \in \mathcal{S} \text{ s.t. } x_i = x \\ \infty & \text{otherwise} \end{cases}.$$

Clearly, it seems already more reasonable to output the value of the closest neighbor if the input is unseen because it is likely to be similar. Defining the function space $\mathcal{F}$ based on assumptions about the relationship between the input and the output is called *inductive bias*. For example, it is commonly assumed that the mapping to be linear to reduce the set of possible candidates. It will significantly reduce the complexity penalty, but also increase both the estimation error and the approximation error. Indeed, If the complexity of the function space $\mathcal{F}$ is too high, then the function f will fit most perfectly the training set but will have an error order of magnitudes higher on unseen samples. This phenomenon is called *overfitting*. In contrast, if the complexity is too low, the error will be about the same for training data and unseen samples, but this error will be larger than it could be. This phenomenon is called *underfitting*. This compromise is referred to as the *bias-variance trade-off* and is illustrated in the classical example of polynomial regression in figure 3.1. The optimal trade-off is achieved when the training and testing errors are similar, which means good generalization ability, but starts to differ if the complexity is further increased.



(a) Optimal function approximation for different complexity



(b) Training and testing error for different complexity and number of samples

Figure 3.1: Effect of function space complexity and number of samples on polynomial regression of the cosine function with small normal noise. The complexity must be small to avoid overfitting when few samples are available, but it is less of an issue when they are numerous. Moreover, more samples lead to lower approximation error since one can train over a richer function space for the same complexity penalty.

As seen above, picking the right complexity for the function space is not straightforward. One solution to get optimal accuracy is to perform a higher-level optimization. First, the original training set is separated into an actual training set for fitting the model and a testing set for evaluating its generalization ability. Then, the optimal function is found for several function spaces, and the associated empirical error is computed on the testing set. Finally, the function space giving the lowest empirical error is selected. This approach is known as *cross-validation*.

Reduced complexity comes with more restrictive global regularity properties: it would be impossible to get a good fit if more complexity is locally required. A simple way to adapt this trade-off automatically is to jointly minimize the empirical error and the complexity of the function approximation as a weighted sum. This kind of penalty term is known as *regularization function*. Let us define a regularization function $r : \mathcal{F} \to \mathbb{R}_+$, which takes a function approximation as input and measures a surrogate of its complexity. The optimization problem becomes,

$$f^* = \arg\min_{f \in \mathcal{F}} (R_{\mathcal{D}}(f) + r(f)). \tag{3.7}$$

Classic examples of regularization for parametric models are the L^1- and L^2-norm of the vector of parameters θ. The ensuing optimization problems are called respectively *Lasso* and *Ridge regression*. The L^2-norm regularization is also termed *Tikhonov regularization* (Tikhonov et al., 1995). It shrinks all coefficients globally but none of them would be close to zero. Although effective to avoid over-fitting, it prevents large values entirely, which is often detrimental in practice. On the contrary, the L^1-norm encourages sparsity. Most of the coefficients would be exactly zero, with a few of them growing very large. In the end, the properties of *Lasso* are usually preferable over the *Ridge regression*, but the non-convexity of the L^1-norm and the discontinuity of its gradient make *Lasso* hard to solve in practice.

3.1.2 Fundamentals of Artificial Neural Networks

Artificial Neuron and Fully-Connected Feedforward Neural Network

An *Artificial Neural Network* (ANN) is a collection of interconnected units called *artificial neurons*. Artificial neurons are inspired by the ones in a biological brain. Each of them receives multiple signals x_i simultaneously coming from input connections as a real-valued vector x, then it mixes those signals using a weighting sum depending on the strength of each connection w_i and finally processes this aggregated signal including a fixed bias through a non-linear function σ called *activation function* to output a real number y. The mechanism is illustrated figure 3.2.

The capability of a single neuron is limited but nonetheless interesting. Assuming the activation function is the hyperbolic tangent, then it can be used for binary classification to split in two the input space with a hyperplane. Several hyperplanes can be combined using more neurons to create complex clusters, for instance by taking the maximum output among them, as seen in figure 3.3. Typically, neurons

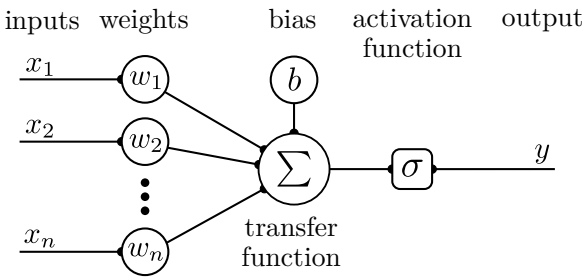


Figure 3.2: Illustration of an artificial neuron

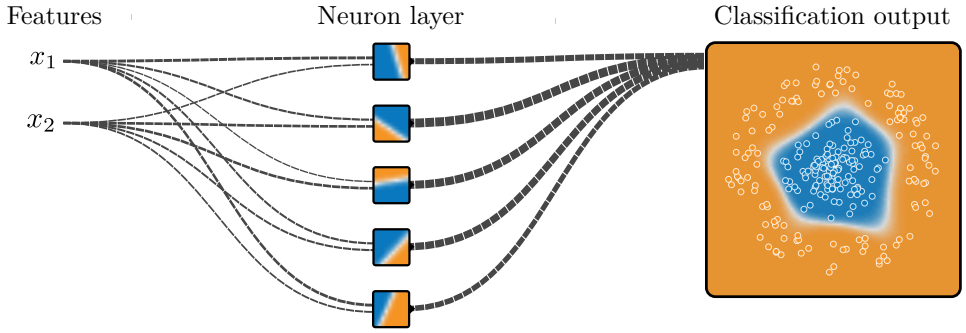


Figure 3.3: Simple binary classifier with a single layer and hyperbolic tangent activation function (adapted from Tensorflow Playground).

are gathered into layers. Each layer may use a different activation function. The input travels from the first layer called *input layer*, to the last layer called *output layer* through some intermediary layers called *hidden layers*. The number of layers excluding the input layer is referred to as the *depth* of the network, while the number of neurons per layer is termed *width*. Given a depth $n \in \mathbb{N}^*$, let $w = (w_i)_{i=0}^{n+1}$ be the sequence of layer widths, $\theta = (\theta_i := (W_i, b_i))_{i=1}^{n}$ the sequence of weights matrices and bias vectors such that $W_i \in \mathbb{R}^{w_i \times w_{i+1}}$ and $b_i \in \mathbb{R}^{w_{i+1}}$ and a sequence of activation functions $\sigma = (\sigma_i)_{i=1}^{n+1}$ such that $\sigma_i : \mathbb{R}^{w_{i+1}} \to \mathbb{R}^{w_{i+1}}$. Let $\mathcal{X} \in \mathbb{R}^{w_0}$ and $\mathcal{X} \in \mathbb{R}^{w_{n+1}}$ be the input and output spaces respectively. A *Feedforward Neural Network* (FNN) is the function $f_{\theta,\sigma} : \mathcal{X} \to \mathcal{Y}$ such that

$$f_{\theta,\sigma}(x) = \left(\bigcirc_{i=1}^{n} \phi_{\theta_i,\sigma_i} \right)(x) = \left(\phi_{\theta_n,\sigma_n} \circ \cdots \circ \phi_{\theta_2,\sigma_2} \circ \phi_{\theta_1,\sigma_1} \right)(x), \tag{3.8}$$

where $\phi_{\theta_i,\sigma_i} : \mathbb{R}^{w_i} \to \mathbb{R}^{w_{i+1}}$ is the transfer function of the layer i and is defined as follows $\phi_{\theta_i,\sigma_i}(x) = \sigma_i(W_i x + b_i)$. The explicit dependency on σ is usually dropped since it is not a free parameter but fixed a priori in contrast to θ. It is known as a *Multi-Layer Perceptron* (MLP) when the weight matrices are dense and unstructured.

Backpropagation and Stochastic Optimization

The generic approach for minimizing the empirical risk or estimation error is by gradient descent. It relies on the *backpropagation* algorithm, which was introduced

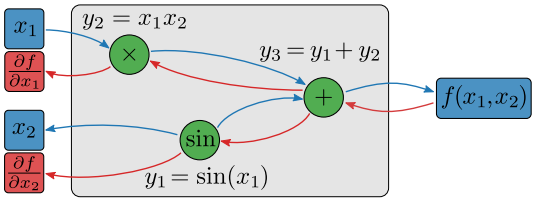


Figure 3.4: Illustration of backpropagation for the function $f(x_1, x_2) = x_1 x_2 + \sin(x_2)$. The function f is decomposed in a set of elementary functions $\{\sin, +, \times\}$ for which the value and gradient is well-known, so that it forms an orientated graph. The value of the function is evaluated by the forward pass in blue, and the partial derivative with respect to each input x_1, x_2 is computed by the backward pass in red.

by Rumelhart et al. (1986). The gradient of the outputs with respect to the network parameters is computed backward using the *chain rule* recursively. Let f_θ be a FNN with n layers, x_i be the output of the i-th layer and ϕ_{θ_i} its transfer function, such that $x_i = \phi_{\theta_i}(x_{i-1})$. For a given sample $\{x, y\}$, the error E associated with a differentiable loss function l is given by $l(y, f_\theta(x))$. It yields,

$$\frac{\partial E}{\partial \theta_i} = \frac{\partial E}{\partial x_i} \frac{\partial \phi_{\theta_i}}{\partial \theta_i}\bigg|_{x_{i-1}}, \qquad \frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial x_{i+1}} \frac{\partial \phi_{\theta_{i+1}}}{\partial x}\bigg|_{x_i}, \tag{3.9}$$

where $\frac{\partial E}{\partial x_n} = \frac{\partial l}{\partial x}(y, x)$ is known analytically. The first equation computes some terms of the gradient of the network $\nabla_\theta E$ with respect to the parameters, while the second equation propagates backward the partial derivative with respect to the input.

During each training iteration, the neural network's parameters are updated proportionally to the gradient $\nabla_\theta E$. Evaluating the neural network for a given input is called the *forward pass*, while computing the gradient is the *backward pass*. This method is directly related to *automatic differentiation*, which is at the core of every open-source deep learning toolbox, e.g. Torch by Meta or Tensorflow by Google among others. It is illustrated in a simple example in figure 3.4.

Thanks to the linearity of the operators, a bunch of data can be processed simultaneously and efficiently by generalizing matrices to tensors. It is referred to as *batch processing*. The gradient of the loss $R_\mathcal{S}(f_\theta)$ in equation (3.6) with respect to the network parameters θ can be computed as follows,

$$\nabla_\theta R_\mathcal{S}(f_\theta) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{N} \nabla_\theta l(y_i, f_\theta(x_i)). \tag{3.10}$$

It is straightforward to find a local minimum of the loss with any first-order iterative optimization algorithm known as *Gradient Descent* algorithms. At each iteration i, the loss and its gradient are evaluated, then the network parameters are updated by doing a step in the opposite direction of the gradient,

$$\theta_{i+1} = \theta_i - \eta \nabla_\theta R_\mathcal{S}(f_\theta)|_{\theta_i} \tag{3.11}$$

where η is the *update step* or *learning rate*. Generally, the number of training samples is so large that evaluating the loss and its gradient would be prohibitive, because of memory capacity or computational power. So, the gradient is not computed over the whole dataset of training samples but on a subset of fixed size termed *mini-batch*. The mini-batch is randomly selected at each iteration. The resulting stochastic gradient is an unbiased estimator of the true gradient. This modified optimization algorithm is referred to as *Stochastic Gradient-Descent* (SGD) algorithm (Ruder, 2016). It would never converge unless the learning rate reduces over iterations, typically with a polynomial rate. Anyway, the vanilla SGD algorithm is never used in practice because it converges slowly and tends to get stuck in poor local minima.

Second-order optimization methods, also known as Newton's methods, adjust the update step optimally for every parameter individually according to the local second-order expansion of the loss function. These methods not only benefit from a fast and steady convergence but are also more stable and find better solutions than first-order methods. They are preferred in many fields including Robotics. In this case, computing the step direction involves the Hessian matrix of the loss,

$$\theta_{i+1} = \theta_i - (\nabla^2_\theta R_\mathcal{S}(f_\theta)|_{\theta_i})^{-1} \nabla_\theta R_\mathcal{S}(f_\theta)|_{\theta_i}. \tag{3.12}$$

For probabilistic models, Kakade (2002) propose to approximate the Hessian with the Fisher information matrix, which is defined as the negative (empirical) expected Hessian of the log-likelihood, i.e.

$$F_\theta(s) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{N} \nabla^2_\theta \log \left[f_\theta(x_i)\right]_{y_i}|_{\theta_i} \tag{3.13}$$

The resulting update step is said to follow the *natural policy gradient* in that it is independent of the parameterization of the function f_θ. This Fisher matrix does not asymptotically converge to the exact Hessian necessarily. The cost of evaluating the inverse matrix H_k^{-1} is prohibitive. Instead, algorithms computing the solution x_k of the linear problem $H_k x_k = g_k$ without explicit matrix inversion are more efficient and have better numerical stability. Approximate iterative algorithms are preferred over direct methods such as the Cholesky decomposition to further reduce the computational cost. Indeed, finding the exact solution is irrelevant because of the already high variance of the gradient estimator. The number of iterations enables to trade-off between accuracy to efficiency. The *Conjugate Gradient* (CG) method is commonly used because it is well-suited for symmetric positive-definite matrices. Although this impedes the theoretical rate of convergence, it works surprisingly well in practice. Still, about 100 CG iterates may be necessary to obtain the desired accuracy. The large memory requirements and high computational complexity make this approach impracticable for large models.

Building on this, Martens and Grosse (2015) introduce *Kronecker-factored Approximate Curvature* (K-FAC) to avoid having to solve a linear problem entirely. K-FAC is a carefully crafted approximation of the Fisher matrix that is neither diagonal nor low-rank and yet can be inverted efficiently. It is derived by approximating

various large blocks of the Fisher matrix as being the Kronecker product of two much smaller matrices. While only several times more expensive to compute than the SGD, the updates produced by K-FAC make more progress in optimizing the objective, resulting in an algorithm that is much faster in practice. Unlike the previous approximation relying on CG, K-FAC works very well in highly stochastic optimization regimes. This is because the cost of storing and inverting this approximation to the Fisher matrix does not depend on the amount of data used to estimate it.

Despite all these advances, second-order methods are still rarely used in practice, as the cost of estimating the natural policy gradient is still several times larger than vanilla SGD. It is often replaced with more advanced first-order methods relying on *momentum* estimates. The momentum is the discounted cumulative stochastic gradient, which is a biased estimate of the first moment of the gradient. Instead of doing a step according to the current estimate of the gradient, the updated momentum is used. The momentum tends to steer in the same direction and is only slowly changing over time, preventing oscillations and avoiding poor local minima. Although effective in practice, it still requires scheduling the learning rate over iterations, which can be very tedious. More recently, adaptive methods to avoid having to tune it manually been introduced, e.g. *Root Mean Square Propagation* (RMSProp) (Tieleman & Hinton, 2012) and *ADAptive Moment estimation* (ADAM) (Kingma & Ba, 2015).

Unlike the other first-order methods, ADAM performs consistently and sufficiently well on a huge variety of real-world problems. It is a tremendous advantage in practice, which is why ADAM is now the de facto optimizer in the industry for deep learning applications. Yet, its performance in terms of speed, stability, and solution optimality is still much worst than approximate second-order methods such as K-FAC in practice. (Amid et al., 2022) focus exclusively on reducing the gap between first- and second-order methods when training MLP. To this end, they introduce LocoProp, a layer-wise loss construction framework that can be combined with any off-the-shelf first-order optimizer. It relies on a three-component loss, target, and regularizer combination, for which altering each component results in a new update rule. Their results are promising on standard benchmark models and datasets, but it remains to be seen how well their method works on real-world problems.

Universal Approximation Theorem

In regression problems, a parametric function is tuned to approximate the mapping between input and output over a finite dataset of samples. ANNs are especially suitable as they are known to be *universal approximators* under some conditions. Universal approximators are classes of functions that can represent any given function space of interest with arbitrary accuracy. Those conditions involve both the regularity of the function to represent and the parameter space of the approximators. Concerning ANNs, their parameter space is fully specified by the network architecture. *Universal approximation theorems* establish the density of a class of functions within a given function space of interest. The space of continuous functions and Lebesgue p-integrable functions are the main functions spaces of interest. It is an open question

whether it is possible to derive analytically necessary and sufficient conditions generic for any possible network architectures on those function spaces. It is still an active field of research, but the particular case of FNNs has been thoroughly studied. The parameter space comprises the depth of the network, the width of each layer, and the activation functions. It gets significantly more difficult for more complex network architecture, involving for instance memory, recursive cells, or skip connections. The original Universal Approximation Theorem is credited to Hornik (1991) and Hornik et al. (1989). It holds for neural networks of arbitrary width and bounded depth on the continuous function space.

Theorem 1 (Universal Approximation Theorem)**.** *Let $\sigma : \mathcal{R} \to \mathcal{R}$ be any continuous function, and $\mathcal{N}_{\theta,\sigma}$ be the class of FNNs with activation function σ and one hidden layer of arbitrary size. Let n, m denote the input and output size respectively and $\mathcal{K} \subseteq \mathcal{R}^n$ be compact. Then $\mathcal{N}_{\theta,\sigma}$ is dense in $C(\mathcal{K})$ if and only if σ is non-polynomial. More precisely, for any continuous function $f : \mathbb{R}^n \to R^m$ and ϵ arbitrarily small, there exists a continuous function $\hat{f}_\epsilon$ such that $\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \epsilon$ where $\hat{f}_\epsilon = W_2 \circ \sigma \circ W_1$.*

This result can be extended to any bounded number of hidden layers by requiring that any additional hidden layer approximates the identity function. Thus, it addresses the case of arbitrary width and bounded depth. The seemingly dual problem of FNNs with bounded width and arbitrary depth has been addressed very recently (Hanin, 2019; Kidger & Lyons, 2020).

Theorem 2 (Dual Universal Approximation Theorem)**.** *Let $\sigma : \mathcal{R} \to \mathcal{R}$ be any non-affine continuous function that is continuously differentiable in at least one point, with nonzero derivative at that point. Let $\mathcal{N}_{\theta,\sigma}$ be the class of FNNs with an arbitrary number of hidden layers of uniform size k and the same activation function σ, except for the output layer for which it is the identity function instead. Let n, m denote the input and output size respectively and $\mathcal{K} \subseteq \mathcal{R}^n$ be compact set. Then $\mathcal{N}_{\theta,\sigma}$ is dense in $C(\mathcal{K}; \mathcal{R}^m)$ with respect to the uniform norm.*

Zhou (2020) has shown similar results for *Convolutional Neural Networks* (CNNs). This type of structured neural network relies on another linear operator than the matrix-vector product. It will be presented soon hereafter.

Network Architecture Meta-Optimization

The activation function can be any function of the output vector of the preceding layer. Most of the time, it is a scalar function applied element-wise, such as the Sigmoid or ReLU. It is preferred over multivariate functions for simplicity. Indeed, the analysis of activation functions lacks a good theoretical foundation and the choice is mostly based on empirical study, hence it is convenient to restrict candidates to scalar functions. Yet, it may be necessary to choose a multivariate function in some cases, to return the maximum value of the output layer for instance.

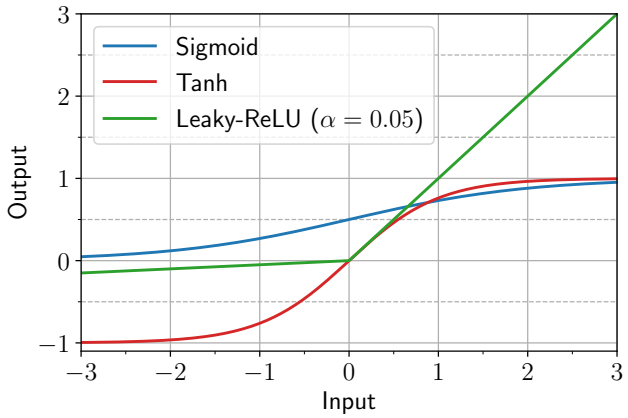


Figure 3.5: Graphical representation of three common activation functions

More exotic functions based on permutations have been proposed recently, e.g. GroupSort (Tanielian et al., 2021). They still result in universal approximators while preserving the norm of the gradient. This property is interesting as it circumvents the infamous *vanishing gradient problem* that prevents the weights from changing after only a few training iterations with backpropagation (Bengio et al., 1994). Thereupon, we present the most common activation functions illustrated in figure 3.5:

- **Logistic function (Sigmoid function or hyperbolic tangent)**

  The Sigmoid function was the first continuous nonlinear function to be used in the context of neural networks. It is defined as follows

  $$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{1}{2}\left(1 + \tanh\left(\frac{x}{2}\right)\right). \tag{3.14}$$

  It takes any real value as input, so it is applied element-wise to the input vector, and its output is bounded between 0 and 1. This function is 1-Lipschitz, which prevents the *exploding gradient problem*. Yet, it significantly shrinks the norm of the gradient, which leads to the vanishing gradient problem as the depth of the network increases. This issue occurs when the input values are large. In this case, the output is almost saturated to $\{-1, 1\}$ and the gradient is close to 0. It is sometimes replaced by the hyperbolic tangent with little or no advantage.

- **Leaky Rectified Linear (Leaky-ReLU) (Nair & Hinton, 2010)**

  The ReLU activation was proposed to address the issue of vanishing gradient. It acts as a filter canceling out negative values and leaving positive values unchanged. Its derivative is either zero or one on R^- and R^+ respectively, which tends to preserve the norm of the gradient between layers. Despite its simplicity, this function is rich enough for MLPs relying on it to be universal approximators thanks to the discontinuity of its derivative. Incidentally, it is much cheaper to evaluate numerically than logistic or trigonometric functions both involving

iterative routines. However, the gradient being exactly zero leads to so-called *dead units*. They are neurons whose weight cannot be trained anymore since the gradient of any function with respect to them would be always zero. This property leads to a sparse gradient, which impedes convergence. Later, the Leaky-ReLU function was proposed to overcome this issue. It introduces a parameter $\alpha \ll 1$ that governs the slope on R^-. It is defined as follows

$$\sigma(x) = \max(\alpha x, x), 0 \leq \alpha \leq 1. \tag{3.15}$$

- **Softmax (Bridle, 1990)**

  The Softmax function is a generalization of the logistic function to multiple dimensions. It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes. It can be further generalized by introducing a temperature parameter $\beta > 0$ to control the spread of the distribution. The lower the temperature parameter the more diffuse the distribution. It can be tuned to approximate the maximum value by a differentiable function, hence its name. The Softmax function $\sigma(x) : \mathcal{R}^n \to [0, 1]^n$ is defined by the formula

$$\sigma_i(x) = \frac{e^{\beta x_i}}{\sum_{j=1}^{n} e^{\beta x_j}}. \tag{3.16}$$

The optimal activation function is specific to each problem. The same goes for the depth and width of the network. Additionally, the size of the training dataset puts an upper bound on the number of parameters of the network to avoid overfitting, though it is unclear to what extent. To this day, the analysis of the benefits of the various network architectures lacks a good theoretical foundation and the choice is mostly based on empirical results. This issue can be partially addressed by using a parametric activation function tuned with backpropagation as any other parameters, but the effectiveness is very limited in practice. Another approach is *meta-optimization*, which consists in solving the optimization problem independently for a finite set of network architecture to select the one leading to the highest performance. Although effective, it is extremely intensive, so only a small subset can be compared. Few years ago, more sophisticated methods based on so-called *Neural Architecture Search* (NAS) or *network adaptation* have arisen (Liu et al., 2021b; Real et al., 2017). NAS is a technique that automates the design of artificial neural networks by alternatively optimizing the architecture and training it. Many improvements have been made to reduce its computational cost, but it is still very demanding and not accessible to most practitioners (Fang et al., 2021). Reducing the cost even further is an active area of research. Concurrently, *Evolutionary Algorithms* (EA) has been used for more than twenty years to jointly search for the optimal neural architecture and the weights of the corresponding neural network. This approach is termed *NeuroEvolution* and is well-suited for small to medium scale neuron networks (Stanley & Miikkulainen, 2002). As such, it has mostly been applied to learning control policies.

Structured Neural Networks

If there is no restriction on the weight matrices W_i, the layers are said to be *fully-connected*. A FNN has a very large number of parameters to train. For example, a neural network with uniform width n has $n(n+1)$ parameters per hidden layer. It is not uncommon for a FNN to accumulate a few millions of parameters to perform seemingly basic tasks. The optimal depth and width are problem specific. Their simple feedforward architecture and the redundancy of parameters make it surprisingly easy, fast, and cheap to train them at first. However, this premature convergence to a suboptimal local minimum requires a much larger depth and width than necessary. This property ultimately limits their capability to tackle complex tasks. First, increasing the depth may lead to the vanishing gradient problem, though it is possible to mitigate this issue by choosing the activation function appropriately. Secondly, large depth and width imply important expressiveness of the model, which leads to overfitting if nothing is done to prevent it through regularization. Finally, they get expensive to evaluate as they grow larger, which makes them impractical for embedded use cases. Researchers have designed specialized linear operators and network architectures to reduce the number of parameters, speed-up evaluation, or have better properties for a specific problem.

Structured neural networks aim to reflect inherent properties of the problem in their architecture rather than through training. This helps to avoid overfitting by reducing the number of parameters while guaranteeing some desired properties that would be hard to enforce differently. CNNs are typical examples of structured neural networks with a specialized linear operator called *discrete convolution*. Lecun et al. (1998) popularized it, and it is now the backbone of many sophisticated architectures (Krizhevsky et al., 2017; Tan & Le, 2019). The convolution operator encodes the translational invariance of the input, which is desirable to process images. It also forces the network to leverage information locality and eventually multiscale correlations with *dilated convolution* (Dumoulin & Visin, 2016). Besides, it allows for a very compact representation of the weight matrix called *convolution kernel*.

Let us consider a vector x of size n and a kernel K of size k. The 1D convolution of x by K with stride s and zero-padding of size p is the vector of size $\lfloor (n-k+2p+s)/s \rfloor$ defined as

$$(K * x)[i] = \sum_{j=-\lfloor k/2 \rfloor}^{\lceil k/2 \rceil} \begin{cases} f[s\,i-j]g[j] & \text{if } s\,i-j \in [0,n] \\ 0 & \text{otherwise} \end{cases}, \tag{3.17}$$

Padding consists in extending the input vector x at its boundaries, essentially to preserve the output size when the stride is 1. Zero-padding is the most common but is sometimes replaced with same-padding, which holds constant the actual boundary values, or circular-padding, which repeats the input vector as if it was periodic. It is straightforward to generalize this formula to higher dimensional inputs. The dimensionality of the kernel is the same as the input. In the case of monochrome images, it is a $k \times k$ matrix where k is the *kernel size*, as illustrated in figure 3.6. A

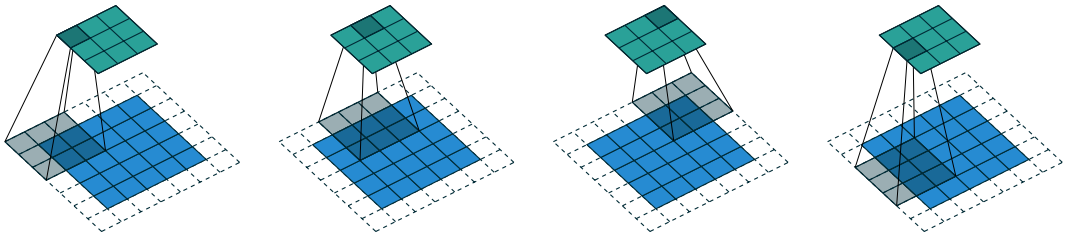


Figure 3.6: Discrete convolution acting as a filter. 3×3 kernel sliding over 5×5 image padded with 1×1 border of zeros using 2×2 stride. (Dumoulin & Visin, 2016)

classical linear layer with a dense matrix has n^2 parameters, whereas a 1D convolution layer only has k parameters where $k \ll n$ is usually very small, e.g. 3 or 5.

Any given convolution kernel applies a specific *filter*. It may be interesting to apply several filters to extract different features. Each of these filters corresponds to one output channel. They are stacked into a single kernel representation by adding one extra dimension to it and generalizing the convolution operator. Similarly, it is desirable to apply the same filter to several input channels so that it can perform cross-correlation, e.g. the one-dimension time series of the relative joint angles of poly-articulated robots or the three monochrome pixel arrays of RGB images. The results for all the inputs individually are summed up to form one output channel.

In any case, the convolution operation can be written as a standard matrix product. The corresponding matrix would be sparse and have a distinctive structure with the same number of free parameters as the kernel. Other well-known types of structured matrices are Toeplitz, Vandermonde, or circulant matrices to name a few. As a general rule, structured matrices admit algorithms for the matrix-vector multiplication that are much cheaper than that of a general matrix product, so it reduces the computational cost of evaluating the network.

CNNs have been state-of-the-art for computer vision tasks, but they are also relevant for *time series forecasting* problems e.g. speech synthesis. In all those scenarios, the size of the input (pixels of images, sequence of words ...etc.) is much bigger than the extracted information. For this reason, they are called *encoders* or *discriminators* depending on the situation. On the contrary, some networks are used to generate more data than actually provided in input and are called *decoders* or *generators*. A widespread example of a decoder is the *Deconvolutional Neural Network* (DNN), which is used to generate spatial or temporal correlated sequences from compressed information, e.g. images or trajectories of robots. This size increase is obtained via dedicated linear operators. One option is the *transpose convolution* (Dumoulin & Visin, 2016). It gets its name from the fact that its sparse matrix representation is the transpose of the classical convolution with the same kernel. It can be implemented efficiently by noticing that the gradient of the matrix product is the transposed matrix, so that the transpose convolution is obtained by swapping the forward and backward passes. The transpose convolution is known to create checkerboard pattern artifacts in practice. Thus, it was mostly abandoned in favor

of the upsampling operator combined with the usual convolution. The upsampling operator simply consists in increasing the size of the input by repeating each element successively. Roughly speaking, *polling* that replaces a group of elements by its mean or max value can be seen as the reverse operation. In particular, DNNs have been used in chapter 5 to generate on-the-fly stable walking trajectories provided some desired high-level patient and gait features as input, e.g. such as the patient weight or the speed and step length.

Based on the success of CNNs, researchers have studied and proposed other types of neural networks based on weight matrices with different structures. It remains unclear whether other types of structured networks can be beneficial to other types of applications and which type of structure can provide both accuracy and efficient computation. The vulnerability of neural networks against adversarial attacks has raised concerns after the demonstration by Goodfellow et al. (2015) that it was fairly easy to fool image recognition networks based on CNNs with an imperceivable but carefully constructed noise in the input. Leveraging the properties of structured matrices to secure neural networks is promising. Robustness against adversarial attacks is even more critical when it comes to control policies for autonomous robots. Even without malicious intent, a faulty sensor may trigger similar unexpected behavior. It must be avoided at all costs to avoid injuring nearby people. Several methods to deal with adversarial attacks in policy learning are discussed in section 4.2.3.

Accelerating Deep Learning

Using the most appropriate data structure for a given problem is one way to accelerate calculations and save memory, and lowering the precision number format is another. The industry has moved from 64-bit precision to 16-bit precision formats by simply replying on batch normalization and gradient scaling, but going further is not straightforward. This issue is related to *quantization*, which consists in approximating a large set (eventually uncountable) by a countable smaller one, here, real numbers by bits. The current standard is the IEEE-754 floating-point arithmetic. It was designed in 1985 to be highly versatile and met the requirements of any application. Thus, this representation is far from optimal for machine learning applications where numbers are usually normalized, keeping most of them between -1 and 1. New representations of real numbers tailored for machine learning have emerged since then. They improve accuracy while retaining the same number of bits, thereby enabling using 8-bit precision formats not only for inference but also for training. Gustafson and Yonemoto (2017) has introduced a new data type called a posit designed as a drop-in replacement for IEEE-754 floating-point arithmetic that establishes the standard binary encoding of real numbers. Posits provide compelling advantages over floating-point arithmetic, including larger dynamic range, higher accuracy, better closure, bit-wise identical results across systems. In principle, the number of operations per second could be significantly higher using similar resources, but it is unlikely to happen as it would require new math pipeline hardware designs. Very recently, Micikevicius et al. (2022) have proposed another data type

that minimally deviates from the IEEE-754 standard. This ensures that software implementations can continue to rely on such IEEE-754 properties as the ability to compare and sort values using integer operations. Yet, supporting this new data type natively still requires adapting the existing math pipeline hardware designs.

One last option to speed up calculations is enhancing the basic scientific computing routines themselves. Typically, the complexity of the best known algorithms for computing the dense matrix product is still higher than the proven minimum. Beyond this, which algorithm to choose in practice strongly depends on the available hardware, the size of involved matrices, and the required numerical accuracy. For instance, the Strassen (Strassen, 1969) and Laderman (Laderman et al., 1992) algorithms are generally faster than the standard naive matrix multiplication algorithm for large matrices but slower for smaller matrices, while always being less accurate. Fawzi et al. (2022) used *Reinforcement Learning* (RL) to discover faster and provably correct algorithms for the multiplication of arbitrary matrices by training an agent to find tensor decompositions within a finite factor space. They showcase the applicability of this approach to structured matrix operations by recovering the state-of-the-art complexity for the skew-symmetric matrix-vector multiplication. Remarkably, they found algorithms tailored for specific hardware by optimizing the actual wall time.

3.2 The Reinforcement Learning Problem

RL is one of the three machine learning paradigms, alongside supervised and unsupervised learning (Sutton & Barto, 2018). It is concerned with how an intelligent agent should take action in an environment in order to perform a given task. Specifically, finding optimal decision sequences that benefit the agent over the long term, even if this requires taking undesirable actions in the short term. Ultimately, the goal is to reproduce the emergence of human-like cognitive skills. No training samples are readily available and the agent has to collect them via interaction with the environment, much like a young child does. The collected data are unlabeled as in unsupervised learning, but the agent enjoys a reward for each action. This reward is supposed to be used by the agent to sort actions: better actions are associated with larger rewards. Nevertheless, providing an informative reward is in no way necessary. It is possible to systematically return zero until task completion, but this makes learning very difficult. Contrary to supervised learning, it is an indirect metric of the optimality of the agent's behavior. It is not evaluated with respect to some ground truth that would be known in advance but rather must be discovered through maximization of the total cumulative reward termed *return*. In this way, RL fills the gap between supervised and unsupervised learning.

Fundamentally, it aims at solving the same problem as optimal control but the mindset and theoretical grounding are dramatically different. The theory of optimal control is concerned with the existence and characterization of optimal solutions and algorithms for their exact computation. Consequently, a white-box model of the system dynamics must be at one disposal. The optimal solution is computed through

gradient descent after the evaluation of the analytical gradient. On its side, RL is about leveraging data to improve upon the current behavior. To do so, it relies on trials and errors and stochastic process formalism to derive algorithms that converge to the optimal policy in expectation over the observed distribution of state. It only requires being able to interact with the environment to assess the current behavior of the agent. A black-box environment whose current state is partially observable is sufficient, and it could even be a real physical system. It is not necessary to be able to freely evaluate the system dynamics for arbitrary state-action pairs, nor to have access to the analytical gradient. RL has been around since the mid-1950s but has not received much attention until the so-called deep learning revolution that started about ten years ago. Due to its generality, reinforcement learning is studied in many disciplines, such as game theory, control theory, collective intelligence, and statistics. Applications range from training a bot to achieve high scores in a game to controlling a robot to complete physical tasks. One of the first practical demonstrations was playing Backgammon at master level (Tesau & Tesau, 1995).

3.2.1 The Agent-Environment Interaction Loop

Global Overview

RL is inherently discrete-time and sequential: the agent starts in an initial state s_0 drawn from some distribution ρ_0 that is supposed to be unknown. At every timestep, the agent takes an action a_t based on an observation o_t of the current state of the world s_t and a reward r_t if any. Then, time passes until the next timestep $t+1$, and the action a_t takes effect on the world in the meantime. A supervisor is responsible for sending a new observation o_{t+1} and reward r_t according to all the previous events from the start and the updated state s_{t+1}. The set of events that occurred between two successive timesteps is called transition step e_t and is represented by the tuple (s_t, a_t, r_t, s_{t+1}) aggregating all the information available. An ordered but possibly truncated sequence of transition steps $(e_t)_t$ is called a trajectory and denoted τ. Finally, it ends at time T when the time limit is reached or a termination condition is triggered, e.g. the task has been performed successfully or a critical failure has occurred. An episode refers to the unfolding of events from start to end. It is characterized by the complete trajectory $\{e_t\}_{t=0}^{T}$. At this point, the world is reset, a new initial state is sampled, and it repeats all over again. The behavior of the agent is defined by a policy π. In general, this policy maps the current observation o_t and the past reward r_t to the next action a_{t+1}. This process is summarized in figure 3.7.

Typically, the practitioner has no control over the evolution of the world following the actions of the agent. It obeys some rules or dynamics equations whether it is a discrete or continuous-time system. Those are pre-defined and cannot be changed.

The supervisor is doing the interface between the environment and the agent. It sends to the agent an observation of the world and a reward. The observation comprises information about the agent itself and its surrounding environment. In robotics, they are called proprioceptive and exteroceptive data respectively. Tradi-

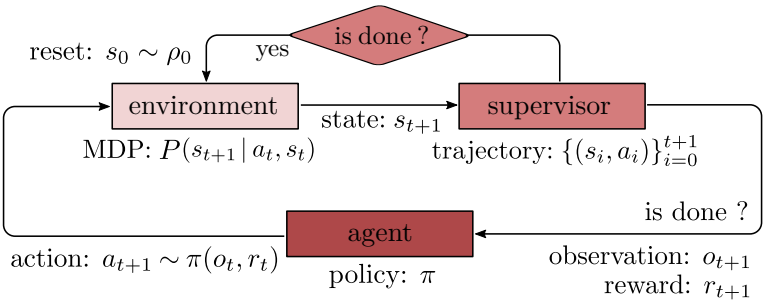


Figure 3.7: Agent-environment interaction loop in Reinforcement Learning. The color intensity of the blocks indicates how much freedom is left to the practitioner to design them.

tionally, the observation only concerns the current state, but it may gather a history of past events. It can have any data structure and contain heterogeneous information, e.g. a binary occupancy grid together with integer raw encoder measurements and the continuous tilt angle estimation. These data are sometimes pre-processed algorithm to extract features using unsupervised learning before being forwarded to the agent, most notably when they include pixel arrays of camera recordings. As for the reward, it is expected to be a single scalar under the classical RL formalism, although this assumption may be dropped by considering *Pareto optimality* (Censor, 1977) instead of strict maximization. It is the responsibility of the practitioner to design the supervisor in a way to guarantee that the policy maximizing the return is really performing the task optimally.

The behavior of the agent is defined by a stochastic policy mapping the current observations o_t to a probability distribution $\pi(\cdot|o_t)$ over the action space $\mathcal{A}$, which can be viewed as a function itself. To be more specific, $\pi(a_t|o_t) = \mathbb{P}(a = a_t|o = o_t)$. In robotics, it is referred to as *control policy* since it is the actual controller of the system. The design of the policy is completely free. In deep RL, the policy is a neural network whose parameters are adjusted during training to achieve optimal behavior.

Discretization of Continuous-Time Systems

For continuous-time systems, doing one transition step implies integrating the equations of motion over a given timestep while holding constant the latest action of the agent. The timestep is usually fixed because there is limited interest in doing otherwise, but it is not strictly required.

If the timestep is very small, then only the integral (or moving average) of the action would affect the state, high-frequency variations being naturally filtered by physical systems. Although this phenomenon is not an issue for traditional control methods, it may significantly impede the performance of RL algorithms for several reasons. First, it makes exploration challenging. The latter is an essential component of RL algorithms which relies on the stochasticity of the actions taken by the agent to obtain informative data (cf. section 3.2.2). If the underlying random process

is not temporally correlated, then its effect would have a single timestep to build up. This would severely hinder exploration and cancel it out entirely at the limit. Apart from that, the difference between consecutive states may not be statistically significant anymore due to the inherent stochasticity of the process or the noise in the observation. Such transition steps bring no information as it is impossible to assess the impact of the actions. Consequently, the agent would be unable to learn anything using RL algorithms considering the transition steps individually. Before getting there, it is essential to increase the size of the training batches when reducing the timesteps to preserve the sample diversity. Indeed, the observed distribution of transition steps must be representative of the closed-loop dynamics of the system under a given policy for associated stochastic estimates to be unbiased.

In this perspective, it seems better to set the timestep as large as possible to maximize the signal-to-noise ratio and thereby the information provided by the individual transition steps. However, the sensitivity of the next state increases with the timestep: a small alteration of the current state or action may radically change the next state after integration. This issue is well-known and studied as it is also pathological to shooting methods in numerical analysis. Because of this, the transitions may appear utterly random, and it would not be possible to learn anything. Before such extreme consequences, increasing the timestep limits the optimal performance that can be expected. This applies not only to policies trained using RL but also to optimal control methods. This effect is marginal at first. In most robotic applications, the action gathers some high-level features driving a traditional *Low-Level Controller* (LLC) that, in turn, updates the motor torques accordingly multiple times during a single transition step. Thus, not being able to update the action does not completely prevent from reacting to unexpected events or disturbances. Yet, the agent can only adapt its behavior with a delay, so the system may become completely uncontrollable at some point depending on the actual LLC. This issue is related to the famous *bang-bang control* problem in classical control theory (Bellman et al., 1956): arbitrary high performance can be achieved with a controller having only two possible actions available (on-off) if the update frequency is high enough.

In the end, there is an optimal trade-off that is problem-specific. When learning policies for legged robots, we observe that it can be as large as 40ms even for a basic LLC such as *Proportional-Integral-Derivative controller* (PID) without noticeably impeding the performance. It must be closer to 5ms for direct torque control, but it is already larger than what is usually required by traditional model-based approaches.

Offline vs. Online Reinforcement Learning

During training, the actual environment may be replaced by a synthetic model that may be unknown from the agent's perspective. If so, it is referred to as *offline RL*, otherwise *online RL*. Offline RL is often necessary as interacting with the real environment may be time-consuming, costly, or put the agent and the environment at risk of physical damage. However, it introduces the additional burden of being able to transfer the policy from simulation to reality. This *sim-to-real transfer* is usually

challenging because of the so-called *reality gap* (Jakobi et al., 1995). If there are discrepancies between the simulated and real environment, the encountered distribution of states is likely to drift over time relative to the expected one. This particular issue is known as *distributional shift* (Yu et al., 2020). If nothing is done during training to prevent this drift from occurring, then the expected return being optimized would be biased and the agent is set to underperform in reality. The agent may finally reach a state at some point that has never occurred in simulation, leading to catastrophic failure. (Zhao et al., 2020) wrote a comprehensive survey reviewing the main issues caused by the reality gap and the state-of-the-art methods to ease transfer to reality.

Nonetheless, offline RL may still be preferable over online RL as it enables the agent to have access to *privileged information* that would be impossible to provide otherwise (Vapnik & Vashist, 2009). It can bring critical information about the environment that makes it substantially easier to perform the task. For instance, it could be the exact map of the environment including its topography for locomotion tasks on legged robots. It implies an extra learning step after training the policy in order to get rid of the privileged information before transfer to reality as it cannot be measured nor estimated experimentally by definition. Training one policy to mimic another one (eventually using different inputs) is an example of *policy distillation* (Rusu et al., 2016) and is characterized by the *Teacher-Student* learning framework (Li et al., 2014). The classic approach for tackling such a problem is *Imitation Learning* (IL), which focuses on training the student from expert demonstrations provided by the teacher (Hussein et al., 2017). The simplest form of IL is *Behavior Cloning* (BC) (Bain & Sammut, 1999; Ross & Bagnell, 2010), which is a supervised learning algorithm. First, trajectories are collected from the teacher. Then, the corresponding state-action pairs are treated as independent training samples and the student is trained to minimize the empirical prediction error. These two steps are repeated iteratively until convergence. It has been applied to self-driving cars (Bojarski et al., 2016) and quadrupedal locomotion (Miki et al., 2022). See section 4.1.2 for details.

Dense vs. Sparse Reward

The supervisor usually provides a dense informative reward at every timestep. The latter is supposed to guide the agent toward a policy that is successfully achieving the task by providing insightful feedback about the optimality of the current policy. In this way, the agent can continuously and gradually improve its behavior, and therefore it is possible to train it using fairly simple algorithms. This kind of reward is said to be *dense*. It is task-specific and must be carefully tailored to incorporate domain knowledge. Designing a reward to improve convergence, speed-up training, or enforce distinct features for the policy is known as *reward engineering* (Dewey, 2014). However, domain knowledge induces some biases toward particular behaviors as it is nothing more than preconceptions regarding how the task should be done. It may prevent the emergence of effective but unanticipated strategies, which is detrimental overall. Besides, incorporating domain knowledge in the reward is time-consuming and requires problem-specific expertise, which means that it is impossible to target

a wide variety of problems simultaneously.

Conversely, a *sparse* reward does not carry any information except when some specific event of low probability occurs. A classical example of sparse reward is returning -1 for failure, 1 for success, and 0 otherwise. It has the advantage to be extremely generic and straightforward to implement without expert knowledge. Nevertheless, it requires more sophisticated training methods to leverage such sparse information. At initialization, the agent must discover how to solve the task at least once to bootstrap learning. The naive strategy is the *random walk* (Pearson, 1905) since the reward is not informative enough to serve as a guide. Yet, many problems are too complicated to be solved that way, and the agent would never start learning anything. Curiosity-driven exploration and curriculum learning (cf. section 3.2.2) are some of the approaches that can be leveraged to overcome this issue.

Infinite-Horizon Markov Decision Process Formulation

The entire agent-environment interaction loop can be framed as infinite-horizon *Markov Decision Process* (MDP), which is a discrete-time stochastic control process (Bellman, 1957). It is an extension of the Markov chain, the difference being the addition of actions to allow choices and rewards to give incentives. Mathematically, a MDP is an 8-tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{O}, \mathcal{A}, P, R, O, \rho_0, \gamma\}$, where $\mathcal{S}, \mathcal{O}, \mathcal{A}$ are the state, observation and action spaces respectively, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, $O : \mathcal{S} \to \mathbb{R}$ is the observation function, ρ_0 is the initial state distribution, and $\gamma \in [0, 1]$ is the discounting factor for future rewards. In principle, the agent only has access to the observation o_t while the transition probability function depends on the state s_t. It is assumed in the following that the state is fully observable and no further distinction will be made between the state and the observation. This simplification is common in the literature as it makes everything clearer without invalidating the reasoning.

The reward function associates an instantaneous score to each transition step. As presented in the next section, it is actually its cumulative sum over complete episodes that is being maximized. Therefore, it is not easy to make sense of the reward function without considering trajectories much longer than the time constant of the problem. Still, this score is intended to assess how promising or profitable the current state is, in light of the previous state, the action taken, and the task at hand, but ignoring any older event. The reward function is a deterministic function of the transition step that can be written as follows,

$$r_t = R(s_t, a_t, s_{t+1}). \tag{3.18}$$

Providing the two consecutive states plus the action is critical for the scoring of a transition step because it enables gauging the efficiency of an action by its effect.

The reward function in RL is the counterpart of the running cost in *Optimal Control Problems* (OCPs). It plays a more important role in RL which lacks equality and inequality constraints in most cases, unlike optimal control. Enforcing hard constraints is challenging since the transition function is unknown to the agent and

the action is a random variable. In particular, any constraint involving the transition function on continuous state spaces is at best guaranteed to be satisfied up to some confidence level. To date, few authors have investigated this question. It is at the heart of our contribution in RL and is discussed thoroughly in sections 4.2.2 and 6.2.

The transition probability function characterizes the agent in the environment. It records the probability of transitioning from a given state s to another one s' after taking action a. It yields,

$$P(s'|s,a) = \mathbb{P}(s_{t+1} = s'|s_t = s, a_t = a). \tag{3.19}$$

The realization of a transition from the current state s_t to the next one s_{t+1} under the effect of action a_t is known as a transition step and is represented by the 3-tuple $\{s_t, a_t, s_{t+1}\}$. The MDP is assumed to be *stationary*, which means that the whole problem is time-invariant. This includes the distribution ρ_0 of the initial state and the transition function that determines how the world changes under the actions of the agent but also the reward function prescribed by the supervisor. Notably, this hypothesis does not hold when *curriculum learning* is applied to gradually increase the task difficulty over training iterations (cf. section 3.2.2), e.g. the magnitude of external forces for robust locomotion. Intuitively, it changes the observed state distribution and subsequently the optimal policy. This issue can be neglected as long as the updates of the environment happen at a much slower rate than the policy itself to approximately decouple both phenomena.

In RL, it is assumed that the world is unknown. The agent and the learning algorithm cannot enquire about the true transition probability function P but can try to estimate it. It is the main difference between the classical *Dynamic Programming* (DP) and RL algorithms (Van Otterlo & Wiering, 2012). It enables targeting every large MDPs where exact methods become infeasible. The task at hand is supposed to be *episodic*, which means that there is at least one termination state and the average length of a complete episode is bounded, as opposed to *continuing* tasks.

All states in MDPs have Markov property, namely, the future only depends on the current state, not the history. In other words, the future and the past are conditionally independent given the present, as the current state already encapsulates all the information possibly available to predict the future:

$$\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_0, s_1, \ldots, s_t) \tag{3.20}$$

In essence, physics laws are local in time, thus this simplifying assumption in RL is generally fairly accurate for real applications. One could imagine a system for which the next state depends on k consecutive states. If so, then it is sufficient to redefine the state as $s'_t = (s_{t-2}, s_{t-1}, s_t)$ to obey the Markov property.

This condition is necessary to prove the convergence of learning algorithms to the optimal policy but has limited practical implications. It often partially holds in practice, but most algorithms are robust enough to find a policy achieving the desired task successfully anyway. The MDP property is true for continuous-time autonomous systems if and only if it is *fully observable*. Any high-order differential equation can

be rewritten as first-order by augmenting the state with as many derivatives as its original order minus one. This augmented state would have Markov property, or equivalently, a history of the past whose length is equal to the order after discretization of the differential equation. Thus stacking a few past observations may help to get closer to full observability in some cases, but the impact has not been studied thoroughly. Although it should strictly improve the performance of the policy theoretically, it rarely happens it makes training harder. Another approach would be to add memory to the agent so that it can remember past events without having to provide them explicitly, but it suffers from similar training issues.

Originally, before the deep learning revolution, RL was mostly tabular and the optimal action was selected as the one leading to the highest future return after the enumeration of all possible choices. Learning the optimal policy mostly consisted in assessing the optimal action in every possible state. Thus, the state and action spaces were finite discrete sets. It is typically the case for classical board games such as Backgammon, Go, and Chess to name a few. In the control field, the true action space is commonly continuous. Regardless, the action was historically limited finite set of values. For continuous-time systems, it does not significantly limit the performance as long as the timestep is small enough. As mentioned in section 3.2.1, only the integral of the actions affects the state, which can be regulated more accurately than the actions themselves. In many problems, the true state space is also continuous, but its quantization is hardly an option as opposed to the action. Its quantization would translate to rounding errors in the observation. It is then impossible for the agent to distinguish states associated with the same value. Whether it can be tolerated highly depends on the problem. As a rule of thumb, it gets exponentially difficult to control the system as the quantization step size increases. More advanced partitioning techniques have been proposed but are only tractable for low-dimensional spaces, e.g. tile-coding (Sutton & Barto, 2018). For this reason, it is generally supposed that the action space is a finite discrete set but not the state space. No such restriction will be enforced in the following unless stated otherwise.

Stochastic World Model

Even if the agent dynamics is theoretically deterministic, assuming the world is stochastic and modelled by a transition probability function $P(s'|s,a)$ is preferable (cf. section 3.2.1). If it appears that it is truly deterministic, then the transitions can be represented with the Dirac function for any state, but it is rarely the case anyway. First, the world dynamics may be randomized on purpose to improve robustness and facilitate sim-to-real transfer. In robotics, one way is to sample different physical properties of the agent at every episode, e.g. the mass distribution for poly-articulated robots. Similarly, the properties of the environment itself could change, e.g. the ground friction coefficient. This would inevitably affect the transition function which models the whole agent-environment interaction and not solely the agent. This technique is known as *domain randomization* and is presented in section 4.1.3. Secondly, the environment is considered stochastic as soon as unexpected events may

disturb the agent, which is often the case. If it happens, the future state for two identical state-action pairs would be different, and therefore the transition function would be stochastic. Next, the dynamics may be stiff, and two undistinguishable state-action pairs may be associated with statistically different next states. Typical examples are collisions and foot slippage in legged robotics: it will slip as soon as it loses contact even if it is only one nanometer above the ground, which may be tricky to predict accurately from the state. A stochastic model would be able to take advantage of such sensitivity. More specifically, it can be interpreted as a smooth approximation of the true transition function. Finally, it is impossible to define a state that really encapsulates every bit of information that may influence the future. Consequently, even if the laws of physics are deterministic at a macroscopic scale, it would appear stochastic since two identical state-action pairs would result in a different next state. In this case, the world is said to be *partially observable* as part of the information necessary to model it is missing. This issue is even more significant given that the state is not directly accessible to the agent but only the observation provided by the supervisor. The observation is distinct from the state and may contain less information, be noisy, or have a delay. Nevertheless, the agent has to estimate the transition function or at least take action based on the observation exclusively.

3.2.2 Key Concepts and Terminology

Expected Return Maximization

The goal is to find the policy π^* that maximizes the discounted cumulative reward in expectation over the distribution of trajectories τ induced by the policy. Let us assume the length T of the episodes is fixed for simplicity. The return for given T-steps trajectory $\tau = \{(s_i, a_i)\}_{i=0}^{T}$ is abusively denoted $R(\tau)$. It gives,

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t, \tag{3.21}$$

where $\gamma \geq 0$ is called the discount factor. This expression generalizes to variable length trajectories without hassle. In the case of infinite-horizon MDPs, the discount factor γ must be strictly lower than one to guarantee that the return is bounded. This condition is necessary to prove of convergence of most learning algorithms.

The expected return for a given policy $J(\pi)$ is computed as follows

$$J(\pi) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)] = \int_{\tau} \mathbb{P}(\tau|\pi) R(\tau)\, \mathrm{d}\tau, \tag{3.22}$$

where $\tau \sim \pi$ refers to the distribution of trajectories under the current policy π. More precisely, the probability of a T-steps trajectory τ is

$$\mathbb{P}(\tau|\pi) = \int_{s_0\in\mathcal{S}} \left\{ \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \right\} \mathrm{d}s_0. \tag{3.23}$$

Finally, the optimization problem in RL is formulated as

$$\pi^* = \arg\max_{\pi} J(\pi). \tag{3.24}$$

The discount factor γ is a hyperparameter that must be tuned carefully. It must not be underestimated as it may affect the optimal solution dramatically, especially for dense reward functions. If the discount factor is small, then the effect of future reward does not matter much in the computation of the return, so that the agent only looks for quick short-term profit without anticipating much in the future. For instance, in economics, it would be more profitable in the short term to rent an apartment than to buy it if the interest rate is high regardless of the prices, while it would be more beneficial in the long run to buy now if the prices are currently low. On the contrary, if the discount factor is very high, then every single point in time is taken into consideration when choosing the next action. At the limit, it does not matter if the event is going to happen right after or in a very long time. It seems appealing at first, as it promotes a truly optimal policy that takes informed actions based on present and future events. However, it is only about probabilities as it is impossible to predict exactly the future, and anyway, the accuracy of such forecasting drops exponentially with the horizon of time. It creates virtual barriers and limits the agent's actions based on prophecies that are not even going to happen, limiting the overall performance. It is risky to tune the discount factor accordingly to the confidence in the probability of future events. Is suffering your whole life to avoid dying in an accident really the dream everyone should pursue? The discount factor is directly related to the horizon of time that will be taken into account for planning the next action: the instantaneous reward that will be obtained more than $\log(0.99)/\log(\gamma)$ steps ahead accounts for less than 1% of the current one. This horizon must be consistent with the time it takes for the effect of an action to completely vanish for a given task. It's not about being reluctant to the unexpected, but rather about providing the opportunity to plan for long-term profits. For push recovery, it makes sense to consider a horizon of 2s, which is enough to do a few steps to avoid falling (cf. section 6.6). Anything longer would encourage the agent to be more cautious and adopt a resting posture that is uncomfortable to the patient but makes it easier to keep balance in the prospect of unexpected events. On the contrary, if the horizon is shorter than the time it takes to fall, it may be more profitable to do nothing and keep falling as long as it does not penalize the instantaneous reward.

Value Functions and Optimality Conditions

The *Value function* measures how promising a state s is in terms of future return. The future return, also called *reward-to-go*, is denoted G_t or $R_t(\tau)$ to point out the trajectory dependency. It is defined as the return that would be obtained if starting counting from time t in state s, i.e. $G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$ for a given trajectory $\tau = \{(s_i, a_i, r_i)\}_{i=0}^{T}$. For infinite-horizon MDPs, the current time cannot be distinguished from any other, so it is the same as is equal to the expected return as

if the trajectory started in that state and the agent acted according to a given policy π forever after. It is sometimes called *state-value* to highlight that it is conditioned by the starting state s, and denoted $V_\pi(s)$. Let $\tau_{t:}, \tau_{:t}$ denote the sequence of transition steps on a trajectory starting after and finishing at time t respectively. It yields,

$$V_\pi(s) = \mathop{\mathbb{E}}_{\tau_{t:}\sim\pi}[R_t(\tau)|s_t = s] = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s]\,. \tag{3.25}$$

Alternatively, it may be interesting to compute the value assuming the initial action a can be arbitrary and does not have to come from the policy. In this case, it is rather called the *action-value* or *Q-value* function and denoted $Q_\pi(s, a)$. It gives,

$$Q_\pi(s, a) = \mathop{\mathbb{E}}_{\tau_{t:}\sim\pi}[R_t(\tau)|s_t = s, a_t = a] = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]\,. \tag{3.26}$$

Then, the action-value is related to the state-value function as follows,

$$V_\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}[Q_\pi(s, a)]\,. \tag{3.27}$$

One can assess the optimality of a policy π at state s by comparing the state-value and action-value functions. It is known as the *advantage* and denoted $A_\pi(s, a)$,

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s). \tag{3.28}$$

If the advantage is positive, then it is better to take action a rather than follow the policy π, it is the contrary otherwise. If the policy is optimal, then the advantage is always negative whatever the state and action.

Most RL algorithms leverage either the value function, the Q-value, or the advantage to improve the policy. Knowing the initial state s_0, the optimal policy maximizes the state-value functions for that state,

$$\pi^* = \arg\max_\pi V_\pi(s_0). \tag{3.29}$$

As before, the optimal state-value function $V^*(s)$ is related to the optimal action-value function $Q^*(s, a)$,

$$V^*(s) = \max_{a\in\mathcal{A}} Q^*(s, a) \tag{3.30}$$

If the optimal action-value function $Q^*(s, a)$ is available, then the optimal policy can be defined as a by-product. The action a that maximizes it is greedily selected for any given state s,

$$\pi^*(s) = \arg\max_{a\in\mathcal{A}} Q^*(s, a). \tag{3.31}$$

Bellman Equations

A Bellman equation is a necessary condition for optimality associated with discrete-time optimization problems. It derives from DP and therefore is central in the RL formulation. Its continuous-time counterpart is the Hamilton-Jacobi-Bellman equation that is presented in appendix A.3 as a method for solving OCP applied to

trajectory planning. It writes the value of a policy at a given time in terms of the return from some initial actions and the value for the remaining decision process. This breaks the optimization problem into a sequence of simpler sub-problems, as Bellman's *principle of optimality* prescribes. In particular, the value functions are decomposed into the immediate reward plus the discounted future value,

$$V_\pi(s) = \mathop{\mathbb{E}}_{\substack{s'\sim P\\ a\sim\pi}} \left[R(s,a,s') + \gamma V_\pi(s')\right] \tag{3.32}$$

$$Q_\pi(s,a) = \mathop{\mathbb{E}}_{s'\sim P} \left[R(s,a,s') + \gamma V_\pi(s')\right]. \tag{3.33}$$

The Bellman equations for the optimal value functions are obtained by replacing the expectation of the action over the policy with the one leading to the highest value,

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s'\sim P} \left[R(s,a,s') + \gamma V^*(s')\right]. \tag{3.34}$$

If the world is known analytically, then it enables computing the value at the current state s knowing the one at the future state s' and the reward obtained by following the policy π. If the world is known analytically, then this turns into a OCP solvable by DP. In most scenarios, we do not know the transition or the reward function analytically, so it cannot be solved directly by applying Bellman equations. Nonetheless, it lays the theoretical foundation in RL.

The Bellman equations enable estimating the value functions from incomplete episodes without having to track them up to termination. The episodes do not have to terminate at some point anymore, which allows for solving continuing tasks. The key idea is *bootstrapping*: the current estimate of a value function takes into account its previous estimate rather than exclusively relying on collected data. The most basic algorithm leveraging this capability is *Temporal-Difference* (TD) learning. The Bellman equations introduced here are said to be one-step look-ahead. Looking more steps ahead would give more weight to the collected data over the previous estimate. Formally, it reduces the bias due to the previous estimate being off, at the cost of increasing the variance. See appendix E.3 for details.

At every iteration, a training batch of fixed size is collected, and a value function estimate is computed accordingly. For this estimate to be unbiased, the distribution of trajectories must be representative of the closed-loop dynamics. For instance, if the agent may end up in a periodic cycle and keep collecting similar samples over and over again, then the training batch would degenerate and provide misleading information to the agent. This issue is commonly mitigated by truncating the episodes to a maximum duration called time limit, despite the horizon of the MDP being theoretically infinite (Pardo et al., 2018). This would likely affect the distribution of trajectories and thereby the optimal policy (see equation (3.23)). Intuitively, this limit must be short often to ensure sufficient sample diversity, but not too short to give enough time for long-term phenomena to build up. There is only one scenario where such bias does not occur: training an agent for solving a continuous task where the initial state distribution matches the stationary one (cf. appendix E.5). This side-effect has

been largely ignored in the literature because the time limit is usually much larger than the equivalent cumulative reward horizon specified by the discount factor.

Policy and Value Function Parameterization

The *target policy* must be distinguished from the *behavior policy*. The former is the one being trained while the latter is used to collect data. They are different or identical depending on the type of algorithm (cf. section 3.2.3). The behavior policy must be stochastic for the sake of exploration (cf. section 3.2.2), but there is no such restriction for the target policy. For continuous control problems, the target policy is necessarily a function approximation parametrized by θ (cf. appendix E). The preferred function approximations in deep learning are ANNs, though *Gaussian Mixture Models* (GMMs) are sometimes used, especially for online RL and IL (Pignat & Calinon, 2019), because of their high expressiveness relative to their number of parameters. If the target policy is deterministic, then it is denoted μ_θ instead of π_θ and maps the current observation o_t to the action a_t that the agent ought to take.

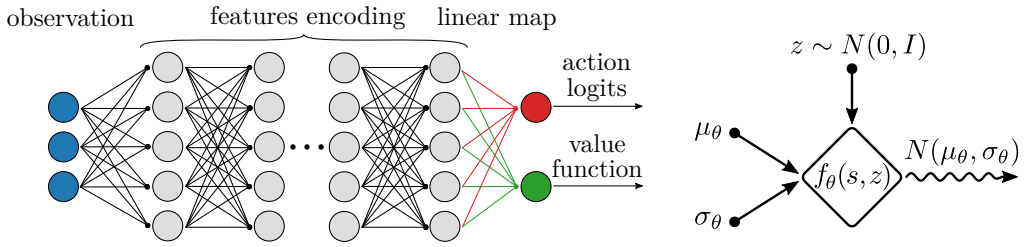
It gets more complicated If the target policy is stochastic: the network outputs values characterizing a given action distribution. Originally, the action space was exclusively finite, and thereby the action distribution was *categorical*. It is fully specified by the probability of each possible action. Since probabilities must be positive, the networks must predict their logarithm instead, hence the name *logits*. Indeed, it is extremely difficult to enforce positivity constraint to the output of a ANN, and clamping its output is not an option as it would break backpropagation. The probabilities are recovered using Softmax activation function for the output layer,

$$\mathbb{P}(a = a_i) = \frac{e^{x_j}}{\sum_{j=0}^{|\mathcal{A}|} e^{x_j}}, \forall a_i \in \mathcal{A} \tag{3.35}$$

In continuous control, the action distribution is mostly *isotropic multivariate Gaussian*. Isotropic means the components of the action space are *independent and identically distributed* (iid) random variables. It implies that the covariance matrix Σ is diagonal, i.e. $\Sigma = \text{diag}(\sigma)$. This distribution is characterized by its mean μ_θ and standard deviation σ_θ. Those parameters are frequently called logits improperly. The standard deviation must be positive. Following the same reasoning as before, the network predicts its logarithms. Let $z \sim N(0, I)$ be a vector of noise drawn from the standard isotropic Gaussian. An action sample at state s can be computed with,

$$a = \mu_\theta(s) + \sigma_\theta(s) \odot z. \tag{3.36}$$

This formula is commonly known as the *reparametrization trick* (Kingma & Welling, 2014), also stated as $\mathbb{E}_{a\sim N(\mu,\sigma)}[f(a)] = \mathbb{E}_{z\sim N(0,1)}[f(\mu + \sigma \odot z)]$. More generally, a policy is said to be *reparameterizable* if it is possible to write it as $a \sim f_\theta(s, z)$, where z is a random noise drawn from some independent distribution ξ. It is true for any continuous distribution since all of them can be derived from the standard uniform distribution, including the normal law itself. Nevertheless, it requires the inverse



(a) Actor-Critic network with shared features (b) Normal reparametrization

Figure 3.8: Classic policy and state-value network architecture: single MLP with up to three hidden layers of variable width ranging from 64 to 256 units. The action is normally distributed, hence the action logits are the mean and standard deviation. The standard deviation is often a free parameter instead of an output of the network.

cumulative distribution function to admit a closed-form to be practical, which is not necessarily the case. It follows $\pi(a|s) = \mathbb{E}_{z\sim\xi}[\mathbb{1}_{\{a\}}(f_\theta(s, z))]$, where the integrand is the probability of action a conditioned by (s, z). It is illustrated in figure 3.8b.

The classical network architecture is a basic MLP because it is easy to train, but it could be a *Long Short-Term Memory* (LSTM) if the state is partially observable. In many training algorithms, the state-value function is jointly learned with the policy. Typically, the policy and the value networks share all hidden layers. In RL, it is a common interpretation that once appropriate features have been extracted, a linear mapping is enough due to the feedback loop. It is consistent with traditional feedback controllers, for which the action is proportional to the error in a given feature space. As a result, the hidden layers are responsible for extracting features and the final layer learns the linear mapping. Interesting, sharing layers also reduces the computational cost and memory footprint. The complete architecture is shown in figure 3.8a.

Curriculum learning

Curriculum learning (Narvekar et al., 2020) derives from the intuition that the difficulty of the task must be adapted to the current capability of the agent for it to learn efficiently. Translating this simple idea into an actual training method poses two questions: how to evaluate the capability of the agent? How to select the most appropriate task based on it?

Vanilla curriculum learning that was introduced first about ten years ago dismisses these interrogations altogether by relying exclusively on a priori expert knowledge (Bengio et al., 2009). The difficulty is monotonically increased over iterations according to a fixed schedule. It is based on the bold assumption that the capability of the agent increases over iterations whatever happens, so that estimating the suitable difficulty of the task is roughly the same as scheduling over iterations if tuned properly. Then, the mapping from the desired difficulty to a task is manually specified. Although this approach has proven successful in practice, it has major flaws. First, if the learning curve is highly non-linear and sensitive, then it is impossible to

anticipate when it is going to start learning anything, and scheduling over time is unreliable. It is typically the case for locomotion tasks in bipedal robotics. At first, the robot is not even able to stand, and it is hard to predict when it is going to be the case. Secondly, the notion of difficulty of the task may be hard to define intrinsically for complex problems as it may combine multiple aspects or depend on the agent itself. For instance, both the ground profile and the external forces contribute to the difficulty of locomotion tasks, and it is difficult to unify them. Moreover, one of these components may be easier for the agent to tackle that the other, making it even more challenging to balance them over time.

Several more advanced curriculum learning methods have been proposed to overcome these shortcomings. The pivotal idea of self-paced and Teacher-Student curriculum learning is to estimate the difficulty of the task from the perspective of the agent. The probability of sampling individual tasks is a function of the capability of the agent in solving them according to a given performance metric, while maintaining enough diversity in the resulting distribution. It mitigates the two aforementioned limitations at once. Indeed, the difficulty is now intrinsically defined, so it implicitly combines multiple criteria and relates to the actual agent's potential. Besides, It is usually easy to determine a good performance metric based on the reward. However, it introduces a coupling between the increase in difficulty of the tasks and the capability of the agent that may create self-excitatory oscillations and prevent monotonic improvement. It would impede convergence and lead to a plateau effect in terms of the performance of the policy. This issue can be avoided by decoupling their respective dynamics by varying the difficulty slowly enough for the agent to catch up with the increase instead of having to lower it again. The easiest way to achieve this is to filter the performance metric with a moving average filter over training iterations. Curriculum learning is also suitable for multitasking, where a policy is trained to solve a heterogeneous set of problems simultaneously. The most difficult task can be presented more often to make sure the agent is equally capable of solving all of them.

Hindsight Experience Replay (HER) is a method that can be viewed as a kind of automatic curriculum learning (Andrychowicz et al., 2017). The core idea is to randomly select the task for the current episode among all the outcomes that have been obtained in the previous episodes. This way, the task to achieve is likely to match the current capability of the agent. This intermediary task is a moving target contrary to the actual task. As such, it is termed a goal to highlight this distinction. The agent will be aware of the current goal as part of the observation. Thus, it just has to learn to reproduce what it has already done before, but this time on purpose. HER is especially well suited for sparse reward problems since the previous episodes act as expert demonstrations guiding exploration. The agent will progressively gain knowledge about the transition probability function of the MDP and extrapolate what to do from the skills already required. For instance, when shooting a ball, being able to hit more and more specific locations would be helpful to finally learn to score. It has been demonstrated on manipulation tasks with a virtual robotic arm. Surprisingly, the performance of the policy at runtime is even better when providing a sparse reward instead of a dense one. Nevertheless, this method is only applicable

if it is possible to define intermediary goals that relate to the actual task whatever the outcome of an episode. In the case of push recovery, the task is to avoid falling. Defining goals that would help the agent learn to keep balance is not straightforward.

Curriculum learning is not always applicable as the appropriate behavior may change dramatically even if the difficulty increases gradually. It is typically the case for emergency push recovery: if the pushes are light enough, then controlling the posture without moving the feet is sufficient to withstand them, while it is necessary to do steps as soon as their strength exceeds some threshold. This issue is related to the notions of *deceptive objective function* presented in section 3.2.2.

Exploration-Exploitation Dilemma

The objective for the agent is to maximize the expected return. Exhaustive search methods would require sampling all possible trajectories for every candidate policy, which is untractable. Alternatively, one could imagine using function approximations and the gradient descent method either to train the policy directly or an estimate of the value function. It partially addresses the issue since it is no longer necessary to review all candidate policies, but it does not prevent computing at least the expectation over all possible states extensively. In practice, only a finite set of trajectories is computed at every iteration, from which how to improve the current policy must be used to infer. For this process to be efficient, it is necessary to try different behaviors in order to discover more promising state-action pairs than what would be obtained according to the current policy. It is true, no matter if the reward is dense or sparse. Although exploration is critical and necessary, it also harms performance to a certain extent. The policy has to deviate from the best strategy found so far in the hope of coming upon a better one. This issue is known as the *exploration-exploitation dilemma*. In a pathological scenario, it may prevent finding any valid solution at all if the agent explores too much, especially if a specific sequence of actions is required to complete the task. Exploitation refers to sticking to the expected best action based on already accumulated data without giving other actions a try along the path. The additional information that would be collected this way to improve the current knowledge is entirely dependent on the intrinsic stochasticity of the world, which is very limited in many cases. Intuitively, it seems important to explore at the beginning of the training to collect a lot of data regarding the state-action space, then gradually decrease it to focus on exploitation at the end. The goal is to find the best solution as fast as possible, but committing to solutions too quickly without enough exploration is undesirable as it is likely to end up in a bad local minimum or even fail. How to maximize the efficiency of exploration is still an open question.

The most common form of exploration relies exclusively on the stochasticity of the behavior policy. The ϵ-greedy is the first strategy that was proposed (Watkins, 1989). A parameter $\epsilon \in [0, 1]$ is controlling the amount of exploration vs. exploitation. With probability $1-\epsilon$, exploitation is chosen and the agent chooses greedily the action that it believes has the best long-term effect. Otherwise, the action is sampled randomly. Tokic (2010) adapt the parameter ϵ based on the agent's uncertainty about the world.

This approach effectively increases the diversity of the trajectories by completely randomizing the behavior at a few timesteps instead of slightly disturbing it all along the trajectory, so it only explores through large jumps and never locally. It may be fine if the action space is discrete but not if it is continuous. For example, this strategy is problematic in robotics where the action commands the motors. First, it would produce loud noise and vibrations during online learning, which may damage the device. Secondly, it would fail for the task requiring high precision.

It seems more appropriate to use a continuous random process with a relatively small standard deviation as behavior policy. Every action would be selected randomly but very localized around a mean value that depends on the current observation. In most cases, the standard deviation is constant or scheduled over time. This approach is double-edged: it is very robust but suboptimal. It does not allow for adjusting the exploration strategy based on the current state. Yet, it is not clear how to learn a function approximation of the optimal standard deviation since the expected return alone does not provide any insight into it. Thus, the objective function must be augmented one way or the other. *Entropy* regularization is commonly used, and in particular it is central in *Soft Actor Critic* (SAC) (Haarnoja et al., 2018). The entropy is defined as the expected negative log-likelihood. Loosely speaking, it measures the expected amount of disorder in the exploration strategy of the agent. Let x be a random variable with probability distribution P. Then,

$$H(P) = \mathop{\mathbb{E}}_{x\sim P}\left[-\log P(x)\right]. \tag{3.37}$$

The agent gets a bonus reward at each time step proportional to the entropy of the policy at that timestep. It changes the original RL problem (3.24) to

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{y=0}^{\infty}\gamma^t\left(r_t - \alpha \mathop{\mathbb{E}}_{a\sim\pi}\left[\log\pi(a|s_t)\right]\right)\right], \tag{3.38}$$

where α is a weighting factor. Mathematically, it can be positive or negative, even though it is always positive in practice. It bolsters exploration if high, and conversely.

In the particular case of Q-learning, the softmax strategy, also known as Boltzmann or Gibbs exploration has been proposed as a substitute for the basic ϵ-greedy to adapt the exploration automatically (Sutton & Barto, 2018). It has the advantage to rely on the already estimated action-value function instead of training yet another function approximation. However, it is rarely used in state-of-the-art algorithms, especially in the continuous domain, because sampling from such a probability distribution is not straightforward. Either way, these noise models are temporally uncorrelated. It is usually effective to force exploration, but not necessarily, no matter if the standard deviation is extremely high. For a MDP representing a continuous-time system, the timestep must be chosen properly to give enough time for the effect of the randomness to build up, but it is not possible if very high reactivity is mandatory for some reason. If the random process changes dramatically at every timestep but has zero mean, then it would have no effect on average. It is true for Gaussian noise but

also for the ϵ-greedy strategy. Temporally correlated random processes can be used to get around this issue. The Ornstein-Uhlenbeck process (1930) is one of them. It is a variant of the random walk that has the tendency to move back towards a central location, with a greater attraction further away from the center. Roughly speaking, it is a continuous-time normal distribution. It is rarely used in continuous control benchmarks because the timestep is large, and hence it has little advantage over uncorrelated normal noise, not to mention implementation difficulties. Various other types of noise have been proposed in the literature to avoid clipping or squashing the actions if bounded, in particular, the beta distribution (Chou et al., 2017) and the truncated gaussian distribution (Fujita & Maeda, 2018). Besides, a popular trick to improve exploration for off-policy methods is to start with a completely random policy that spans uniformly the whole action space. Then, after a few iterations, it switches back to a more sensible exploration strategy. It is a step forward to ensure that the state-action space has been explored thoroughly and that the history of past data has not degenerated at the early stage.

EA has been relying on parameter perturbations since the 70s, notably *Evolutionary strategies* that we briefly present in appendix E.9. They are known to explore the search space extensively, but they have poor sample efficiency compared to RL methods, including on-policy algorithms (cf. section 3.2.3). This can be explained by its inability to leverage the temporal structure of the problem. For the interested reader, Eiben and Smith (2015) offers a thorough introduction to the field of Evolutionary Computing. It is very possible to apply the same technique to RL since the policy is a parametric function approximation. Matthias et al. (2018) were first to investigate how adding noise in the parameter space rather than the action space can be effectively combined with off-the-shelf RL algorithms to improve exploration. Adding noise in the parameter space in RL can be seen as a bridge between Evolutionary Strategies and classical RL methods to take the best of both worlds. This form of exploration is applicable to both high-dimensional discrete environments and continuous control tasks, using on- and off-policy methods. Besides, It works with any observation and action space. Experiments were conducted on the widespread *DeepMind Control Suite* (dm-control) (Tunyasuvunakool et al., 2020), which gathers a set of control tasks based on Mujoco (Todorov, 2010). The results indicate that parameter noise leads to a more consistent exploration and a richer set of behaviors than adding noise in action space, especially in tasks with a sparse reward.

Adding noise to the action is well suited for problems that can be solved by simply exploring the state space. First, it must be uniformly challenging for the action to drag the state in any direction whatever the current one. Mathematically, it implies that the graph of the MDP is *strongly connected* or *irreducible*, which means that any state *communicates* with any other state. Two states i, j communicate if they are *accessible* from each other after an arbitrary number of steps. Maze solving fits perfectly with this description, but it may not be the case. Even if the MDP is irreducible, some states may only be accessible after a specific sequence of past events, which would have an extremely low probability under random actions. Secondly, the reward must be dense and informative, otherwise, it would not be possible to sort out

different states. It is said to be a *hard-exploration problem* if these two requirements are not satisfied. How to solve this class of problem is an active research topic.

Extrinsic vs. intrinsic Reward

A dense and informative reward is a form of extrinsic motivation for the agent to complete the task. As such, it is supposed to constantly steer the agent toward better policies. However, landscapes induced by such a hand-crafted reward are often deceptive (Lehman & Stanley, 2011), which means that part of the time, the reward is pointing the wrong way. It follows that a randomly initialized agent is unlikely to uncover a path through the search space that ultimately leads to achieving the task. For biped locomotion, rewarding falling the farthest is likely to encourage the agent to jump away rather than walk, which exemplifies a deceptive local optimum.

One solution is to bootstrap learning by guiding the agent with a few expert demonstrations to avoid relying exclusively on exploration at the beginning of the learning (Nair et al., 2018; Vecerik et al., 2017). This approach, combined with the decomposition of the task into a curriculum of subtasks requiring short sequences of actions, was successfully applied to the infamous 'Montezuma's Revenge' atari game but failed on 'Pitfall!' (Salimans & Chen, 2018). They are respectively the second and first hardest atari games. However, there is no solution readily available for many real-world problems, so it is impossible to provide expert demonstrations. Moreover, curriculum learning is not generic as it involves decomposing the task into subtasks or parametrizing it to adjust its difficulty.

A more promising direction is to augment the original reward with some *intrinsic reward* promoting *curiosity* (Schmidhuber, 1991). Such a reward is dense, without expert bias, and compatible with any existing training algorithm. Hence, it is broadly applicable to many problems without domain knowledge. In general terms, curiosity is about encouraging the agent into seeking to learn more about the world (both itself and its surrounding environment) without explicitly seeking to achieve an objective. Such an intrinsic reward is often referred to as a bonus, to distinguish it from a regularization term. The entropy in equation (3.38) can be viewed as a naive curiosity bonus that promotes diversity of action for itself without looking at its effectiveness.

Lehman and Stanley (2011) propose highly generic curiosity-driven *intrinsic reward* search for the behavioral *novelty*. Because there are only so many simple behaviors, the search for novelty leads to increasing complexity. A good metric of the novelty is the sparsity in the behavior space, that is, the space of unique behaviors. Areas with denser clusters are rated less novel and therefore rewarded less. Basically, it boils down to encouraging the agent to explore states it has never encountered before. This approach was originally developed with Evolutionary Computing in mind. In this context, the performance of the agent is only assessed at the end of the episodes. Thus, Lehman and Stanley define the novelty as the average distance between the final state of the agent and its k-nearest neighbors in the history of all the final states of previous episodes. They have shown that novelty search significantly outperforms objective-based search in some maze navigation and biped

walking tasks. In RL, the reward is provided at every transition step and not only at the end. Generalizing the metric introduced by Lehman and Stanley requires storing the whole history of previously visited states. This causes scalability issues because computing the k-nearest neighbors is getting more and more impracticable as the amount of data keeps growing unboundedly. Nonetheless, (Seo et al., 2021) have successfully applied to simple locomotion and navigation tasks from dm-control. For discrete state spaces, one could keep track of the state visitation frequencies: the states that have been visited less frequently would be rated as more novel. Although this approach is computationally effective, it scales poorly with dimensionality as it completely ignores the similarities between states.

Parametric function approximations such as neural networks do not face these scalability issues regarding the number of samples or the dimensionality. However, it is not clear what to approximate with it since the state visitation is ill-defined in the continuous domain. (Burda et al., 2019) came up with a very peculiar but effective way to overcome this issue called *Random Network Distillation* (RND). A first neural network is randomly initialized and kept fixed forever after. This mapping is fully deterministic and forms a scalar (smooth) embedding for observations without any physical meaning. Then, a second network is trained to approximate the specific scalar observation embedding defined by the first one. Both shares the same architecture but are initialized differently. This way, the second network is guaranteed to have the ability to perfectly reproduce the first one regardless of its expressiveness. The novelty of a given state is characterized by the error between the output of the two networks for the corresponding observation. This prediction error can be interpreted as an exotic metric of the similarity between the current state and all the previously visited states, much like the average distance from the k-nearest neighbors. How sharply the similarity between neighboring states drops relative to their actual distance depends on the generalization ability of the model and thereby its expressiveness: the similarity would drop faster if the expressiveness is increased. Thus, the complexity of the architecture must be chosen in relation to the problem at hand, which may be tricky. Besides, if the state is partially observable, different states would map to the same observation, which introduces a bias. Yet, Burda et al. (2019) have successfully applied their method to 'Montezuma's Revenge' but failed on 'Pitfall!'. They are respectively the second and first hardest atari games.

Alternatively, Oudeyer et al. (2007) intend to assess the knowledge about the world instead of simply monitoring how well the state space all a whole has been explored so far. A subsidiary model called expert is responsible for predicting the future observation from the current one and the ongoing action of the agent based on all the collected data from the beginning of the learning, namely learning a forward dynamics model. There is no assumption regarding the learning rate of the model, thus this approach complies with one-shot learning methods or non-parametric function approximations like KNN, as well as with slowly learning ANNs that would be trained concurrently to the policy. The error between the actual and predicted future state is then used as a basis to define an intrinsic reward. This prediction error is expected to capture the state visitation frequency to some extent. It should be large

for novel states that have been encountered before, and small if frequently visited in the past. It is related to the *epistemic uncertainty*, namely data are missing to properly approximate the dynamics at the current state. Consequently, it seems reasonable to maximize the prediction error. Stadie et al. (2015) applied this method on many Atari games with mixed results. In particular, the agent fails on 'Montezuma's Revenge'. This is because the prediction error can be large for many other reasons:

- the true dynamic model may be stochastic, known as *aleatoric uncertainty*,
- the state is only partially observable and information is missing,
- The expressiveness of the function approximation is not high enough.

The aleatoric uncertainty is the most problematic, namely the agent being attracted by local sources of entropy unrelated to exploration. A famous thought experiment is the so-called "Noisy-TV" problem. It is a pathological scenario where a screen displays white noise. The next state is unpredictable, and so the agent would enjoy a large intrinsic reward by just looking at the TV, encouraging it to stay in place without getting any closer to solving the actual task. (Achiam & Sastry, 2017) replace the classical L^2-norm used to compute the prediction error by a metric based on the *Kullback-Leibler* (KL) divergence over iterations. This approach is very robust to aleatoric uncertainties but tends to be computationally expensive. Moreover, it does not help much with unlearnable situations.

Oudeyer et al. suggest maximizing the learning progress by monitoring the evolution of the error rate. This ensures that the agent will not stay in front of white noise or unlearnable situations because this does not lead to a decrease in prediction error. The corresponding intrinsic reward is defined as the inverse of the difference between the expected mean prediction error in the close future and the mean error rate in the close past, which is an estimate of the local derivative of the mean error rate. They validated their approach on several simple robotic tasks both in simulation and reality. However, they focus their analysis on exploration, ignoring the actual performance of the agent, so it is hard to infer whether it would be effective in real-world applications with concrete objectives. More recently, Pathak et al. (2017) proposed to learn an inverse dynamics model, i.e. learning to infer the action from the past and current states. As opposed to the forward dynamics model, all the variability in the world that is not affected by the action taken by the agent would be useless to make this prediction and filtered out. Pathak et al. have demonstrated that an intrinsic reward combining both forward and inverse dynamics models is insensitive to aleatoric uncertainty for the most part. In particular, they achieved high scores in the video games 'Doom' and 'Super Mario Bros'.

All the previous approaches focus on providing a long-term novelty bonus. It is sufficient to deal with a local exploration, i.e. exploring the consequences of short-term decisions. However, a global exploration that involves coordinated decisions is beyond their reach. To circumvent this limitation, Badia et al. (2020) propose an *episodic novelty* metric based on an inverse dynamics model. The history of inferred past actions is stored in a buffer over a time window of fixed length. The latter is supposed to be consistent with the short coordinated decision sequences

that the agent must perform to solve the task. Then, for every predicted action, its novelty is defined as the distance from its nearest neighbors. The episodic novelty of the predicted action is then used to modulate the long-term novelty bonus for encountering new states coming from RND. This method has been able to solve 'Pitfall!' for the first time without expert demonstrations.

3.2.3 Taxonomy of Algorithms for Continuous Control

Many learning algorithms have been introduced over the years, and it is impossible to review all of them with their connections. Thus, only the most famous ones are listed here. A usual tree graph representation has been adopted, even though this choice is questionable considering the modularity of the algorithms. Only the state-of-the-art methods will be presented in detail in this section. A thorough discussion of the historical algorithms and the main alternatives to RL is available in appendix E. As mentioned in section 3.2.2, algorithms can be partitioned according to three different criteria: model-free or model-based, on- or off-policy, and value- or policy-based.

Model-Based vs. Model-Free Methods

Traditional model-based control methods rely on a model of the system for planning actions on a real robot. This theoretical model is sometimes a very rough approximation to make computations tractable, e.g. the *Linear Inverted Pendulum Model* (LIPM). Moreover, the sensor measurements may be noisy and provide a partial observation of the current state of the system, not to mention communication delays. These discrepancies limit the performance that can be expected from such controllers.

Model-free policy learning methods are promising approaches for generating robust feedback controllers that incorporate model knowledge and implicitly plan over a horizon through trials and errors. Such policies have demonstrated their capability to handle large model uncertainties and unexpected events at runtime if combined with transfer learning techniques such as domain randomization (cf. section 4.2.2). However, they suffer from interpretability issues, often lack theoretical stability guarantees, and are difficult to fine-tune to tightly adjust the closed-loop behavior.

Alternatively, the reduced dynamics in the observation space can be learned explicitly. The resulting model is differentiable, so it can be used in conjunction with classical *Model-Based Predictive Control* (MPC) (Deisenroth & Rasmussen, 2011; Ha & Schmidhuber, 2018; Racanière et al., 2017). However, model learning is fundamentally hard. It was applied successfully to discrete state and action spaces, but it struggles in the continuous case. MuZero (Schrittwieser et al., 2020) is famous for playing Chess, Go, and Atari games above the human level.

Policy- vs. Value-Based Methods

RL algorithms are iterative. First, sample trajectories are collected following the current policy to take action. Then, it leverages those data to improve the policy.

This loop is repeated until convergence. An algorithm will be policy-based, value-based, or both depending on the way it makes use of the collected data. A method is said to be policy-based if the policy is parametrized with a function approximation π_θ and the parameters θ are optimized. It is value-based if the same goes for a value function instead. These two approaches are not mutually exclusive as one can imagine training jointly the policy and a value function.

If a method is purely policy-based, then it solves the problem explicitly by updating the parameters of the policy θ by gradient descent so as to maximize the expected return. One example of such a basic algorithm is REINFORCE (Williams, 1992), i.e. vanilla policy gradient. Conversely, if a method is purely value-based, then it finds the solution indirectly by estimating an action-value function and defining the current policy as a by-product. This basic algorithm is known as Policy Iteration (Sutton & Barto, 2018). It is proven that defining the policy greedily according to equation (3.31) guarantees monotonic improvement of the policy if the action-value function is associated with the current policy and estimates it perfectly. This is the direct application of the *Policy Improvement Theorem*.

Theorem 3 (Policy Improvement Theorem)**.** *Let $\pi(a|s), \pi'(a|s)$ be two stochastic policies and define $Q_\pi(s, \pi') = \mathbb{E}_{a\sim\pi'}[Q_\pi(s, a)]$. If $\forall s \in \mathcal{S}$, $Q_\pi(s, \pi') \geq V_\pi(s)$, then it holds that $\forall s \in \mathcal{S}, V'_\pi(s) \geq V_\pi(s)$. It means that π' is at least as good a policy as π.*

$Q_{\pi_k}(s, \pi_{k+1}) = \max Q_{\pi_k}(s, a)$ and for a greedy policy $V_{\pi_k}(s) = \mathbb{E}_{a\sim\pi_k}[Q_{\pi_k}(s, a)]$, hence $Q_{\pi_k}(s, \pi_{k+1}) \geq V_{\pi_k}(s)$. Moreover, to show convergence to the optimal policy, it is enough to show that if there is no improvement in the value function at any state. If $\exists k$ s.t. $\forall s \in \mathcal{S}, V_{\pi_{k+1}}(s) = V_{\pi_k}(s)$, then $V_{\pi_k}(s)$ satisfies the Bellman optimality equation, and thus $V_{\pi_k}(s) = V^*$. This policy is fully deterministic, which means that exploration only comes from the initial condition and the transition function. If both are deterministic, then the trajectory would be unique. In practice, those conditions are often met, so that the agent can hardly discover interesting behaviors. *State-Action-Reward-State-Action* (SARSA) is an early algorithm based on this principle that is now famous (Rummery & Niranjan, 1994). Nowadays, it is not an active research topic as this approach is significantly outperformed by others. Alternatively, the optimal action-value function can be estimated in place of the current one. The same technique as before is used to define the current policy. This basic method is called Q-learning (Watkins, 1989).

Hybrid methods being both policy-based and value-based are known as *actor-critic* (Konda & Tsitsiklis, 2000). It covers a wide range of very different algorithms, but the core idea remains the same: learning a value function to compute the update direction for the parameters of the policy. The actor is the policy that takes action, then the critic estimates a value function to suggest a meaningful update direction to the actor. Most state-of-the-art algorithms derive from this paradigm. Such methods are not restricted to estimating the action-value function. Indeed, the state-value function can be used as a baseline to reduce the variance of the gradient estimate. Variance reduction techniques are presented in appendix E.7.1.

On- vs. Off-Policy Methods

RL algorithms can be partitioned into two categories: *on-policy* and *off-policy*. The distinction between them is simple but has many implications. An algorithm is said to be on-policy if the target and behavior policies are the same in expectation, off-policy otherwise. Still, the behavior policy may be somehow related to the target policy even in the off-policy setting. Typically, it is a past version of the target policy in state-of-the-art algorithms. Which quantities are involved in the learning is specific to every algorithm, depending on whether they are policy-based, value-based, or both. It could be the expected return, or the state- and action-value functions under the target policy. In addition, the off-policy setting allows learning directly the action-value function under the optimal policy. As stated in equation (3.34), this quantity only depends on the transition function. *Asynchronous A2C* (A3C) (Mnih et al., 2016), *Trust Region Policy Optimization* (TRPO) (Schulman et al., 2015) and *Proximal Policy Optimization* (PPO) (Schulman et al., 2017) are prominent on-policy methods, while *Deep DPG* (DDPG) (Lillicrap et al., 2016), *Twin-Delayed DDPG* (TD3) (Fujimoto et al., 2018), SAC are some of the most effective off-policy methods. These algorithms and many others are reviewed in section 3.2.3 and appendix E.

In on-policy algorithms, the collected data are only valid for the current iteration and must be thrown away afterward since the target policy has been updated and therefore now differs from the behavior policy. The principal advantage of the off-policy setting is to allow the agent to learn from the whole history of previously collected data without introducing bias since matching between the behavior and target policy is not required. More specifically, any available trajectory can be used to improve its estimate as long as the MDP is *stationary*. In robotics, this assumption never truly holds as the physical properties of robots slowly change over time due to wear and tear. This process is usually very slow compared to the update step and hence can be ignored during the training. Yet, it would be interesting to never fully stop learning to continuously adapt to these changes, which is one of the main purposes of online RL. More generally, it is recommended to discard the oldest data by storing a history of fixed size called *replay buffer*. It avoids keeping data that are no longer consistent with the current MDP. More importantly, it reduces the variance of the gradient estimate for the policy update. This in turn enables converging to better solutions and speeds up the training by permitting larger update steps without instability. Intuitively, the closer the transition steps in the replay buffer from the closed-loop behavior under the target policy the more information they provide on how to update the policy to improve its performance. This is related to *Importance Sampling* (IS), which is a technique for estimating quantities associated with a particular distribution, while only samples drawn from another one are available (cf. **??**). In practice, it consists in weighting each transition step involved in the update of the policy according to their relevance at the time being.

The difference between on and off-policy methods is fundamental. The former aims at a moving target. It is a local optimization process. Only trajectories induced by the current policy are sampled to improve it. It is ideal in the sense

that it optimizes the parameters solely for what the current policy does in closed-loop interaction with the environment. The collected transition steps belong to a low-dimensional manifold embedded in the state-action space, so that only vectors tangential to this manifold are considered as candidate update direction for the policy parameters. In this way, estimating accurately a meaningful update direction requires a minimal amount of data. On the contrary, off-policy methods are closer to global optimization. It requires mapping the whole state-action space instead of only the state space, but the function to estimate is stationary in contrast to on-policy methods. It enables mixing new samples under the current policy and old ones during training, which dramatically improves sample efficiency. However, the history of state-action pairs span a region of the state-action space much larger than necessary.

Neither off-policy nor on-policy is strictly superior to the other. Off-policy methods tend to face more difficulties in finding appropriate policies for complex problems, whereas on-policy algorithms have poor sample efficiency. What matters the most to the end-users depends on the scenario. In robotics, it is not uncommon to have to collect hundreds of millions on transition steps, corresponding to months in reality. It is nonetheless possible to train a policy in a few hours in simulation as the throughput is around tens of thousands per second. Yet, it is not suitable for online RL where increasing the throughput means duplicating real environments, which is very costly, especially in the robotic field. Moreover, interactions in the real world are time-consuming and demand special care, so they must be limited by all means.

Landscape of Policy Learning Algorithms

Nowadays, the dichotomy between value and policy-based methods is hardly relevant for continuous control tasks. All modern algorithms are actor-critic, and so they are both value- and policy-based to some extent. Similarly, recent algorithms blur the line between on- and off-policy paradigms. It is also quite common that model-based planning and control methods are somehow combined with model-free RL algorithms. One way to do so is to provide pre-computed trajectories to guide the agent, either as expert demonstrations (Levine & Koltun, 2013) or as an additional term in the reward function (Xie et al., 2019). Another option is to train an agent to predict high-level features that will be realized via traditional MPC, e.g. the position and timing of the next footprints. The motivations are the same as always: improving learning stability and sample efficiency while ending up with more powerful controllers. The landscape of the most renowned algorithms is depicted in figure 3.9. There is no vertically ordering by the time of publication, but top-down connections do reflect the heredity relationship between algorithms. Zhu (2021) provides a continuously updated exhaustive listing of all existing methods.

The existing policy learning methods are roughly ordered according to their sample efficiency in figure 3.10. Still, it is important to keep in mind that the sampling efficiency alone is not sufficient to sort out algorithms in terms of wall time, which is what really matters in the end. The latter depends on the cost of collecting samples relative to the policy update rule. Although Evolutionary Strategies are known to

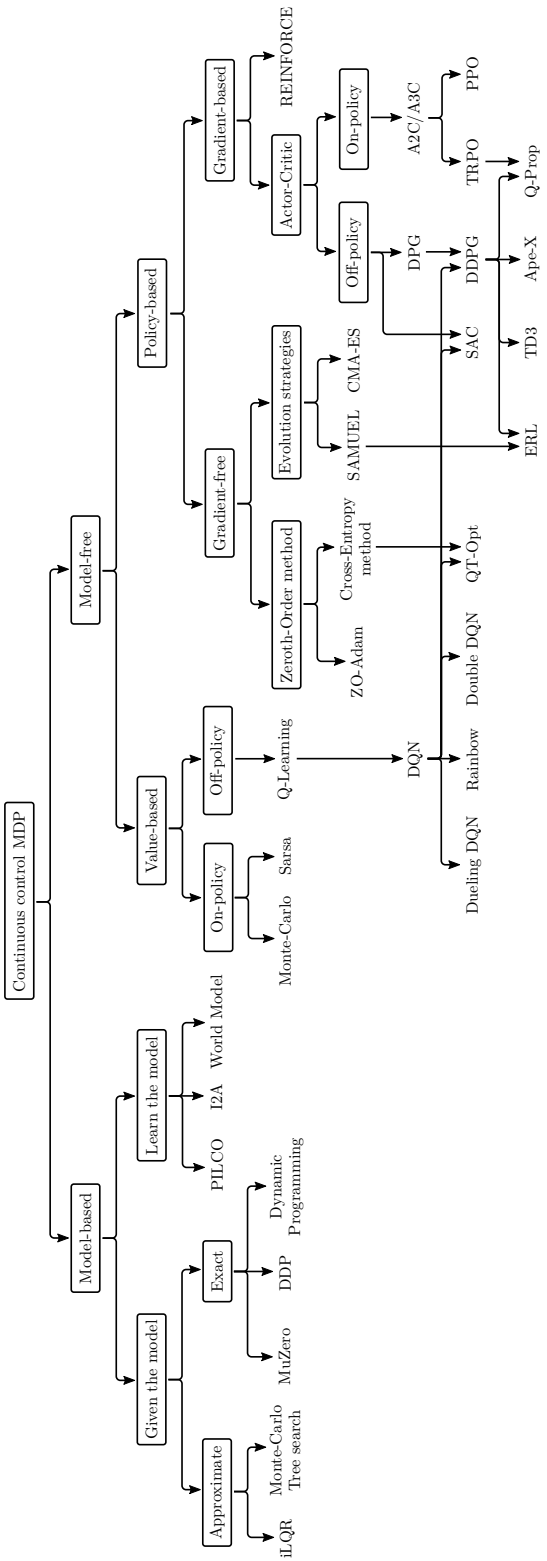


Figure 3.9: Taxonomy of policy learning algorithms

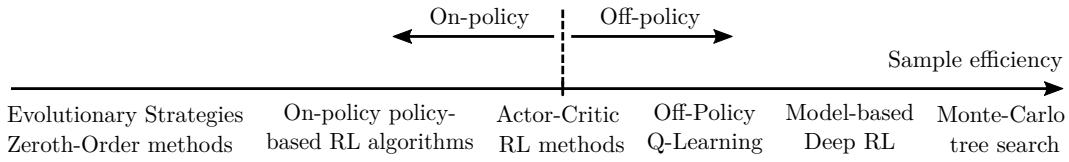


Figure 3.10: Sample efficiency of policy learning algorithms

have poor sample efficiency (cf. appendix E.9), their policy update rule is extremely cheap and they explore the search space extensively. Besides, they are highly parallelizable, which speeds up computations even further. All in all, Evolutionary Strategies may be faster overall than RL algorithms in practice.

Off-policy methods are ostensibly more efficient than on-policy ones (cf. section 3.2.3). Thus, many on-policy algorithms offer the opportunity to further improve their sample efficiency with optional off-policy updates. As for any on-policy algorithm, all previous training samples are still discarded and new ones are collected under the current policy at every training iteration, then the policy is updated in the usually on-policy way. At this point, any additional policy update that would be performed on the same training batch would get more and more off-policy. This discrepancy would introduce a bias in the gradient estimate that cannot be ignored.

It is tempting to correct this bias using IS when training a stochastic policy. However, it does not work well because of the high variance of *likelihood-ratio gradient* (LR) estimators. More recently, weighted IS (Mahmood et al., 2014) and adaptive techniques (Hachiya et al., 2009) have been proposed to overcome those limitations. Weighted IS is biased but asymptotically correct and with much lower variance. Although it has some benefits for pathological problems, it brings marginal improvements at best in general. More advanced methods to reduce the bias without adding much variance are presented in appendix E.7.1. How to improve these approaches, especially in the context of off-policy learning, is still an active research topic. In any event, these additional policy updates are doomed to become less and less effective compared to the first one. The correction of the bias is usually combined with another technique to limit it in the first place. The most common consists in bounding the KL divergence at every update, but other variants are found in state-of-the-art on-policy methods. PPO is known to have a much cheaper policy update rule than TRPO thanks to several mathematical approximations. As such, off-policy updates are rarely performed for TRPO, whereas they are about 20 per iteration for PPO.

It is possible to get around this issue by training a deterministic policy instead. It enables getting rid of the IS entirely. *Deterministic Policy Gradient* (DPG) (Silver et al., 2014) was proposed first, following between DDPG and TD3 and others. However, the convergence of these algorithms is usually not guaranteed with non-linear function approximators, and extensive hyperparameter tuning is required to counterbalance their inherent instability. To overcome these limitations, SAC manages to learn a stochastic policy in an off-policy way by relying on the reparametrization trick to compute the gradient instead of IS. It reduces the variance significantly and

guarantees monotonic improvement of the policy.

To go further, it is possible to unify on- and off-policy methods to get the best of both worlds. Q-Prop (Gu et al., 2017) can be seen as using an off-policy critic to reduce the variance of the policy gradient or using on-policy *Monte-Carlo* (MC) returns to correct for bias of the critic gradient. The critic learns the action-value function off-policy. Then, it uses its first-order Taylor expansion as a control variate, resulting in an analytical gradient term through the critic and a MC policy gradient term consisting of the residuals in advantage approximations. More recently, LR and *ReParametrization gradient* (RP) gradients have been unified by enabling interpolation between the two approaches (Liu et al., 2018; Parmas & Sugiyama, 2021; Tang & Abbeel, 2010). See appendix E.7.1 for details.

Concurrently, there is a new trend in combining Evolutionary Strategies with RL methods. Various approaches have been proposed over the years, *Evolutionary Reinforcement Learning* (ERL) being one of the most successful (Khadka & Tumer, 2018). It mixes EA with DDPG, but any off-policy RL algorithm would be equally valid. It outperforms EA, PPO and DDPG on Atari games and dm-control benchmarks.

State-of-the-Art On-Policy Algorithms

Vanilla policy gradient is presented in appendix E.5. The monotonic improvement of the policy is not guaranteed. A conservative constant step size would limit the risk of a catastrophic performance drop but cannot prevent it entirely. Indeed, even tiny differences in parameter space may have a large impact on the policy locally due to ill-conditioning. Dynamically adjusting the step size to make it as large as possible would greatly improve learning stability and sample efficiency. The key is bounding the distance between the current and updated policy according to the similarity of their outputs. This idea is simple but has strong mathematical foundations.

The objective is still to maximize the expected return $J(\pi)$ over the trajectories induced by the current policy π. Any independent baseline can be subtracted from the objective function without affecting its optimum. Let us consider the improvement of the current policy π relative to the behavior policy β that was used to collect samples, $J(\pi) - J(\beta)$. It is equal to the expectation of the relative advantage of the current policy (Kakade & Langford, 2002),

$$\pi^* = \arg\max_{\pi} J(\pi) - J(\beta) = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t A_\beta(s_t, a_t)\right]. \tag{3.39}$$

The expectation over the trajectory distribution can be reformulated to make the state and action distributions explicit. It yields,

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ a\sim\pi}} \left[A_\beta(s, a)\right], \tag{3.40}$$

where ρ_π is the unnormalized discounted *stationary state distribution* induced by policy π. Only transition steps associated with the behavior policy β are available,

so IS is used to swap the action distribution.

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ a\sim\beta}} \left[\frac{\pi(a_t|s_t)}{\beta(a_t|s_t)} A_\beta(s_t, a_t) \right]. \tag{3.41}$$

The likelihood ratio between the target and behavior policy $\pi(a_t|s_t)/\beta(a_t|s_t)$ is known as the *importance weight*. It weighs the importance of the transition steps in the update of the policy π. Intuitively, the more likely the current policy would be to take the same action as the behavior one, the more relevant it is to take into account the associated advantage $A_\beta(s_t, a_t)$, and conversely. At this point, the state distribution ρ_π is not consistent with the collected data, but it is not possible to swap it for the right one. One way to get around this limitation is to maximize a lower bound of the real objective function. Kakade and Langford (2002) first suggested the *Conservative Policy Iteration* (CPI), but it only applies to specific policy update rule. Schulman et al. (2015) generalized it by relying on metrics of the distance between the action distributions associated with the current and the behavior policy, e.g. the *Total Variation* (TV) distance, which is defined as $D_{\text{TV}}(\beta||\pi)[s] = (1/2)\|\beta(\cdot|s) - \pi(\cdot|s)\|_1 = \int_{a\in\mathcal{A}} |\beta(a|s) - \pi(a|s)|$. The following lower bound holds (Achiam et al., 2017),

$$J(\pi) - J(\beta) \geq \underbrace{\mathop{\mathbb{E}}_{\tau\sim\beta} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi(a_t|s_t)}{\beta(a_t|s_t)} A_\beta(s_t, a_t) \right]}_{L_\beta^{\text{CPI}}(\pi)} - \frac{2\gamma\epsilon_\pi}{1-\gamma} \mathop{\mathbb{E}}_{s\sim\rho_\beta} \left[D_{\text{TV}}(\beta||\pi)[s] \right], \tag{3.42}$$

where $\epsilon_\pi = \max_{s\in\mathcal{S}} |\mathbb{E}_{a\sim\beta}[A_\beta(s, a)]|$. Another widespread metric is the KL divergence. It is the expectation of the log-likelihood ratio between the two distributions, i.e. $D_{\text{KL}}(\beta||\pi)[s] = \mathbb{E}_{a\sim\beta}\left[\log \frac{\beta(a|s)}{\pi(a|s)}\right]$. It can be interpreted as the expected excess of *surprise* for sampling from the current policy in place of the behavior one. The TV distance is related to the KL divergence by Pinsker's inequality, $D_{\text{TV}}(\beta||\pi) \leq \sqrt{D_{\text{KL}}(\beta||\pi)/2}$. Combining this with Jensen's inequality,

$$J(\pi) - J(\beta) \geq L_\beta^{\text{CPI}}(\pi) - \frac{\sqrt{2}\gamma\epsilon_\pi}{1-\gamma} \sqrt{\mathop{\mathbb{E}}_{s\sim\rho_\beta} \left[D_{\text{KL}}(\beta||\pi)[s] \right]}, \tag{3.43}$$

Schulman et al. (2015) have proven that monotonic improvement of the policy is guaranteed as long as the right-hand side in equation (3.43) inequality increases. Thus, the true objective can be replaced by the lower bound. However, it is not desirable because it would lead to very small step size. TRPO enables doing larger steps using a trust region constraint,

$$\pi_{k+1} = \arg\max_{\pi} L_{\pi_k}^{\text{CPI}}(\pi) \text{ s.t. } \bar{D}_{\text{KL}}(\pi_k, \pi) \leq \delta, \tag{3.44}$$

where $\bar{D}_{\text{KL}}(\pi_k, \pi) = \mathbb{E}_{s\sim\rho_{\pi_k}}[D_{\text{KL}}(\pi||\pi_k)[s]]$, and $\delta > 0$ is the step size. It comes with a worst-case performance degradation guarantee that depends on δ,

$$J(\pi_{k+1}) - J(\pi_k) \geq \frac{-\sqrt{2\delta}\gamma\epsilon_{\pi_{k+1}}}{1-\gamma}. \tag{3.45}$$

Finally, this optimization problem is solved efficiently using a linear approximation of the surrogate objective and quadratic approximation of the KL constraint. This results in a second-order optimization method involving the natural policy gradient. Introducing the parameter θ of the policy,

$$\theta_{k+1} = \arg\max_{\theta} g_k^T(\theta - \theta_k) \text{ s.t. } \frac{1}{2}(\theta - \theta_k)^T H_k(\theta - \theta_k) \leq \delta, \tag{3.46}$$

where $g_k = \nabla_\theta L_{\pi_k}^{\text{CPI}}(\pi_\theta)|_{\theta_k}$ and $H_k = \nabla_\theta^2 \bar{D}_{\text{KL}}(\pi_k, \pi_\theta)|_{\theta_k}$. It is a simple *Quadratic Program* (QP) that can be solved in closed-form,

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g_k^T H_k^{-1} g_k}} H_k^{-1} g_k. \tag{3.47}$$

In practice, the CG method is used to compute x_k s.t. $H_k x_k \approx g_k$. As before in section 3.1.2, it is much cheaper than evaluating the inverse matrix H_k^{-1} and has better numerical accuracy. The QP is only an approximation of equation (3.44). Therefore, it is not guaranteed that the surrogate objective truly improves and that the KL constraint is satisfied. Equation (3.47) may compute large steps that cause a catastrophic drop in performance. Line search is performed to prevent it,

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{x_k^T H_k x_k}} x_k. \tag{3.48}$$

where $\alpha \in [0, 1]$ is the backtracking coefficient, and j is the smallest nonnegative integer such that $\pi_{\theta_{k+1}}$ satisfies the KL constraint and produces a positive surrogate objective. The gradient g_k is equal to the jacobian of the original objective $\nabla_\theta J(\pi_\theta)|_{\theta_k}$, which is given by the Policy Gradient theorem 10. Its computation requires estimating the advantages $A_{\pi_k}(s_t, a_t)$ beforehand. Different estimator are possible but *Generalized Advantage Estimator* (GAE) (Schulman et al., 2016) is by far the most common. It enables a trade-off between bias and variance. More details are provided in appendix E.7.1. On its side, the Hessian H_k corresponds to the average *Fisher Information Matrix* (FIM) and can be computed without second-order derivative (Kakade & Langford, 2002; Schulman et al., 2015),

$$H_k = \mathop{\mathbb{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k}^T \right]. \tag{3.49}$$

TRPO algorithm is summarized in algorithm 1. It is a great improvement over the vanilla policy gradient, but it has several flaws. First, it cannot be used in conjunction with a more advanced SGD optimizer such as ADAM since the update step is governed by the natural policy gradient. Next, the policy update is computationally expensive and difficult to implement compared to the original method. As mentioned in section 3.2.3, TRPO allows for performing several policy updates using the same training batch, but it is not worth it because it increases the wall time in practice.

Algorithm 1: Trust Region Policy Optimization (Schulman et al., 2015)

Input: Backtracking coefficient: α, Initial parameters of the policy: θ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect a set $\mathcal{D}_k$ of N trajectories τ of length T on policy $\pi_k = \pi(\theta_k)$;
  Compute advantages $\hat{A}_t = \hat{A}_{\pi_k}(s_t, a_t)$ using any advantage estimator;
  Estimate policy gradient $\hat{g}_k$ as

$$\hat{g}_k = \frac{1}{N} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \gamma^t \hat{A}_t \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k};$$

  Estimate the average KL Hessian / Fisher Information Matrix $\hat{H}_k$;
  Use the Conjugate Gradient method to compute $\hat{H}_k x_k \approx g_k$;
  Compute initial policy step $\Delta_k = \sqrt{\frac{2\delta}{x_k^T H_k x_k}} x_k$;
  for $j = 0, 1, 2, \ldots$ **do**
    Compute candidate parameters $\theta = \theta_k + \alpha^j \Delta_k$;
    if $L_{\pi_k}^{CPI}(\pi_\theta) \geq 0$ **and** $\bar{D}_{KL}(\pi_k, \pi_\theta) \leq \delta$ **then**
      Update the policy $\theta_{k+1} \leftarrow \theta$;
      break;
    end
  end
end

PPO has been proposed to get around these downsides. According to equation (3.45), the potential degradation of the policy for using the surrogate objective $L_{\pi_k}^{\text{CPI}}(\pi)$ in place of the true one is proportional to the TV divergence. It is straightforward to show that, $\|\beta(\cdot|s) - \pi(\cdot|s)\|_1 \leq \|\frac{\pi(\cdot|s)}{\beta(\cdot|s)} - 1\|_\infty$. It yields,

$$1 - \epsilon \leq \frac{\pi(a|s)}{\beta(a|s)} \leq 1 + \epsilon \implies \mathop{\mathbb{E}}_{s \sim \rho_\beta} [D_{\text{TV}}(\beta||\pi)[s]] \leq \epsilon. \tag{3.50}$$

Hence, it seems reasonable to constraint the likelihood ratio to limit the worst-case performance degradation. Instead of adding this constraint explicitly, the objective function is clipped in a way that it is never favorable to violate it. It is both effective and cheap since it is sufficient to follow the direction of the gradient. PPO introduces the surrogate objective $L_{\pi_k}^{\text{CLIP}}(\pi)$,

$$L_{\pi_k}^{\text{CLIP}}(\pi) = \mathop{\mathbb{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{\infty} \gamma^t \min\left(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t\right) \right], \tag{3.51}$$

where $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_k(a_t|s_t)}$ and $A_t = A_{\pi_k}(s_t, a_t)$. This expression looks complicated, but it is in fact quite simple. Let us consider every transition step individually. If the advantage is positive, then the objective is equal to $\min(r_t, 1+\epsilon)A_t$. It increases

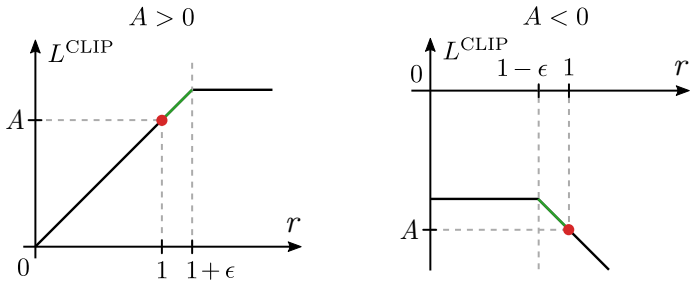


Figure 3.11: Clipped surrogate objective L^{CLIP} of PPO for a single transition step as a function of the likelihood ratio r, for positive (left) and negative advantages (right). The red circle shows the starting point before any optimization step, i.e. $r = 1$.

if the action becomes more likely. i.e. $\pi_\theta(a_t|s_t)$ increases. At a certain point, it will hit the ceiling $1 + \epsilon$ and the new policy does not benefit by going further away from the old one. Conversely, if the advantage is negative, the objective is equal to $\max(r_t, 1 - \epsilon)A_t$. This time, it increases if the action becomes less likely, as long as it is larger than $1 + \epsilon$. This behavior is illustrated in figure 3.11. An extra term that penalizes the average KL divergence $\bar{D}_{\text{KL}}(\pi_k, \pi)$ is often added. It can be interpreted as a compromise between the relaxed trust region formulation in equation (3.44) and the lower bound maximization in equation (3.43). In general, its weighting factor α is adaptive, using a very simple heuristic: compute $d = \bar{D}_{\text{KL}}(\pi_k, \pi)$, if $1.5 < \delta/d$ then $\alpha \leftarrow \alpha/2$, if $1.5 < d/\delta$ then $\alpha \leftarrow 2\alpha$. δ is supposed to be the higher bound of the update steps, but here is used as a target. It is based on the assumption that it is always profitable to make them as large as possible. PPO algorithm is summarized in algorithm 2. It has been demonstrated that it works at least as well as TRPO.

The value function is learned along with the policy. It is used in the computation of the advantage estimator to reduce the variance. If the parameters of the policy and the value network are independent, then they are going to extract different features. It would be beneficial to share the same encoding between them. It should end up with more robust features, and thereby improve the overall performance. Therefore, it has been suggested to share all hidden layers between the policy and value network. However, it creates a coupling between two different objective functions that may be locally conflicting. In practice, it improves the performance in some applications and is detrimental to others. Phasic Policy Gradient (Cobbe et al., 2021) achieves the best of both worlds by splitting the optimization into two phases. During the first phase, the whole policy network and the output layer of the value function are trained separately to optimize their respective objective. Then, during the second phase, the whole value network is trained while minimizing distortions to the policy. It gives complete freedom to optimize the policy, so it enables sharing features without negative side effects. Furthermore, the value function can be trained more aggressively with a higher level of sample reuse, which improves sample efficiency.

Algorithm 2: Proximal Policy Optimization (Schulman et al., 2017)

Input: Clipping ratio: ϵ, Initial KL penalty weight: α_0, KL target: δ, minibatch size: M, number of epochs: K, Initial parameters of the policy: θ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect a set $\mathcal{D}_k$ of N trajectories τ of length T on policy $\pi_k = \pi(\theta_k)$;

Compute advantages $\hat{A}_t = \hat{A}_{\pi_k}(s_t, a_t)$ using any advantage estimator;

for $l = 0, 1, 2, \ldots, K$ **do**

Randomly sample a minibatch $\tilde{\mathcal{D}}_l$ of size from $\mathcal{D}_k$;

Update the policy by maximizing the surrogate objective

$$\frac{1}{M} \sum_{\tau \in \tilde{\mathcal{D}}_l} \sum_{t=0}^{T} \left\{ \gamma^t \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_k(a_t|s_t)} \hat{A}_t, \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_k(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) ; - \alpha D_{\text{KL}}(\pi_k || \pi_\theta)[s_t] \right\}$$

via Stochastic Gradient-Descent, typically using ADAM.

end

Compute the average KL divergence $d = \bar{D}_{\text{KL}}(\pi_k, \pi_{k+1})$;

if $1.5 < \delta / d$ **then**

Update the KL penalty weight $\alpha_{k+1} \leftarrow \alpha_k / 2$;

else if $1.5 < d / \delta$ **then**

Update the KL penalty weight $\alpha_{k+1} \leftarrow 2\alpha_k$;

end

end

State-of-the-Art Off-Policy Algorithms

TD3 is a state-of-the-art algorithm for off-policy learning. It extends DDPG presented in appendix E.7 to further improve its stability by introducing a few mechanisms that are now shared among many algorithms:

- **Double Q-Networks**

  There is still only one policy but two Q-networks Q_{ϕ_1}, Q_{ϕ_2}. Those two networks are initialized with different random parameters ϕ_1, ϕ_2. One target Q-value is defined for each of them, denoted $\hat{Q}_1, \hat{Q}_2$ respectively. The only effect is to modify the way TD targets are computed. For a given transition step $e_t = \{s_t, a_t, r_t, s_{t+1}\}$, the maximum action-value is approximated by the minimum of the two target Q-values with respect to the target policy. The TD targets are shared between the Q-networks, but the latter are updated separately based on their own residual errors. Their respective loss function $L_i(\phi_i)$ are given by

$$L_i(\phi_i) = \frac{1}{N} \sum_{e_t \in \mathcal{B}} \left(r_t + \gamma \min_{j=\{1,2\}} \hat{Q}_j(s_{t+1}, \hat{\pi}(s_{t+1})) - Q_{\phi_i}(s_t, a_t) \right)^2. \tag{3.52}$$

Finally, the policy is optimized toward the best action associated with the first Q-network Q_{ϕ_1} arbitrarily. Lan et al. (2020) generalized this technique to any number of Q-networks and showed that it leads to an unbiased estimate with lower approximation variance.

- **Target Policy Smoothing**

  If the policy is fully deterministic, then it may overfit to narrow peaks in the target Q-value function. It worsens the effect of the overestimation of the true action-value, impeding the performance of the algorithm. To mitigate this issue, a random noise ϵ is added to the target action $\hat{a}$ outputted by the target policy. Doing so smooths out sharp peaks in the predicted Q-value along with changes in action. Indeed, it is no longer reliable to exploit peaks that are thinner than the standard deviation of noise σ, indirectly enforcing that similar actions should have similar values. In a way, it mimics the idea of SARSA, which is already using a noisy action for the sake of exploration, then using it to update the Q-value estimate as a consequence of being an on-policy algorithm. In practice, the noise is normally distributed with zero mean. It is drawn individually for all batch samples, then clipped to be bounded. For a given transition step $e_t = \{s_t, a_t, r_t, s_{t+1}\}$, it gives

$$\hat{a} = \mu(s_{t+1}) + \epsilon, \quad \epsilon \sim N(0, \sigma) \tag{3.53}$$

Chapter 4

Related work

Contents

4.1 Related Work on Online Trajectory Planning 99
4.1.1 Model Prediction Control 99
4.1.2 Imitation Learning 102
4.1.3 Trajectory Planning and Function Approximation 106
4.2 Related Work on Robust and Safe Control Policy 110
4.2.1 Classical Control Methods 110
4.2.2 Policy Learning Methods 113
4.2.3 Predictability and Safety in Reinforcement Learning 117

4.1 Related Work on Online Trajectory Planning

4.1.1 Model Prediction Control

Online Whole-Body Optimization

Model-based optimal control (aka. *Model-Based Predictive Control* (MPC)) blurs the line between planning and control (Wieber, 2006). This method aims at solving the same optimization problem as trajectory planning, as an integral part of the feedback control loop. Thus, the optimization problem must be solved online and repeatedly rather than once and for all, taking into account the updated estimate of the system state at every timestep (cf. section 2.2.3). Compared to the more traditional "plan then execute" paradigm, this approach enables the compensation of unexpected events and prevents the accumulation of errors due to uncertainties (Diehl et al., 2005). Extremely promising results have been obtained in simulation (Erez et al., 2013; Tassa et al., 2012).

Solving the *Optimal Control Problem* (OCP) for the exact whole-body formulation is computationally expensive, which is incompatible with the high-frequency precondition for closed-loop control without a lower-level controller. Indeed, it is commonly accepted that the minimal update period to maintain stability during very dynamic motions for humanoid robots is 1kHz. This limitation hinders the applicability of this method to real devices (Herdt, 2012). Still, Koenemann et al. (2015) have obtained impressive experimental results by combining two controllers in

cascade running: a *High-Level Controller* (HLC) and a *Low-Level Controller* (LLC). The HLC is responsible for solving the OCP at low frequency, while the LLC is tracking the resulting trajectory at high frequency. This method was validated on HRP-2 with 50ms and 5ms as high- and low-level update periods, effectively removing the need for offline trajectory planning. However, the underlying formulation is restricted to simple motions that neither make nor break contact with the environment.

Another limitation encountered when applying classical optimal control is the non-linearity and non-convexity of the OCP for the whole-body formulation. This is problematic because there is no guarantee of the existence of a solution, nor the convergence of the algorithm to a feasible point if any. Moreover, when it does converge, at best it is guaranteed to be a local optimum, and the required number of iterations is highly variable (Caron & Kheddar, 2017; Dantec et al., 2021). Different approaches have been suggested to handle solver failure. The most basic one consists in keeping the previous solution as the target trajectory for the LLC, while constantly retrying for updated state estimate until a valid solution is found. This is sufficient for tasks with a low failure rate (Erez et al., 2013; Koenemann et al., 2015), but it is unlikely to work for complex tasks. A more reliable method would be to rely on a simpler fallback control strategy that is guaranteed to converge and fast to compute. Caron and Kheddar (2017) have proven in simulation that it was satisfactory for walking on rough terrain despite a failure rate as high as 40%.

Being able to solve the OCP in 30 to 50ms for a whole-body model is already a significant breakthrough, but it is not enough to allow dynamic motion without resorting to trajectory tracking using a LLC. The computation time can be reduced dramatically by splitting the optimization into hierarchical sub-problems. Traditionally, the sequence of future footprints is planned first (timings and placements), then the centroidal dynamics, and finally the motor torques realizing it (Apgar et al., 2018; Caron & Kheddar, 2017; Dai et al., 2014; Herdt et al., 2010). This approach enables computation as fast as 300Hz on typical hardware Caron and Kheddar. The resulting motions are stable but rather conservative compared to offline trajectory optimization frameworks (Carpentier & Mansard, 2018; Carpentier et al., 2016).

On a different note, the tasks to solve are almost always *multi-modal*, i.e. there are several very different trajectories to achieve it. Similarly, the mapping from the task parameters to the output solution is not continuous in general if constraints are considered: smoothing varying the former may change dramatically the latter. Together, the behavior of the robot ends up being hardly predictable. It is not an issue if the system is not supposed to interact with humans, but it is in the context of collaborative robotics, e.g. with exoskeletons being the extreme embodiment. Typically, the task space gathers the current state of the system and some decision variables for the user to play with. It is tedious and unreliable to seek uniqueness and continuity by modifying the cost function and constraints. Besides, the absence of regularity according to the task implies that safety must be proven mathematically for all possible scenarios, which is almost impossible when it comes to human-robot interaction. It is a major barrier to software certification in human-centric applications.

Explicit Formulation

The explicit MPC formulation (namely pre-computed) can be considered when solving the optimization problem online is not an option (Alessio & Bemporad, 2009). The objective is to replace online optimization with a function approximation mapping tasks to trajectories, so that online operation reduces to a single evaluation. It allows for fast computations in constant time, eliminating the need for a LLC entirely. Historically, the mapping is computed offline using multi-parametric programming techniques (Pistikopoulos et al., 2007). It consists in solving the OCP analytically as a function of the task belonging to a given class. This is only tractable for *Quadratic Program* (QP) under linear constraints. Thus, the function must be piecewise linear.

This MPC approach can be viewed as a lookup table of linear gains over a polytopic partition of the task space (Bemporad et al., 2000). Incidentally, it has the advantage of providing a continuous manifold of solutions. It circumvents the predictability issue and can be certified numerically on a finite set of scenarios. However, it has major drawbacks as well: it requires linearizing the dynamics and the constraint of the original problem, and the size of the partition grows exponentially with the dimensionality of the task space.

Warm-Start

A more realistic compromise is to warm-start the OCP solver during online execution with a good initial guess. This does not affect the cost of each optimization iteration, only the number required to converge: an initial guess that is close enough to a local optimum leads to a much lower number of iterations. We observe empirically that it also significantly reduces the variability of the total computation time and prevents most optimization failures, two ubiquitous issues when it comes to *Non-Linear Program* (NLP) because of bad local minima.

The trajectory from the previous iteration is not always appropriate to warm-start the solver. Indeed, if some decision variable has changed in the meantime, then the solution of the OCP may be completely different. A much better candidate would be the nearest neighbor in the task space over a finite dataset of trajectories computed offline. It can be computed efficiently after building a multidimensional binary search tree data structure, so-called a *K-D tree* (Bentley, 1975). Berenson et al. (2009) have applied this method experimentally for manipulation planning. Yet, it does not scale well with the dimensionality of the task space as the distance between samples increases exponentially, similarly to multi-parametric programming.

Another option is performing regression using a function approximation. The approach offers a compressed representation of the dataset as a continuous mapping from tasks to trajectories. It has the advantage to be cheap to evaluate, but the exact model must be chosen carefully to avoid being inaccurate even on the samples themselves. This kind of so-called "Memory of Motion" has been assessed in simulation for simple tasks on several robots and toy models (Lembono et al., 2020; Mansard et al., 2018). Different function classes have been compared, including *Gaussian Mix-*

ture Model (GMM) and *Feedforward Neural Network* (FNN). For single-step motion planning on Talos, the number of iterations goes from 9.5 ± 4 to 3 ± 0.5 with 100% success rate, which is a great improvement both in average and variance. However, it is limited to uni-modal tasks. Otherwise, the mapping from tasks to trajectories would appear pseudo-random, and naive regression techniques are doomed to fail. Increasing the number of samples is counter-productive as it strengthens the averaging of surrounding solutions, which is meaningless. Up to now, the only solution that has been proposed is to rely exclusively on *K-Nearest Neighbors* (KNN), despite their above-mentioned limitations. Dantec et al. (2021) have validated this approach on Talos very recently. It is the first successful demonstration of whole-body MPC in real-life experiments for a torque-control humanoid robot.

In all these works, the initial guess is still fairly inaccurate, and they do not address the root cause of the problem, namely the non-uniqueness of the solutions. Having a better initial guess could speed up computation even further, one iteration being the limit if already feasible. However, non-linear solvers tend to be very sensitive to this kind of discrepancy, so just a small error can substantially increase the computation time. As a result, it is as difficult as classical MPC to certify the safety contrary to explicit MPC. It has no guarantee of convergence, even though it is better in practice than without warm-starting.

Warm-starting enables running a whole-body MPC at up to 100Hz (Dantec et al., 2021). It is only sufficient for quasi-static motions, yet expecting to run much faster is unrealistic. Indeed, the computation cost for a single iteration is too costly without simplifications of the original formulation. Therefore, this approach does not eliminate the need for a dedicated LLC.

4.1.2 Imitation Learning

Prediction Error Miminization

In essence, policy learning consists in optimizing the open- or closed-loop performance of a controller mapping the current state observation to the next action, potentially driven by some user-specified decision variables. In the same vein of explicit MPC, it replaces the entire OCP solving at runtime by a single function evaluation. One such approach is *Reinforcement Learning* (RL), which focuses on the closed-loop performance. In theory, policy learning may exhibit richer and nicer behaviors than the previous approaches based on MPC because no model simplifications or hierarchical decomposition is required. However, it comes with additional challenges.

In *Imitation Learning* (IL), a policy is trained to reproduce the behavior of an expert in the form of demonstrations (Attia & Dayan, 2018; Hussein et al., 2017). Those demonstrations are supposed to provide insight into how the task should be performed by the agent. They can originate from heterogeneous sources simultaneously. There is a myriad of possible sources, among them: a set of specialized policies already available, a whole-body MPC or motion capture recordings. Thus, this approach is extremely generic. *Behavior Cloning* (BC) is the simplest algorithm

to do imitation (Bain & Sammut, 1999). Every transition step is considered individually and the objective is to minimize the prediction error using standard regression. It corresponds to optimizing the open-loop performance in expectation under the expert state distribution. In a closed-loop at runtime, the policy might encounter completely different observations than those under expert demonstrations once it has made a mistake, leading to the compounding of errors and ultimately catastrophic failure. Mathematically, there is a distributional shift between the expert and actual state distribution (open vs. closed loop respectively) because the policy was optimized for the wrong objective function. Ignoring this phenomenon leads to very poor performance at runtime (Ross et al., 2011; Ross & Bagnell, 2010). It is typically mitigated via a supplementary training phase called policy refinement, during which the actual closed-loop performance is assessed and optimized using RL, leaving out the expert. The idea is to render the optimal trajectory manifold attractive, so that the policy not only maintains the system on that manifold but also brings it back if necessary.

Open-Loop Reward Maximization

Atkeson and Liu (2013) has alleviated the distributional shift by aggregating many local *Differential Dynamic Programming* (DDP) policies for various tasks and using the policy of the nearest neighbor in the current state. As already explained, KNN does not scale well with dimensionality. To circumvent this issue, the dataset is refined dynamically based on the error over the induced state distribution. New trajectories are added this way until it achieves good results for any task. This algorithm is called *Trajectory-Based Dynamic Programming* (TBDP). Contrary to the memory of motion presented previously, the prediction is supposed to be already valid, which means that it is not necessary to go through a second optimization phase. It addresses concerns about runtime cost and convergence failure, but not the predictability of the behavior. Moreover, all the policies must be kept in memory as this approach is nonparametric, which is technically difficult for large databases. Despite these limitations, this approach has been validated successfully in simulation on a complex humanoid robot. Levine and Koltun (2013) has shown that it works well as long as the evaluation task is part of the training set. Otherwise, it fails almost systematically due to the poor generalization ability of nonparametric models.

Ross and Bagnell presents another approach called *Stochastic Mixing Iterative Learning* (SMILe). It is known as a *reduction-based algorithm*, which is a kind of IL algorithm that aims to solve a complex task by leveraging the capability of an oracle to provide demonstrations for a series of derived simpler tasks (Beygelzimer et al., 2005), e.g. training a policy using a MPC as an oracle to achieve robust and versatile locomotion. The policy is trained iteratively. At every iteration, the oracle provides expert demonstrations for the current state distribution in closed-loop as a black box, and the policy is updated to mimic them. At the end of the process, the policy is mathematically guaranteed to perform reasonably well under its own state distribution in a closed-loop. Specifically, assuming the action space is countable, the

drop of expected return between the optimal and expert policy is smaller than the squared trajectory length times the expectation that the policy makes a mistake.

This algorithm is an improvement over SEARN (Daumé III et al., 2009) in that it removes the need of sampling complete episodes. Consequently, it demands substantially fewer interactions with the oracle, which is very beneficial because it is by far the main bottleneck. In SMILe, the policy is stochastic and defined as a mixture of trained policy snapshots for all previous iterations. Inspired by ϵ-greedy exploration strategy, at every timestep the action is taken according to the expert with probability $1 - \alpha$, the current policy otherwise. This is important as the first few policies may make many mistakes and visit states that are irrelevant as the policy improves. The efficiency of SMILe has been assessed on a racing game and Mario Bros with a human expert and near-optimal planner respectively. The results are promising in terms of both performance and sample efficiency.

Ross et al. state that SMILe may be unsatisfactory for robotic applications as the resulting controller would be unstable if some policies in the mixture are worse than others. To avoid this issue, they introduce *Data AGGERation* (DAGGER) algorithm that learns a deterministic policy. The cornerstone is that all the expert demonstrations are kept in a large database instead of thrown away. The current policy is updated to minimize the reconstruction error computed on this ever-growing database. During training, the behavior policy is still stochastic and follows the ϵ-greedy strategy. DAGGER is also closely related to TBDP (Atkeson & Stephens, 2008). While both SMILe and TBDP are building up a non-parametric model for the policy consisting in fetching the nearest neighbor in a database, DAGGER trains a function approximation, i.e. an *Artificial Neural Network* (ANN). Interestingly, the required number of iterations scales nearly linearly with the effective horizon of the problem. DAGGER was compared to SMILe on the same synthetic benchmark as before and the performance gain is significant. Notably, the failure rate at runtime goes down to zero while it was not the case for SMILe. However, DAGGER still assumes that the expert demonstrations are optimal for all states. In practice, they rely on DDP to generate them, so they are optimal locally at best and no longer valid if the state has drifted away significantly. Levine and Koltun have illustrated this phenomenon on a 2D walker. Recomputing the next optimal action for every visited state in a MPC fashion would prevent such an issue. Yet, the computational cost would be quadratic in the trajectory length, and it would still not guarantee optimal demonstrations as the solver may find local minima or even completely fail.

Closed-Loop Return Maximization

Levine and Koltun overcame the limitations of DAGGER by maximizing the actual return of the policy, making it less vulnerable to suboptimal experts. Their method is called *Guided Policy Search* (GPS) and comprises two additional mechanisms. First, the oracle adapts itself in relation to the current policy. Originally, the objective of the DDP is to find the policy that maximizes the expected return under its own state distribution. Here, the objective is modified to maximize the expected return

under the state distribution induced by the current policy instead. This is equivalent to adding a bonus term to the original objective function that promotes generating trajectories with high log-likelihood according to the current policy. Secondly, the current policy is not trained to blindly mimic the expert, but rather to maximize the expected return under the current policy in an off-policy way with weighted *Importance Sampling* (IS). Hence, the trained policy must be stochastic. It has isotropic multivariate Gaussian action distribution, so it can exploit the reparametrization trick: the mean is represented by an ANN and the standard derivation is fixed. After learning, only the mean is extracted to get a deterministic policy at runtime.

Levine and Koltun claim that bootstrapping learning with naive imitation on a pre-computed dataset of expert demonstrations is critical. Indeed, DAGGER wastes a lot of effort on states where the DDP policy is not valid for inherently unstable systems such as a walker. Pre-training significantly improves the overall performance and sample efficiency by avoiding falling into poor local minima from the start.

Apart from improving the robustness of the policy, adaptation of the oracle is absolutely necessary when the initial samples cannot be reproduced by any policy, typically when it acts differently in similar states as a result of multi-modality. Among all possible solutions, it gives priority to the one that is the highest log-likelihood according to the current policy. The policy is a function approximation, so it is important to limit its expressiveness to ensure high generalization ability and avoid overfitting. However, it implies that the policy would not be capable to reproduce accurately all possible trajectories. Adaptation ensures that DDP outputs only solutions that can be accurately reproduced, so that at the end of the process, the final policy matches exactly the expert demonstrations. Unlike TBDP, policies trained with GPS generalizes well to tasks that are not part of the training set, e.g. training a humanoid robot to walk on constant slopes and evaluating on random terrains. Stochastic off-policy learning is known to be unstable because the variance of the *likelihood-ratio gradient* (LR) scales poorly with the dimension of the action space.

Kahn et al. (2017) introduce the *Policy Learning using Adaptive Trajectory Optimization* (PLATO) algorithm, which is a variant of GPS replacing DDP with MPC. For a DDP-based oracle, the expert demonstrations are provided by a finite set of locally-optimal controllers for different tasks, each of them bringing the state of the system back to its underlying nominal trajectory. Whereas for MPC-based oracle, the expert demonstrations are all coming from a unique globally-optimal controller for a given OCP that may involve an approximate model of the system. Although a limit cycle may arise, there is no nominal trajectory per se since the OCP is repeatedly solved based on the current state. Assuming the exact model is considered, MPC-based methods are expected to outperform their DDP-based counterparts as the oracle truly acts optimally over the whole state space. Kahn et al. have demonstrated the superiority of their algorithm over GPS for aerial navigation tasks in simulation. The agent learns faster while experiencing substantially fewer catastrophic failures during training. However, this approach is not tractable for complex problems in which solving the OCP is computationally demanding because it must be done repeatedly at every timestep. Typically, in legged robotics, solving the OCP

takes at least 20ms using state-of-the-art methods (cf. section 4.1.1). It is about 100 times slower than simulation (Lee et al., 2018; Todorov et al., 2012), thereby slowing down training by the same factor.

4.1.3 Trajectory Planning and Function Approximation

Sim-to-Real Transfer

Imitation approaches are effective in simulation but are likely to fail once transferred to reality. This is mainly due to the domain shift between simulation and reality (Haider et al., 2021; Hays & Singer, 1989; Zhao et al., 2020). GPS generalizes to unseen tasks but not to model uncertainties. How to learn such a robust policy is an open question. The most famous methods to handle the reality gap are domain adaptation and domain randomization.

Domain adaptation is useful when there is a mismatch between simulation and reality regarding the mapping from state to observation. The most famous application is learning to perform manipulation tasks based on raw pixels from cameras. Learning these tasks entirely in simulation is substantially less expensive in price and time than resorting to real experiments, but simulated data are usually not realistic enough to transfer skills to the real device. One option is to extract domain-invariant features that are rich enough to perform the task properly. Another way is to train an additional network to fake real data from simulated data as faithfully as possible. These two approaches are not mutually exclusive and can be combined. Bousmalis et al. (2018) was able to grasp objects in reality by doing so.

Domain randomization consists in modifying randomly the original transition probability function of the world throughout the learning. Various aspects can be altered: the parameters of the physical model, the motors and sensors, the feedback loop, or the characteristics of the external environment. As opposed to domain adaptation, it does not require knowing precisely the target environment. (Tobin et al., 2017) has used this technique to perform grasping in reality without relying on any real data or prior knowledge during learning. It has proven effective countless times in robotic applications. However, it is bounded to learn a suboptimal policy because it optimizes the performance on a much larger domain than the one on which it will be applied in practice.

Identifying online the domain to adapt the policy accordingly is expected to yield better performance as it does not suffer from this limitation. Learning a unified control strategy performing some sort of online system identification internally would require the network to have a memory or a history of observations as input, which is notoriously very difficult to train by RL. Yu et al. (2017) overcome this limitation by considering separately system identification and control using two networks connected in cascade. The first one is responsible for estimating the set of unknown parameters from a history of past observations. Its estimate is then forwarded to a policy network alongside the current observation. These two networks are trained separately using supervised and reinforcement learning respectively. Following this

explicit decomposition, a simple feedforward policy can theoretically perform optimally as long as the true value of the parameters is accurately estimated. Peng et al. (2020) extend this approach to further improve the performance. A variational autoencoder is used in place of a classical FNN for system identification, which allows the estimation of the unknown parameters in a low-dimensional latent space. Seeking to reduce the dimensionality makes sense because different parameters may have a similar effect on the transition probability function. It has the benefit to improve the robustness to uncertainties and ease feature extraction by the policy.

These extra inputs driving the policy can be interpreted as privileged information (Vapnik & Vashist, 2009). During training, they are perfectly known, but they must be estimated at runtime. Alternatively, Miki et al. (2022) have unified identification and control as a black box via an additional teacher-student training phase. More precisely, another policy without privileged information as input is trained to mimic the first one. To make this possible, the new policy must keep track of past events one way or the other. It can be implicit or explicit, e.g. using a *Recurrent Neural Network* (RNN) or forwarding as input a partial history of past observations respectively. They obtained very impressive results in reality on a quadrupedal robot.

The feedback loop introduces a coupling at runtime between the network estimating of the unknown parameters and the policy itself for which neither of them has been trained. The feedback loop is likely to be unstable in reality, leading to catastrophic failure of the system if the policy is not made robust to this side effect. This concern is not addressed by any of the previous approaches.

A similar approach is meta-learning, also referred to as learning to learn. It is more promising as online adaptation is much faster and reliable (Arndt et al., 2020). The basic principle is to train a policy that can be easily adapted to any domain. Formally, it is an average policy in parameter spaces that can be specialized to a given domain within a very few gradient descent steps. The idea is enticing but the existence of such a set of parameters is questionable for complex problems such as bipedal locomotion, specifically when the tasks already encompass many user-specified decision variables for various environments. The results were satisfactory for learning to shoot a hockey puck to a target location with a real robotic arm. It is hard to conclude if it can be applied reliably on a more complex and intrinsically unstable system such as a humanoid robot. If it fails right from the start, then no data can be collected and it cannot learn anything.

Leveraging Expert Policies

Policies trained using RL algorithms have been able to solve complex tasks in reality after training in simulation only, e.g. rough-terrain quadrupedal locomotion (Miki et al., 2022) and bipedal stair traversal without exteroceptive sensors (Siekmann et al., 2021b). These approaches systematically rely on domain randomization to deal with the reality gap. One of the main drawbacks of this approach is the great difficulty in enforcing specific features for the induced trajectories. This is especially problematic for lower-limb exoskeletons, where the gait pattern is supposed to be

natural and conformable for the user. Inverse RL aims at tackling this limitation by inferring the reward function automatically from expert demonstrations, but it scales very poorly with the dimensionality of the state and action spaces (Arora & Doshi, 2021). Another major issue is unpredictable close-loop behavior and lack of robustness certificate. Preliminary works have been done on this point but much remains to be done (Wu et al., 2022).

A more conservative approach is to completely ignore the control aspect during learning by merely predicting nominal trajectories and then relying on a classical off-the-shelf LLC to track them. It seems reasonable because an adequate and thoroughly tested LLC is readily available in most robotic applications. The main advantage is to put less pressure on learning since the reality gap does not have to be handled at this stage. In fact, this responsibility is only transferred to the LLC. The latter is implicitly expected to render the trajectories attractive by applying minor correction if necessary, which is typically out-of-reach of basic *Proportional-Integral-Derivative controllers* (PIDs) for floating base robots. Consequently, the LLC has to be sophisticated since its capability is now bounding the overall performance. Moreover, if the closed-loop behavior is not satisfactory, then it is not clear whether the planning formulation or the controller must be revised because of the coupling between them.

Daumé III et al. (2009) suggest that policy learning could be used to generate trajectory. Indeed, trajectories are sequences that can be viewed as structured predictions whose underlying data structure is the policy itself. If the policy predicts the target torque, then a simulator is used to integrate the dynamics and output the future observation. It is not even necessary to rely on a simulator if the policy predicts the target state. It is enough to simply pass on earlier predictions as inputs for future ones, just like the unfolded representation of RNNs. This leads to a degenerate form of IL where the system dynamics is deterministic and trivial. Trajectories computed using only single-pass, greedy predictions with these methods achieve competitive results versus trajectory learning (Ross et al., 2011). Nevertheless, by reducing policy learning to an indirect approach to generating trajectories, the former loses its main advantage over the latter: the policy itself is confined to open-loop control, which limits the performance compared to closed-loop control.

Kudruss et al. (2015) have realized multi-contact stair climbing on HRP-2 by simply tracking the nominal motion in Euclidean space using *Inverse Dynamics* (ID). Going one step further, Da and Grizzle (2019) were able to achieve robust walking on MARLO using standard regression and basic PID as low-level controller. It is capable to walk on uneven ground using the trajectory network to regulate the pelvis velocity. The user can update the target velocity at any time without losing balance. This approach was motivated by the theoretical study of Wieber and Chevallereau (2006) suggesting that online selection of the nominal trajectory could stabilize walking. However, previous works on imitation indicate that naive regression is not reliable and may not be sufficient in some cases. Apart from the reality gap, it suffers from the same issues as policy learning, including multi-modality and generalization ability. Predicting whole trajectories at once prevents the compounding of uncertainties, so the impact of the reconstruction error on the stability is less dramatic. Nonetheless,

fitting accurately the solutions to the original problem is still important. That is all the more true for kinematic constraints. For humanoid robots, 1 degree of error on the ankle shifts the flying foot by 1 cm vertically. It considerably affects the impact time and causes shuffling that may lead to falling. Likewise, consistency between position and velocity is often assumed by the LLC. If this is the case, then ignoring it would create motor vibrations. It is possible to predict only the position and get the velocity by analytical differentiation of the network, but it would lead to approximation error due to the discretization of the timesteps. The velocity generated by trajectory optimization frameworks is compensating for this phenomenon, so it is better to learn it as if it was an independent quantity. All in all, fitting accurately is always beneficial because tracking would be easier for the LLC.

Leveraging Nominal Trajectories

Each of the imitation methods reviewed in section 4.1.2 relies on a model-based optimal control algorithm for the oracle, namely DDP or MPC. It guarantees the attractiveness of the nominal trajectory manifold without extra burden. However, having to provide a controller instead of plain trajectories as demonstrations is very limiting. First, DDP faces difficulties to handle constraints. It is only recently that an extension based on the augmented Lagrangian has been proposed for equality constraints, while inequalities are still out of reach (Kazdadi et al., 2021). Secondly, it prevents leveraging effective trajectories obtained experimentally without numerical optimization, e.g. motion capture recordings or human demonstrations by moving the joints of the robot manually in transparent operation mode. Finally, it is incompatible with any existing motion planning framework that generates optimal trajectories without an accompanying stabilizing controller.

Mordatch and Todorov (2014) introduced an algorithm called *Distributed Trajectory Learning* (DTL) that basically follows the same principle except that the oracle provides nominal trajectories directly, as raw temporal sequences. It enables using any trajectory planning method and not just DDP. The direct collocation framework in particular is numerically robust and scalable (cf. appendix A.3.1). Notably, it has proven its ability to generate natural motions for humanoid robots in complex scenarios (Gurriet et al., 2018; Hereid & Ames, 2017; Huynh et al., 2021). Its application to Atalante is presented thoroughly in appendix A. Like Levine and Koltun (2013), they are still learning a policy, and the OCP is adapted to seek a trade-off between the optimality of the demonstrations and the capability of the policy to mimic them. Formally, a penalty term is added to the original OCP to promote finding solutions consistent with the policy. It results in a joint optimization problem that is solved iteratively with *Alternating Direction Method of Multipliers* (ADMM) (Boyd, 2010). A special effort is required to ensure the attractiveness of the trajectory manifold under the policy. To this end, they derive a *Linear Quadratic Regulator* (LQR) from DDP by linearizing locally the system dynamics. It specifies the gradient of the policy to behave the same, which simply translates to a second term in the regression loss. The addition of the penalty term to the OCP relates the solutions for different

tasks through the function approximation. Combined with the joint optimization, it enables performing global optimization over the task space to encourage finding solutions that all belong to the same smooth trajectory manifold whose regularity complies with the function approximation. This approach is effective against multi-modality and ensures good generalization ability. However, adding a penalty term is not enough to provide mathematical guarantees. The weighting factor must be tuned manually, and it is impossible to cancel out the reconstruction error completely. It has only been assessed on toy models, a 2D walker with 7 *Degrees of Freedom* (DoFs) being the most advanced one. However, this approach is likely to fail once transferred to a real device since they do not address the reality gap.

4.2 Related Work on Robust and Safe Control Policy

4.2.1 Classical Control Methods

Proportional-Integral-Derivative Controllers

PIDs are widely used in industrial applications (Goswami & Vadakkepat, 2019). They are commonly used to output a command torque proportional to the tracking error in position and velocity for each motor separately (cf. section 2.2.3). It renders the nominal trajectory locally attractive at the motor level by actively compensating for small disturbances and preventing the compounding of uncertainties. Although PIDs are mathematically simplistic in isolation, the coupling between them through the mechanical structure gives rise to complex dynamical behaviors, let alone motor backlash or delay. In practice, using basic PIDs at the motor level is well-suited for any application without human-robot interaction or a need for adaptive compliance. This way, stable walking on flat ground has been realized on the exoskeleton Atalante, when supplemented by the heuristics of switching directly to the next nominal step as soon as the flying foot hits the ground instead of waiting for step completion (Huynh et al., 2021). However, it does not give any guarantee regarding the attractiveness of the whole-body trajectories for free-floating base or under-actuated robots. To get around this limitation, PIDs can output targets in a feature space, followed by some weighted ID to compute the command torques that are supposed to realize these targets. Apgar et al. (2018) validated experimentally this approach on Cassie for a feature space gathering the pose of the feet and pelvis.

Simplified Models

It is necessary to actively maintain balance based on actual stability criteria instead of blindly trusting the supposedly intrinsic stability of the nominal motion. Upright standing offers a large variety of recovery strategies that can be leveraged in case of emergency to avoid falling, among them: ankle, hip, stepping, height modulation, and foot-tilting for any legged robot, plus angular momentum modulation for humanoid robots (Yuan et al., 2020; Boer, 2012, Chapter 5). For moderate perturbations, in-place recovery strategies controlling the *Zero-tilting Moment Point* (ZMP)

(Hyon et al., 2009), the centroidal angular momentum (Stephens & Atkeson, 2010), ankle control (Mesesan et al., 2021) or the foot tilting (Li et al., 2017) are satisfactory. Most of these approaches involve a very crude model to approximate the centroidal dynamics, namely the *Linear Inverted Pendulum Model* (LIPM) (Kajita et al., 2001). To handle stronger hazards, Kajita et al. (2003) were the first to propose an unconstrained LQR controller based on ZMP trajectory generation for Asimo. This type of LQR admits a closed-form solution, so it is very cheap to evaluate.

Later, Wieber (2006) improved the previous formulation to enhance robustness to external pushes by tracking into account box constraints on the ZMP. Such linear MPC cannot be solved analytically but the solution can be found very efficiently using QP and has strong convergence guarantees. At this time, it was only investigated in simulation for HRP-2. The latter was able to withstand pushes having a normalized force impulse – force magnitude times duration over robot weight – of $0.3\mathrm{N\,s\,kg^{-1}}$.

Linearity of the MPC imposes simplifying assumptions and restricting constraints. Getting rid of this prerequisite would free the positions of the feet in the OCP whereas it was prescribed by the nominal trajectory, extending the set of strategies that can be leveraged by the controller to maintain and recover balance. As long as the MPCs remains unconstrained, it can still be solved efficiently and invariably converges to a solution, which is no longer globally but locally optimal. Wittmann et al. (2015) and Feng et al. (2016) concurrently demonstrated this approach in reality for flat-foot walking on the humanoid robots LOLA and Atlas respectively. Built upon the work of Kajita et al., they formulated unconstrained non-linear MPCs capable of foot placement strategies. Remarkably, Atlas was able to recover from pushes of $0.5\mathrm{N\,s\,kg^{-1}}$ in single support. In both cases, the MPC is running at 500Hz and combined with PIDs at the motor level for tracking the target trajectory.

Alternatively, Englsberger et al. (2011) have showed promising results on real robots a few years earlier using admittance control (cf. section 2.2.3). Krause et al. (2012) extended this method for taking into account a finite horizon rather than exclusively the current time, which has been validated experimentally on the DLR-Biped robot. In their work, they assume that the nominal footsteps are realized. As previously mentioned, this condition is necessary to make the constraints linear. It is cheap to solve numerically, but it prohibits shifting the footsteps to improve stability or doing extra recovery steps in case of emergency.

Mesesan et al. (2021) were able to achieve state-of-the-art push recovery capability using only reactive control. Their approach is designed to rely exclusively on ankle control at first, then reactive stepping based on the *Divergent Component of Motion* (DCM) if necessary. Despite being purely reactive, this controller allows for foot placement strategies. More precisely, if the swing foot strikes the ground at the end of the horizon, then it can be seen as an extension of the projected support polygon (cf. section 2.2.2) with proper mathematical foundation and consideration of the impact timing. This approach was assessed in simulation on the humanoid robot TORO. It enables to recover from pushes of up to $0.5\mathrm{N\,s\,kg^{-1}}$ in single support. The control can run faster than 1kHz, which is enough for direct torque control.

Whole-Body Model

All of these methods still rely on variants of the inverted pendulum model to make online execution tractable. For instance, the *Floating-base Inverted Pendulum* (FIP) model has been used to derive a constrained non-linear MPC during the single support phase to walk on rough terrain for HRP-2 (Caron & Kheddar, 2017). A HLC is solving the OCP at 30Hz, while a low-level LQR is running at 300Hz. This kind of hybrid control architecture is important to track accurately the generated trajectories, but also to handle situations where the OCP solver fails to converge. For the double support phase, it relies on a more conservative model-free controller that handles multiple non-coplanar contacts, and it switches back to single support as soon as the non-linear MPC admits a solution. The validity of the model affects the robustness of the controller to some extent that is hard to predict. It is related to sim-to-real transfer. Since reduced performance is expected in practice, conservative behaviors with large safety margins are preferred to prevent instabilities. Ultimately, the motion tends to be restricted to moderately fast and conservative motions. Besides, being able to withstand strong pushes larger than $2.0\mathrm{N\,s\,kg^{-1}}$ requires substantially different recovery strategies as it is necessary to do several recovery steps.

Whole-body MPC as the potential to tackle such complex problems. However, it is impossible to solve such NLP during online execution, even with the most powerful algorithm to date DDP. Furthermore, having a blind trust in the model to design a controller is hazardous. Although it would perform optimally if the model is accurate, it lacks robustness to model uncertainties, which often leads to instability in practice. For example, Vigne et al. (2020a) were unable to achieve stable walking using an inverse dynamics controller that actively compensates the mechanical deformation according to a flexible model observer leveraging *Inertial Measurement Unit* (IMU) data. Yet, they obtained much better performance by using basic decentralized PIDs and adding a feedback term on the deformation angle and velocity. For robustness to the flexible model, the gains of the controllers are tuned using a linearized *Serial Elastic Actuator* (SEA) model for each joint to which they apply a steady-state LQR. Robust MPC (Campo & Morari, 1987; Villa & Wieber, 2017) and more recently stochastic MPC (Heirung et al., 2018) are more comprehensive approaches that addresses model uncertainties, but how to use such techniques on real robotic platforms it is still an active research topic.

State Estimation

The state of the world is indirectly observed through sensors. Depending on their signal-to-noise ratio, it may be necessary to use advanced sensor fusion algorithms, which increases the overall complexity. Beyond this, the state is usually only partially reconstructed. Advanced state observers such as *Extended Kalman Filter* (EKF) (Anderson & Moore, 2012; Lillicrap et al., 1960) and invariant EKF (Hartley et al., 2020) may help, but they neglect any coupling that may exist between the observer and controller dynamics. For linear systems, everything works as one naively expects:

the feedback loop is stable if both the observer and the controller are individually stable. However, it is no longer guaranteed for non-linear systems. In practice, the feedback loop is likely to be unstable unless the observer and controller are tuned to decouple their respective dynamics. Classically, the time constant of the controller is chosen much larger than that of the observer, which limits the overall performance. Policy learning methods such as RL do not suffer from this limitation. Indeed, the agent is aware of the observer and controller that will ultimately be used on the real robot during training, so it can learn to make them work together when possible. The policy does not have to be end-to-end for this to hold, namely responsible for the whole feedback loop including observation and control. In the robotic field, the policy is generally confined to the high-level controller.

4.2.2 Policy Learning Methods

Relation with Optimal Control

RL purses the same goal as explicit MPC and memory of motion, i.e. enabling online operation without hierarchical control architecture, model approximation, or linearization. The output is an end-to-end black-box controller mapping sensors to controls, with little to no understanding of the underlying observation and control strategies. RL is more generic than MPC because it is black-box optimization method. As opposed to IL, RL approaches may be completely problem-agnostic if combined with intrinsic curiosity techniques (Pathak et al., 2017). All it requires in RL is to collect a large amount of data, usually synthetic by running simulations using one of many physics engines readily available. The challenge here is being able to create diverse virtual environments that encompass the reality in order to compel the agent to explore all the desired strategies. Then, it relies on the generalization ability of the policy network to guess what must be done for unseen observations. On the contrary, the underlying OCP in MPC must be well-posed for every practical scenario that the agent may be facing in practice, and only involve analytical constraints that are differentiable. It is arguably harder to achieve than creating environments for RL. For instance, it is only recently that a constraint for the non-coplanar multi-contact stability condition of legged robots has been proposed (Caron et al., 2015).

Several ground-breaking advances were made in RL during the last decade. Learning to solve complex problems without any expert demonstration in an end-to-end manner with RL is increasingly popular and an active research topic. It has shown impressive effectiveness at solving complex continuous control problems for toy models, (Heess et al., 2017; Lillicrap et al., 2016; Peng et al., 2018) being some of the most famous simulation-oriented studies.

Real-World Applications to Legged Robots

Real-world applications are still rare and almost exclusively concern quadrupeds among legged robots. Regarding bipedal robots, a few policies have been trained

and transferred to reality for the biped Cassie or its humanoid upgrade Digit. However, these successes were mainly due to the high accuracy of the model used in simulation rather than special efforts to handle the reality gap. For instance, Xie et al. (2019) were able to achieve stable walking experimentally on Cassie in the absence of disturbances, whereas the policy was trained in simulation without involving any transfer learning technique such as domain randomization. The latter predicts targets motor positions that are forwarded to low-level PIDs assuming zero target velocity. Such an easy transfer cannot be expected for less agile and more anthropomorphic humanoid robots. It is even worse for Atalante because of the completely unknown user dynamics. Still, it is worth mentioning the methods that worked on legged robots even if applying them to Atalante is not straightforward.

Hwangbo et al. (2019) learned dynamic and agile maneuvers on the quadruped ANYmal which were smoothly transferred to reality. Domain randomization is used to sample the model parameters, while domain adaptation enables them to reproduce the behavior of the real actuators in simulation. The transfer function of the actuators is trained using supervised learning with collected data on a test bench. Very recently, they stopped relying on domain adaptation in favor of privileged information estimated online as a black-box (Miki et al., 2022).

More recently, (Siekmann et al., 2021b) went further and manage to learn stair traversal walking on Cassie using only proprioceptive sensors. It also takes into account a few user-specified decision variables as input, i.e. the desired forward, lateral and rotational speed in the world plane. Domain randomization enables dealing with disturbances for which it was never trained, including pushes. Although effective, it leads to more conservative behaviors than necessary. Note however that, used incorrectly, this technique can prevent learning completely: it is indeed fundamental to increase the variability progressively (Li et al., 2021). Alternatively, the stability can be improved by predicting high-level features for a model-based controller (Castillo et al., 2021), but it bounds the overall performance and is harder to analyze.

In practice, the state of legged robots is often partially observable. The agent would significantly underperform or even fail to solve the task if a basic feedforward network is used as policy and only the current observation is provided. This issue can be addressed by feeding the policy with a history of previous observations instead (Li et al., 2021). Alternatively, a network having memory capability by keeping track of some internal state (either implicitly or explicitly) may be used, e.g. RNN (Siekmann et al., 2021b). Both approaches are theoretically equivalent in terms of optimal performance. They are still relevant if the state is fully observable because it improves robustness to noise and model uncertainty. However, they make training significantly harder (Miki et al., 2022). Which one is the most appropriate depends on the application and the actual network architectures being compared.

Robust locomotion and standing push recovery for humanoid robots using RL falls short of expectations on real devices. Learning walking on flat ground without nominal trajectory was demonstrated on a mid-size humanoid (Rodriguez & Behnke, 2021). However, the motion was slow and unnatural, with limited robustness to external forces. Regarding standing push recovery, promising results were obtained

in simulation by several authors concurrently (Ferigo et al., 2021; Yang et al., 2020; Yang et al., 2018). Yet, the emerging behaviors were often unrealistic and hardly transferable to reality. Moreover, safe and predictable behavior must be enforced or at least strongly promoted. It is especially important for Atalante where human-robot interaction is ubiquitous. None of the previous methods address this concern.

Dynamically Stable Natural Motions

In RL, the agent is only guided by the reward function, so it is basically free of trying any strategy to solve the task. Thus, the resulting policies typically exhibit richer and more exotic behaviors than using classical model-based approaches. However, training is often unstable and struggles to converge as it easily gets stuck in local minima. As with any other black-box optimization, a very large amount of data is required. The vast majority of RL algorithms maximize the expected return without considering hard constraints. The only way to enforce them is the penalty method, i.e. by adding extra terms to the reward function. Thus, the constraints have no guarantee to be satisfied, and tuning the weight of every reward component is very tedious. It makes it very hard to enforce specific constraints on the overall agent's behavior. That may be fine in some cases, but not physical devices that are supposed to interact with humans such as bipedal exoskeleton.

More generally, learning policies from scratch is difficult. Thus, it is desirable to leverage any prior knowledge that may already be available. If it comes in the form of expert demonstrations. In which case, IL seems more appropriate than RL. A naive approach would be to train first the policy with the sole objective to accurately mimic the expert via IL, then to refine it to improve robustness and transfer to reality via RL. This kind of bootstrapping approach may be effective on simple tasks, but it does not work on long-running tasks with a sparse reward landscape because of domain shift. Uchendu et al. (2022) uses an expert policy to provide initial guidance. It is only used to collect meaningful data with non-zero rewards, so it does not have to be optimal. Actions are taken under the expert policy at the beginning of the episodes, then, at one point, it switches to the current policy. Inspired by curriculum learning, they progressively reduce the number of guiding steps according to a fixed schedule. This method lowers the sample complexity for non-optimism exploration methods from exponential to polynomial with respect to the horizon. They demonstrate performance improvement on a set of realistic simulated robotic tasks. It outperforms both pure IL and RL algorithms in the small-data regime. Still, this method would fail if the trained policy is not capable to mimic expert demonstrations. Moreover, it does not help to get tight control of the final behavior of the policy since it relies solely on RL at the end of the training.

Similarly, GPS or DTL are able to learn a function approximation of the solution to a constrained non-linear MPC over a task space, no matter if the problem is multi-modal. Relying on whole-body MPC or trajectory optimization to generate expert demonstrations is double-edged. On the one hand, it is sample efficient since it is a first-order optimization method, and it makes it easy to satisfy the behavior

of the policy by tuning the underlying OCP. Hard constraints and specific features are straightforward to enforce this way. It is very important for exoskeletons since the gait pattern is supposed to enhance rehabilitation of the patients. On the other hand, coming up with a OCP formulation that is flexible enough to fit any task of interest requires a lot of expertise. For locomotion problems, the OCP must be parametrized with high-level features of the environment such as a segmentation map of the environment. Moreover, the dynamics and constraints must be differentiable, which is not always true, e.g. in the unilateral contact model for legged robots. Besides, GPS and DTL are robust the distributional shift but not to the reality gap. This could be alleviated via domain randomization. DTL relies purely on expert demonstrations for training, without any actual simulation. Therefore, the world parameters such as the ground profile must be randomly sampled in the OCP directly. It implies an even more generic OCP formulation, which is very challenging. On the contrary, the expert demonstrations in GPS are only guiding samples while the actual return is being optimized by the policy. So it is straightforward to perform domain randomization directly in the simulation while the OCP is still solved for the theoretical environment. The policy would have to find a compromise between fitting the theoretically optimal policy and being robust to domain shift.

For both GPS and DTL, it is possible to provide raw sensor data as a partial observation of the real state, while the oracle is having access to privileged information. This enables fusing state observation, system identification, and control as a unified network, whereas Miki et al. (2022) had to split training into two stages. Peng et al. (2018) have synthesized extremely natural motions on a simplified humanoid model by leveraging motion capture recordings. They have no interest on transfer to reality in this work. The main focus was on combining RL and expert demonstration to generate physically valid motions. The nominal is used to define a set of reward components that are normalized and aggregated together. This approach is orthogonal to previous work on IL. Later, Peng et al. (2020) extend it to handle the reality gap. It involves both domain randomization and online estimation of physics parameters. It was applied to the quadrupedal robot Laikago.

Concurrently, Li et al. (2021) train a policy that predicts the target motor positions for decentralized low-level PIDs. First, they generated a finite database of optimal trajectories over a low-dimensional task space. Then, they forward to the policy the nominal state at the current time for a given set of user-specified decision variables. It uses a neural network with residual structure, namely the output of a trainable *Multi-Layer Perceptron* (MLP) is added to the current nominal state to obtain the actual target motor positions. They claim that such a policy is easier to train since applying minor corrections to the nominal trajectory is generally sufficient. Indeed, the output of the MLP should be almost zero most of the time, except when an unexpected event occurs or during transitions from one trajectory to another following the update of the decision variables. They validated their approach experimentally on the bipedal robot Cassie.

As Peng et al., most reward components are related to reproducing the nominal trajectories. Inverse RL aims at inferring automatically the right reward compo-

nents to reproduce the expert demonstrations without having to tune each of them manually. However, it does not scale well with the dimensionality and the existing techniques only support discrete action spaces (Arora & Doshi, 2021).

4.2.3 Predictability and Safety in Reinforcement Learning

Constrained Policy Optimization

Being able to enforce hard constraints explicitly in RL is an active research topic. It enables RL to tackle the exact same OCP than MPC but without needing the analytical gradient. As a result, it would bring the same flexibility and tight control of the agent's behavior. Achiam et al. (2017) introduced *Constrained Policy Optimization* (CPO) based on *Trust Region Policy Optimization* (TRPO) (cf. section 3.2.3). This method is effective but limited to discounted cumulative constraints over complete episodes. The distance of the optimal policy from the boundary of the feasible domain is governed by the reward function and termination conditions in a way that is hard to predict. This is problematic as it means that the safety thresholds of the constraints and the reward function must be jointly hand-tuned by trial and error, which is a highly demanding task only suitable for domain-specific experts.

Building upon CPO, Van Havermaet et al. (2021) present *Masked CPO* (MCPO) to resolve the safety-performance trade-off without needing to hand-tune the reward. Rewards are considered unsafe if associated with transition steps for which the constraint is violated. If so, they are masked out when computing the advantage estimator. Thus, the agent has no incentive to go beyond the limit, which breaks the coupling between the safety thresholds and the reward function. Although effective, this algorithm is data-intensive, and the policy update is very costly.

To circumvent this limitation, Chow et al. (2019) post-process the unconstrained action using a so-called *safety layer* (Dalal et al., 2018). In their case, it is a linear projection onto a safe hyper-plane that ensures the policy always predicts actions satisfying the desired constraints. This approach avoids having to project the training parameters onto the feasibility set induced by the constraints. Unlike previous methods that are deeply grounded in TRPO theory, it can be plugged into any policy gradient method, both on- and off-policy. They chose *Proximal Policy Optimization* (PPO) as on-policy algorithm since it is much cheaper to update the parameters than TRPO and easier to implement. They demonstrated the effectiveness of their method on a real-world robot for an obstacle-avoidance problem.

Another option is the barrier function method, which allows a compromise between penalty and hard constraints. It can be viewed as a special kind of penalty that is added to the surrogate objective directly. It requires having access to the analytical gradient of the constraints. In turn, the optimal policy is mathematically guaranteed to satisfy the constraints in contrast to adding gradient-free penalty terms in the reward. This approach is compatible with any policy gradient algorithms, just like the L^2-norm regularization. The constraints may be violated throughout the training, depending on the actual barrier function being employed. This relaxation

helps to speed up converge and achieve better final performance compared to methods handling hard constraints such as CPO. Liu et al. (2020b) drew from *Interior Point Optimization* (IPO), which uses the logarithmic barrier function in particular. Unlike IPOs, the slope of the barrier must be adjusted manually. Yet, doing so is as easy as for MCPO because the penalty marginally affects the return as long as the constraint is satisfied, thereby avoiding any coupling with the reward function.

Presumably, Liu et al. deal with ill-conditioning by clipping the violation of the constraints before going through the barrier function, but this point is not discussed explicitly in their work. No gradient is backpropagated in such a case, and all theoretical guarantees about the constraint violations are lost. In practice, it works well if and only if the policy must be feasible upon initialization. To get around this limitation, Liu et al. (2020a) split the optimization in two phases. First, they find a feasible policy by exclusively reducing constraint violation by penalizing the reward. Then, they proceed with the same method as before.

All the previous approaches only support cumulative constraints. Liu et al. (2021a) summarize all the existing methods and discuss their pros and cons. It stands out that very few methods can handle instantaneous constraints, one of them being the safety layer. Besides, having to define problem-specific analytical constraints tempers the advantage of RL over classic control approaches.

Alternatively, Ma et al. (2021) relies on spacetime bounds around a nominal trajectory and early termination to enforce hard constraints indirectly. The idea is to stop the episodes prematurely as soon as the state deviates too much from the nominal motion. It is a strong incentive for the agent to avoid such states because it prevents the return from accumulating any longer. The nominal does not have to be valid, but the spacetime bounds must be loose enough for a valid solution to exist. It completely alleviates to need for hand-crafting reward components to mimic the nominal. The bounds are specified in a feature space gathering the relative joint angles, the position of the *Center of Mass* (CoM), the orientation of the pelvis, and the distance between the end-effectors (namely the feet and the hands). Very complex and natural motions were generated using this approach for a humanoid model based on motion capture, including running, doing back-flip, and dancing. They use the same residual structure with low-level PID than Li et al. (2021). The initial state is sampled randomly along the nominal trajectory at a specific time.

Ma et al. (2021) and Park et al. (2019) progressively adapt the sampling distribution to focus primarily on the skills that are currently harder to learn rather than using a fixed uniform distribution, which is sometimes abusively referred to as importance sampling. More precisely, the probability to sample a given nominal motion obeys a Boltzmann distribution (with some offset to be non-zero everywhere) based on an average return estimate. This estimate is updated at every training iteration according to the performance of the current policy.

Smoothing using Low-Pass Filters

Policies trained with RL are usually jerky, with the action varying discontinuously like a bang-bang controller. This impedes performance once transferred to real devices, creates loud noises, and causes premature wear. For exoskeletons, it could even be harmful to the user inside or at least uncomfortable. A naive approach would be to post-process the output of the policy with a low-pass filter to cancel out high frequencies. Although it can be sufficient (Peng et al., 2020), it is not recommended (Mysore et al., 2021). If the filter is only added at runtime, then the feedback loop is likely to be unstable. Indeed, it introduces an unexpected delay that dramatically changes the dynamical response of the system expected by the policy. On the contrary, adding the filter during learning breaks the Markov assumption. If the delay is large, then it is necessary to provide as inputs the history of past events or the internal state of the filter, or at least to use a neural network with memory. Notably, PIDs act as linear low-pass filters, and hence their internal state should be observed unless their gains are high enough to ignore this effect. Moreover, the random process involved in exploration is generally a white noise signal added to the mean action and having the controller update rate as its fundamental frequency. Thus, the filter is going to alter the exploration to some extent that depends on its cutoff frequency. Training may fail because of this side effect.

Smoothing by Monte-Carlo Averaging Post-Process

More generally, policies may behave unexpectedly in some circumstances if nothing is done to prevent it, especially for states that were never observed. This risk is getting more serious with the ever-growing complexity of the network architectures, which enhances their generalization ability at the cost of a higher sensitivity to overfitting. Indeed, modern machine learning models are not rule-based but neural networks, which are facing interpretability or explainability issues: it is notoriously difficult to characterize the rationale behind their predictions. This situation is unacceptable for robots that are supposed to operate in a human-populated environment or even interact with humans, and guaranteeing that a policy would behave properly once deployed in reality is more pressing than ever. This concern has drawn a lot of interest since their vulnerability to adversarial attacks was unveiled Goodfellow et al. (2015) after years of being disregarded. For now, little work has been done on this topic. Several approaches with strong theoretical guarantees have been proposed for neural networks with discrete outputs. However, they are tailored for supervised learning and cannot be translated easily to RL. In the latter case, the prediction error accumulates over time and can make the system unstable in a closed-loop.

Keeping the action within bounds though hard constraints may help but would hardly solve the issue. As a workaround, independent safety checks are generally performed at runtime to make sure everything works as expected. Yet, it remains difficult to be exhaustive. This is problematic as it may damage the device or even hurt people around it if any, not to mention exoskeletons. Alternatively, one may

try to anticipate in simulation every possible situation that may occur in practice. However, it is only applicable to robots operating in controlled environments, which is rarely the case for floating base robots and completely dismissed in the event of human-robot or multi-agent interactions, e.g. the behavior of the user inside the exoskeleton Atalante. Moreover, it implicitly requires knowing the system perfectly, including the dynamic model and hardware defects such as noise and delay, which is unreasonable. Thus, it is more promising to seek to certify mathematically that the policy would behave properly in any circumstance by design.

Cohen et al. (2019) relies on randomized smoothing of a pre-trained classifier to derive a tight condition of the validity of its prediction. The predicted class is the one with the highest probably after disturbing the input with an isotropic normal noise, which is estimated using *Monte-Carlo* (MC) sampling. The noise level determines the tradeoff between robustness and accuracy. Later, Everett et al. (2021) applied this technique to Q-learning and policy-based algorithms with discrete action spaces. The output of the action-value network or the policy is smoothed out respectively, according to the worst-case perturbation. It consists in taking the minimum value in a ball whose radius equals the observation uncertainty after training to choose the action at runtime. They derived a robustness guarantee but limited to linear bounds, and it is not suitable for high-dimensional environments.

Wu et al. (2022) present a unified framework *Certifying RObust Policies* (CROP) that provides robustness certification on both the action and reward. It involves local average smoothing with isotropic normal noise instead of worst-case perturbation, which is closely related to the work of Cohen et al. Still, it is only applicable to discrete action spaces. Indeed, the basic is always the same: verify that the output would not change at all by disturbing the input after smoothing. In the context of RL with continuous action space, such a condition is too restrictive.

Controllability and Lyapunov Stability

What matters is guaranteeing the stability of the system in a closed-loop under a given control policy. Chow et al. (2019) were the first to propose a method based on the notion of Lyapunov functions. Lyapunov functions have a long history in control theory. They provide an effective way to guarantee the global safety of a behavior policy during training via a set of local, linear constraints. More precisely, Chow et al. employ a safety layer to project the action onto a Lyapunov-safe hyper-plane to make sure the policy only outputs actions satisfying the Lyapunov constraints.

Similarly, Zhang et al. (2022) learn a Lyapunov function and use it as a contraction metric to assess exponential stability. From it, they develop stability certificates for unknown dynamical systems under adversarial perturbations. Their method is computationally tractable but its validity is not proven theoretically as they train a function approximation using naive regression without regularity conditions. Whether it is possible to obtain stronger guarantees, for instance by bounding the Lipschitz constant of the network, is an open question.

Alternatively, Mandlekar et al. (2017) and Jin and Lavaei (2020) present two different stability criteria that do not involve learning a function approximation leveraging the notion of controllability. However, they are only applicable to *Linear Time-Invariant* (LTI) systems in principle, which is too restrictive for robotic platforms. Going further, (Castañeda et al., 2020) use RL to learn the state-dependent parameters involved in the classical input-output feedback linearization of any second-order affine autonomous systems (cf. appendix A.2.2), i.e. any real poly-articulated robot. It leverages the knowledge of an approximate theoretical model to bootstrap learning. Thus, the behavior of the policy should be reasonable from the start, ultimately enabling online learning. If the parameters are perfectly estimated, then it provides strong closed-loop stability guarantees. However, it is unlikely to be the case in practice, and how to verify this condition was never addressed in their work.

Lipschitz-Constrained Network Architectures

All the aforementioned approaches are smoothing locally the output of the network after learning via MC sampling. This leads to performance degradation that is usually acceptable in supervised learning but not in RL. The reasoning is the same as before regarding filtering the output of the policy: it changes the closed-loop dynamics of the system without the policy being trained for it. Furthermore, MC sampling adds a substantial computational burden at runtime. It is more promising to seek global smoothness during training. One way is to use neural networks with globally bounded Lipschitz constant. Parseval network is a form of ANN in which the Lipschitz constant is constrained to be smaller than 1 (Cisse et al., 2017). All the singular values of the weight matrices are set to 1, which can be interpreted as a special case of Björck orthonormalization (Björck & Bowie, 1971). After each gradient descent, the closest orthonormal matrix is computed through an iterative application of the Taylor expansion of the polar decomposition. Another approach is spectral normalization (Miyato et al., 2018). The largest singular value of each weight matrix is enforced to be less than 1 by estimating the largest singular value and associated vector using power iteration. It is inexpensive, but it does not guarantee gradient norm preservation. Because of this, the network tends to underuse its Lipschitz capacity.

(Anil et al., 2019) improved upon Cisse et al.'s method. First, the projection is done in the forward pass and not after. This helps to keep the weight matrices close to orthonormal during learning. Then, norm-constrained weight matrices are combined with the gradient norm preserving GroupSort activation function. Together, it actually imposes the norm of the gradient for the entire network and throughput the whole input space rather than solely the global Lipschitz constant. However, such neural networks are much harder to train than classical MLP. Policy distillation may be an option, but it would have little advantage over smoothing after learning. Although effective in supervised learning benchmarks, bounding the Lipschitz constant is too restrictive for continuous control tasks. For bipedal locomotion, bursts of motion are necessary to start recovering balance as soon as possible. Preventing them would significantly shrink the area of capturable states. Indeed, the DCM diverges

exponentially fast, and it is critical to react on time.

Smoothing via Adversarial Robustness

Alternatively, Pattanaik et al. (2018) train a policy to be robust to adversarial noise without bounding the Lipschitz constant. The policy is trained first with any classical RL algorithm. Then, the same algorithm is used once again, but observations are perturbed by adversarial attacks at every timestep. The agent must adapt itself to the task successfully regardless. Mandlekar et al. (2017) rely on the same approach, except that they train the policy from scratch with adversarial attacks. Although the authors are not benchmarking their method against any other, their procedure is expected to outperform smoothing after training. It is also less restrictive than bounding the Lipschitz constant, but it is difficult to infer the relation between both. Nevertheless, the policy would be suboptimal in the sense that it has to deal with perturbations at every timestep while it is very unlikely to happen in reality.

Smoothing through Regularization

Promoting global smoothness during learning through regularization terms seems more appropriate. Regularization terms are penalty costs added directly to the surrogate objective function instead of additional reward components. Their gradient is estimated via *ReParametrization gradient* (RP), which assumes that the state distribution is invariant as if the training data were collected from a dedicated behavior policy in the off-policy setting (cf. section 3.2.2). This formulation has several advantages over additional reward components. First, the gradient estimate has much lower variance (cf. appendix E.7.1). Next, the regularization terms can be negative without giving the agent the opportunity to kill itself as a way to improve the total objective. Finally, they do not compete against the original reward with the same level of priority, which eases the tuning of the penalty factors.

Smoothness regularization makes the overall behavior of the agent both safer and more predictable. It reduces the search space and prevents falling into shallow local minima by focusing on reliable strategies that are globally effective. Thus, it improves both sample efficiency and learning stability. Moreover, such regularization urges the agent to have consistent behavior throughout the state space. More precisely, the state space will be partitioned into a few large clusters, each of them corresponding to a different strategy. Indeed, variations of the observation marginally modify the action within each cluster, while it undergoes large variation when crossing the boundaries. By increasing the penalty factor, the agent would prioritize minimal action and hence prefer a small subset of strategies rich enough to avoid failure but nonetheless suboptimal in relation to the original return. However, a large penalty factor would impede performance to some extent that is hard to predict as it depends on both the reward and transition probability functions. Finding a good trade-off between performance and reliability is a tedious manual procedure.

Even though smoothness cannot replace domain adaptation or randomization, it makes the policy not only more resilient to sensor measurement errors but also to changes in environment dynamics parameters. Besides, smoothness regularization improves the robustness to adversarial attacks, but it does not provide any guaranteed (Zhang et al., 2020). It is not a major flaw since it is out of reach of smoothness-inducing approaches for continuous control tasks anyway. Several formulations have been proposed concurrently in recent years. They all involve some form of spatial regularization. Jin and Lavaei (2020) penalize the Lipschitz constant of a deterministic policy, i.e. the norm of the gradient along the trajectories,

$$L_S = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\|\nabla_\theta \mu_\theta(s_t)\|_2^2\right]. \tag{4.1}$$

This technique was originally termed *double backpropagation* (Drucker & Cun, 1992). Since it does not guarantee that the Lipschitz constant is globally bounded, they rescale the weight matrices of the policy network after every training iteration if necessary. This ensures that the Lipschitz constant stays under a threshold based on a lower-bound estimate. Finally, they add a cost for the temporal finite difference,

$$L_T = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\|a_t - \mu_\theta(s_{t+1})\|_2^2\right]. \tag{4.2}$$

They state that it induces consistency during training: the observation is not supposed to change much between successive iterations and so is the action. Relying on the L^2-norm is a major drawback because it is known to strongly discourage sparsity in favor of uniform distribution. Thus, it prohibits seldom bursts of motion, damaging the performance for tasks such as push recovery.

Independently, Mysore et al. (2021) came up with the following spatial and temporal regularization terms,

$$L_S = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi \\ \epsilon_t\sim N(0,\sigma)}}\left[\|\pi_\theta(a|s_t) - \pi_\theta(a|s_t+\epsilon_t)\|_2^2\right], \tag{4.3}$$

$$L_T = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\|\pi_\theta(a|s_t) - \pi_\theta(a|s_{t+1})\|_2^2\right], \tag{4.4}$$

where ϵ is the noise scale. It does not affect the actual observation received by the agent. Unlike Jin and Lavaei, they do not bound the Lipschitz constant explicitly. According to their study, the spatial cost alone forces what they called "bands on controls": it partitions the state space in clusters and the agent keeps away from their boundaries as long as possible, so that the state never leaves the cluster in which it started. Adding temporal regularization breaks this segmentation and allows the state to cross boundaries from time to time if it is beneficial in the long run. Therefore, combining both spatial and temporal terms is essential to achieve the desired behavior. These conclusions are based on experiments and lack theoretical grounding. The performance of this approach was validated on a quad-rotor drone. It results in 80% reduction in power consumption and consistently flight-worthy controllers. Their formulation penalizes exploration as a side effect, but it can be counterbalanced with an entropy bonus.

Cooman et al. (2021) compared the effectiveness of equations (4.2) and (4.4). They are referred to as supervised and contrastive smoothing respectively. The supervised formulation guarantees that taking action a_t in state s_t can lead to state s_{t+1}. Thus, there is a strong temporal connection between the consecutive actions in the supervised formulation that is non-existent in the contrastive one. Despite this discrepancy, it appears that their effects are roughly identical, at least on *DeepMind Control Suite* (dm-control) continuous control benchmarks.

Shen et al. (2020) introduced another metric for spatial smoothness exclusively,

$$L_S = \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\max_{\epsilon\in B_\infty(\sigma)} D_{\text{J}}(\pi_\theta(\cdot|s_t)||\pi_\theta(\cdot|s_t+\epsilon))\right] \tag{4.5}$$

where $B_\infty(\sigma)$ is the ball of radius σ for L^∞-norm, and $D_{\text{J}}(P||Q)$ is the Jeffrey's divergence $1/2(D_{\text{KL}}(P||Q)+D_{\text{KL}}(Q||P))$. This spatial regularizer is well-motivated. Indeed, Zhang et al. has proven that the value function drop under adversarial perturbations can be upper-bounded by the *Total Variation* (TV) up to a constant. The TV is itself upper-bounded by the *Kullback-Leibler* (KL) divergence, and Jeffrey's divergence matches the KL divergence if the variance of the action is state-independent. However, the inner maximization is fairly costly to evaluate since it is a non-linear optimization problem on its own. It can be interpreted as an adversarial attack. Shen et al. roughly approximate the solution by 10 iterations of projected *Stochastic Gradient-Descent* (SGD). They claim that it works sufficiently well in practice.

Nevertheless, many more powerful methods exist. Zhang et al. (2020) uses a hybrid variant of the first order algorithms *Projected Gradient Descent* (PGD) and *Stochastic Gradient Langevin Dynamics* (SGLD). PGD consists in doing several iterations of *Fast Gradient Sign Method* (FGSM) for the bounded L^∞-norm adversary (Kurakin et al., 2017; Madry et al., 2018). The core idea of SGLD is to add noise to the gradient estimator. It gives,

$$\epsilon^{i+1} \leftarrow \Pi_{B_\infty(\sigma)}\left(\epsilon_i + \alpha \text{ sign}\left(\nabla_\epsilon f(x+\epsilon)|_{\epsilon_i} + \sqrt{2\alpha/\beta}\xi\right)\right), \tag{4.6}$$

where f is the function to maximize, α is the update step, β is the temperature parameter, ξ is an isotropic standard normal noise, and $\Pi_{B_\infty(\sigma)}$ is the projection operator on $B_\infty(\sigma)$. The number of iterations is chosen heuristically. It must be large enough to reach the edge of the ball. This method is cheaper than second-order methods and more reliable than projected SGD as it can escape saddle points and shallow local optima. Mysore et al. proposed to replace the inner maximization with a simple averaging. It is much cheaper to evaluate, and it works well if the number of samples per batch is large, which is often the case.

One last option is to predict the rate of the change of the action and augment the observation with the current action (Chisari et al., 2021). The actual action is obtained via explicit Euler integration. Then, it is enough to penalize the norm of the prediction to promote temporal smoothness. This method was validated successfully in reality on miniature race car driving. All these approaches have never been compared with each other, so it is impossible to guess which one is the most efficient.

Chapter 5

Online Trajectory Planning

Contents

5.1 Trajectory Planning Problem . . . 126
5.1.1 Optimal Control Formulation . . . 126
5.1.2 Vector Representation of the Solutions . . . 127
5.2 Trajectory Learning . . . 128
5.2.1 Naive Formulation as a Standard Regression . . . 128
5.2.2 Efficient Task Sampling and Certifiability . . . 129
5.2.3 Limitations . . . 132
5.3 Guided Trajectory Learning . . . 133
5.3.1 Unifying Trajectory Planning and Learning . . . 133
5.3.2 Trade-off Between Generalization Ability and Optimality . . . 135
5.4 Iterative Solving . . . 136
5.4.1 Alternating Direction Method of Multipliers . . . 136
5.4.2 Proposed Consensus Optimization Algorithm . . . 138
5.4.3 Theoretical Analysis . . . 138
5.5 Structured Prediction with Neural Network . . . 143
5.5.1 Autoregressive Model . . . 143
5.5.2 Generative Single Forward Pass Model . . . 145
5.6 Experimental Validation . . . 148
5.6.1 Toy Problem: Van der Pol Oscillator . . . 148
5.6.2 Application to Atalante: Flat Foot Walking . . . 151
5.7 Concluding Remarks . . . 158

Optimal control unifies planning and control under the same framework. There is no need for any pre-computed nominal trajectory anymore, but it comes with extra challenges. First, the problem must be simplified to run fast enough for online operation, restricting the search space to a subset of feasible motions. Secondly, it is difficult to certify the closed-loop behavior of the robot. It is especially important for bipedal exoskeletons since there is a user inside and people nearby. It is safe and predictable to put them at ease, but also natural to enhance rehabilitation.

These limitations can be alleviated by first generating offline a finite database of nominal trajectories, and then tracking them using classic control methods. Trajectory planning is carried out without any simplification and over a long horizon since computation time is no longer a concern. The controller is supposed to only apply local corrections to compensate for disturbances, so it is sufficient to validate

each trajectory individually to make sure the overall behavior is appropriate in any circumstance. Nevertheless, versatility is somewhat limited because the device has only access to a discrete set of primitive motions at runtime. It is sufficient for limited community ambulation (Huynh et al., 2021), but it must be planned beforehand for every specific task and users cannot go at their own pace, giving the unpleasant feeling of being forcibly manhandled.

We propose to enhance the versatility of trajectory-based control methods by enabling online trajectory planning. This conservative approach strictly improves upon what has already proven successful over the years to bridge the gap between trajectory-based and trajectory-free control methods. It avoids taking the risk of making anything worst and preserves the advantages of decoupling planning and control while empowering the user with fine-grained tuning capability. To this end, we train offline a function approximation to reproduce the solutions of any given *Optimal Control Problem* (OCP) over a continuous task space. It acts as a memory of motions having no impact on the control side. Thus, it does not induce any side effects concerning the reality gap or closed-loop behavior.

Our main contribution in this chapter is *Guided Trajectory Learning* (GTL). It consists in modifying the original trajectory optimization problem to enforce it exclusively generates trajectories that can be perfectly reproduced by a given function approximation when possible and regardless its expressiveness. It results in a consensus optimization problem that is untractable on its own. We overcome this limitation by solving it iteratively via *Alternating Direction Method of Multipliers* (ADMM), which converges to the exact solution under reasonable assumptions. The computational cost is comparable to generating a finite database of trajectories and scales well with the dimensionality of the task space.

5.1 Trajectory Planning Problem

Trajectory planning consists in finding the state and command evolution over time to perform a given task optimally. Then, a lower-level trajectory-based controller is responsible for tracking the resulting trajectory (cf. section 2.2.3).

5.1.1 Optimal Control Formulation

Let us consider a time-invariant time-continuous dynamical system of the form $\dot{x}(t) = f(x(t), u(t))$, where $x(t) \in \mathbb{R}^p, u(t) \in \mathbb{R}^q$ are respectively the state and the command of the system applied at time t, and f denotes the dynamics of the system. Given a task to perform $\tau \in \mathcal{D}_\tau \subset \mathbb{R}^m$, where $\mathcal{D}_\tau$ is a closed compact set denoting the task space, a trajectory optimization problem for such a system can be formulated as

$$y_\tau^* = \underset{y \in \mathcal{C}_\tau}{\arg\min}\, L_\tau(y) \tag{5.1}$$

where L_τ is the total cost function, $y : t \mapsto (x(t), u(t), T)$ is a trajectory of duration T with implicit temporal dependency. Optimal quantities are denoted $\star_\tau^*$ to highlight

the dependency on the task τ. $\mathcal{C}_\tau$ is the feasibility set

$$\mathcal{C}_\tau = \{y \in \mathcal{D}_x \times \mathcal{D}_u \times \mathcal{D}_T \mid 1 \leq i \leq m,\ g_i(\tau, y) \leq 0,\\ 1 \leq j \leq l\ ,\ h_j(\tau, y) = 0\},$$

where $\mathcal{D}_x, \mathcal{D}_u, \mathcal{D}_T$ are closed compact sets. Concretely, $\mathcal{D}_x, \mathcal{D}_u$ embody kinematic and power limitations respectively, while $\mathcal{D}_T$ is related to design choices. The total cost function $y \mapsto L_\tau(y)$ must be bounded on $\mathcal{C}_\tau$ for any task $\tau \in \mathcal{D}_\tau$, and the constraint functions g_i, h_j must be twice continuously differentiable with respect to the task τ.

The generic constraints g_i, h_j on complete trajectories y are usually boundary conditions $h_b(\tau, y(0), y(T)) = 0$ or running constraints $g_t(\tau, y(t), t) \leq 0, h_t(\tau, y(t), t) = 0$ for every time $t \in [0, T]$. The periodicity and duration of the trajectory are examples of boundary conditions, while the dynamics equation and the admissibility conditions are part of the equality and inequality constraints respectively. The task τ gathers a set of high-level features, i.e. the desired step length and forward velocity for walking robots. The total cost adds up a running cost l and a boundary cost l_b, namely

$$L_\tau(y) = \int_0^T l(\tau, y(t), t)\,\mathrm{d}t + l_b(\tau, y(0), y(T)). \tag{5.2}$$

Computing a global minimum for non-linear non-convex problems is unattainable. Finding a local minimum is nonetheless sufficient for most applications, which is guaranteed to exist if $\mathcal{C}_\tau$ is non-empty since it is a closed compact. For this reason, arg min refers indifferently to a global or local minimum in the following for easiness.

5.1.2 Vector Representation of the Solutions

Trajectories must be encoded for the sake of learning. Such a vector representation is not unique. They are all equivalent but some are easier to reproduce accurately than others. Moreover, this encoding must be enforced in trajectory planning to avoid introducing approximation error at this stage.

Continuous functions can be parametrized by the coefficients of a single high-order polynomial. However, this choice is not appropriate because it scales poorly with the trajectory duration, and it gets more or more ill-conditioned as the order increases. Bezier splines or simple piecewise interpolation of discretized signals can be used to get around these limitations. Both are cheap to evaluate, but we prefer piecewise interpolation for portability. Indeed, it is compatible with all optimization frameworks including *Differential Dynamic Programming* (DDP) since it naturally arises when linearizing the system dynamics locally.

Predicting the velocity is avoided by recomputing it according to the original interpolation scheme. It ensures physical consistency between the position and velocity sequences, which is important as it is implicitly assumed by any *Low-Level Controller* (LLC). The position error propagates linearly to the velocity, which is acceptable.

It is convenient to discretize trajectories with a fixed number of breakpoints instead of a fixed timestep. Otherwise, it would be necessary to add or remove breakpoints based on the trajectory duration. Thus, the duration itself cannot be a free

parameter because it would require modifying the structure of the *Non-Linear Program* (NLP) dynamically. Similarly, it would change from one trajectory to another if it is a decision variable, so it would much harder to warm-start the solver. Besides, it enables predicting whole trajectories at once using a single forward pass, which is much cheaper to evaluate and easier to train than an autoregressive model. The resulting sampling frequency would be variable, while the target of the LLC must be updated at a given fixed rate. Thus, it is necessary to resample the discrete trajectory according to the interpolation scheme. In principle, what matters is the prediction error after velocity inference and resampling at the controller update period.

A unique function decodes the raw vector representation to return a discrete trajectory where the state and command are temporal sequences of fixed length L, i.e. $((x(t_1), x(t_2), \dots, x(t_L)), (u(t_1), u(t_2), \dots, u(t_L)), T)$. The entire encoding-decoding procedure, velocity inference, and resampling are omitted in the following for the sake of simplicity. In this context, we can consider without loss of generality that the generated trajectories are discrete from the start.

5.2 Trajectory Learning

Trajectory learning is a degenerated form of policy learning. The agent is trained to predict the complete sequence of states and commands that achieves various tasks under nominal conditions. The observation only contains information about the task, namely some gait features requested by the user such as the speed and steering angle, without any feedback from the environment. This approach is conservative: it completely decouples planning from control so that the reality gap is not a concern, which alleviates the need for *Reinforcement Learning* (RL). Thus, we restrict ourselves to vanilla *Behavior Cloning* (BC) using supervised learning: each individual observation-action pair must be reproduced as accurately as possible, ignoring the closed-loop dynamics. While BC is not suitable for training end-to-end policies, here we assumed that some controller capable of stabilizing the predicted trajectory is readily available. In this work, the expert demonstrations are provided by a motion planning framework that can generate the optimal trajectory for any given task.

5.2.1 Naive Formulation as a Standard Regression

The objective is to replace solving the trajectory optimization problem with a function that can be queried in no time. Classically, this problem is formulated as a standard regression. Let us consider a function $\hat{Y} : (\tau, W) \mapsto y$ parametrized by $W \in \mathbb{R}^n$ and predicting complete trajectories y in one go. The set of parameters W is optimized to approximate the optimal trajectories as accurately as possible, i.e. to minimize the prediction error in expectation over the whole task space $\mathcal{D}_\tau$. It yields,

$$W^* = \underset{W \in \mathbb{R}^n}{\arg\min} \underset{\tau \sim U(\mathcal{D}_\tau)}{\mathbb{E}} \left[R_\tau(y_\tau^*, W) \right], \tag{5.3}$$

where $R_\tau(\cdot, W)$ is the prediction error for task τ. It is usually given by,

$$R_\tau(y, W) = \|y - \hat{Y}(\tau, W)\|_2^2. \tag{5.4}$$

The distribution of tasks is uniform but this choice is arbitrary. Any other distribution could be used to give priority to tasks selected more often or harder to predict in practice. For any given task τ, the predicted trajectory $\hat{Y}(\tau, W^*)$ must be feasible, if not optimal. Although the feasibility constraints of the original trajectory optimization problem are not enforced explicitly, they are satisfied at the limit when the prediction error vanishes.

The expectation of the prediction error cannot be computed analytically. Thus, it is estimated by the empirical prediction error over a finite database of N optimal trajectories $\{(\tau_i, y_i^*)\}_{i=1}^N$, namely,

$$W^* = \underset{W \in \mathbb{R}^n}{\arg\min} \sum_{i=1}^{N} R_{\tau_i}(y_i^*, W). \tag{5.5}$$

In practice, each of these trajectories is generated individually by the same motion planning framework.

5.2.2 Efficient Task Sampling and Certifiability

The number of trajectories N must be minimized since they are costly to generate. Yet, the highest priority is to certify that the function approximation outputs safe trajectories over the whole task space. Specifically, it consists in guaranteeing that the prediction error remains below a given threshold whatever the task. The largest acceptable threshold δ is specific to every application. Its definition for Atalante is presented in section 5.6.2. This problem is common to all trajectory learning methods, including the one that we propose in the next section.

It is assumed in the following that the total prediction error is completely canceled for the training samples. Thus, every prediction is feasible and safe for training tasks, and bounding the maximum deviation apart from them is sufficient to certify that everything would be fine over the whole task space. In addition, the mapping from tasks to trajectories is continuous with bounded Lipschitz constant K for the L^∞-norm. It can be interpreted as a problem of robustness to adversarial attack. How to enforce these conditions in practice will be discussed later in section 5.3.

Mathematically, the worst-case prediction error is upper-bounded by the Lipschitz constant K and the furthest distance from all training samples ϵ,

$$\max_{\tau \in \mathcal{D}_\tau} \|y_\tau^* - \hat{Y}(\tau, W)\|_\infty \leq K\epsilon \tag{5.6}$$

where $\|y_\tau^* - \hat{Y}(\tau, W)\|_\infty$ corresponds to the discrete-time maximal derivation between the optimal and predicted trajectory. It is always possible to define a unique scalar

Lipschitz constant K for the raw vector representation, even when dealing with heterogeneous data, by simply normalizing each element accordingly. ϵ is given by

$$\epsilon = \max_{\tau \in \mathcal{D}_\tau} D_{nn}(\tau), \tag{5.7}$$

where $D_{nn}(\tau) = \min_{1 \leq i \leq N} \|\tau - \tau_i\|_\infty$ is the nearest neighbor distance. Finding the nearest neighbor of each point is an optimal problem on its own. It is called the Voronoi diagram and can be computed iteratively in linear time of the number of points (Dwyer, 1991). It turns out that computing the partition can be avoided if only the maximum distance is of interest, which is the case here. Specifically, it can be reformulated as half of the maximum inter-point distance, taking into account edge effects. It yields,

$$\epsilon = \max_{\tau \in \mathcal{D}_\tau} \left(1/2 \max_{1 \leq i \leq N} D_{nn}(\tau_i), \max_{\tau \in \partial \mathcal{D}_\tau} D_{nn}(\tau) \right). \tag{5.8}$$

Equation (5.8) is much easier to analyze than the original formulation. One can show that the continuous boundary of the task space $\partial \mathcal{D}_\tau$ can be replaced by the finite set of vertices of its convex hull. The limiting distribution of the maximum inter-point distance for *independent and identically distributed* (iid) random variables is related to the *Generalized Extreme Value* (GEV) theory. It has been well studied by many authors (Dette & Henze, 1989; Györfi et al., 2019). For a uniform distribution on a hypercube domain, it has been conjectured from *Monte-Carlo* (MC) studies that,

$$N D_{nn}(\tau_i)^m - a \log(N) - b \log\log(N) - c \xrightarrow[N \to \infty]{} G, \tag{5.9}$$

where G is the extreme-value distribution $\mathbb{P}(G < z) = \exp(-\exp(-z)), z \in \mathbb{R}$ so-called standard Gumbel distribution, and a, b, c are constants that only depends on the dimensionality of the task space $m \geq 3$. Moreover, $a = 1, b = 0$ for $m = 3$ and $a \to 0, b \to 1$ for $m \gg 1$. For high-dimensional task space ($m >= 6$), the edge distance is dominating in equation (5.8), while it is the inter-point distance otherwise. According to Jensen's inequality and using $\mathbb{E}[Z] \leq \mathbb{E}[\max(Z, 0)]$, an upper bound of the expectation of ϵ can be derived that

$$\mathbb{E}[\epsilon] \leq \left(\frac{\gamma + a \log(N) + b \log\log(N) - c}{N} \right)^{1/m}, \tag{5.10}$$

where γ is the Euler's constant. It turns out to be a very tight bound for $m \gg 1$ large. A lower bound can be computed as follows,

$$\frac{1}{2N^{1/m}} \leq \mathbb{E}[\epsilon]. \tag{5.11}$$

This bound is attained when splitting the domain into N partitions of equal size. It depends on the number of samples N and the domain itself. It has no closed-form expression in general. For a hypercube, dividing each dimension evenly in k

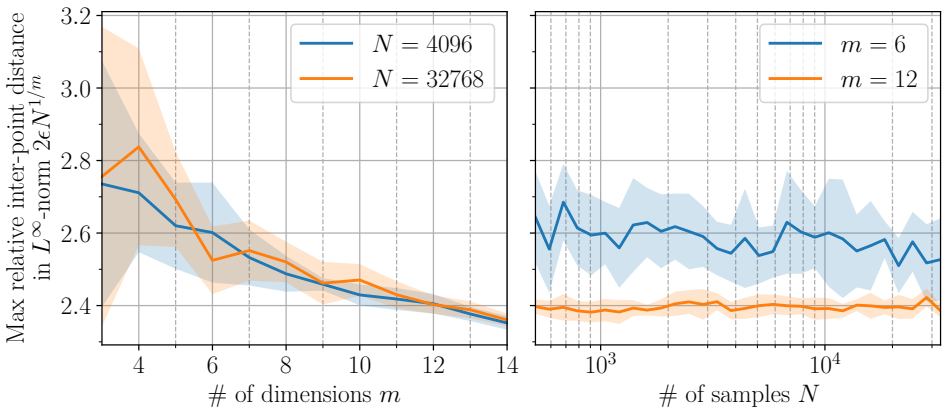


Figure 5.1: Monte-Carlo study of the maximum inter-point distance for N iid uniformly sampled random variables on the unit hypercube $[0, 1]^m$.

sub-intervals and taking samples on the regular grid passing through their middle points is the optimal partition for size $N = k^m$. Typically, it is the case if the task variables are mutually independent. It is not always true, e.g. higher forward speed necessitates larger step length at some point.

These results demonstrate that it is much better to spread the training tasks $\{\tau_i\}_{i=1}^N$ on a regular grid. The MC study in figure 5.1 indicates that $\mathbb{E}[\epsilon]$ is consistently more than double when sampling tasks randomly with uniform distribution. Therefore, having the same worst-case prediction error requires 2^m times more samples, which is extremely large for high-dimension. It is interesting to note that discretizing the task space and sampling points randomly is not helping. It is not surprising since a single hole already doubles the maximum inter-point distance, which is bound to happen if the number of samples is strictly smaller than the size of the grid. Still, discretizing the task space dramatically reduces the variance of the maximum inter-point distance, which is beneficial.

It is clear from equation (5.8) that the allowed variation between training samples is only twice the worst-case prediction error. The minimum size of the partitions of the task space is determined by the Lipschitz constant K of the mapping from tasks to actions and the maximum acceptable prediction error δ, i.e.

$$K\epsilon \leq \delta. \tag{5.12}$$

It means that the number of samples N must be at least $(K/(2\delta))^m$, even if optimally distributed. If the computational cost is prohibitive, then the dimensionality of the task space must be reduced or the mapping must be made smoother. Being able to generate more than 100000 trajectories is unlikely. It corresponds to about 7 sub-intervals per task variable for a hypercube with 6 dimensions. For a single motion pattern, the data overview in section 5.6.2 about flat-foot walking with Atalante suggests that it is sufficient. Indeed, the total variation over the whole task space $\max_{i \neq j} \|\hat{Y}(\tau_i, W) - \hat{Y}(\tau_j, W)\|_\infty$ never exceeds 0.1 rad. The constraint

on the allowed variation prevents mixing completely different patterns, but it is not fitting our preliminary assumptions anyway. In particular, a continuous task space is required, while the type of gait would be discrete information. Nonetheless, the advantage would be questionable. Apart from certifiability concerns, it is important to sample enough data to capture all the information regarding the mapping from tasks to trajectories according to Nyquist-Shannon sampling theorem.

5.2.3 Limitations

The standard regression does not bring any guarantee regarding the total prediction error for the training samples. It is a major issue as it impedes the performance of the real robot. Indeed, trajectory-based control methods assume that the reference is feasible, namely physically consistent, dynamically stable, and satisfying some kinematic constraints – e.g. having both feet flat on the ground in double support. If not, then the local attractiveness of the trajectory manifold is no longer guaranteed, not even in the absence of disturbances. Conversely, non-parametric methods do not face such an issue, but it does not help to enforce the global Lipschitz constant for the mapping from tasks to trajectories. The following lower bound can be defined,

$$\max_{i \neq j} \frac{\|\hat{Y}(\tau_i, W) - \hat{Y}(\tau_j, W)\|_\infty}{\|\tau_i - \tau_j\|_\infty} \leq K. \tag{5.13}$$

Clearly, it would be very large if nothing is done to keep the Lipschitz constant bounded. Thus, it would require a prohibitive number of samples to certify the prediction accuracy over the whole task space. Increasing the number of samples is likely to increase the lower-bound estimate, growing to infinity if the mapping is discontinuous. It is typically the case because the planning problem is a NLP. Finding a global minimum is not guaranteed, and it would converge to a local minimum at best. Hence, it is very sensitive to initialization, and modifying the tasks very slightly may completely change the solution. Not to mention that the problem may be intrinsically multi-modal. At the extreme, the mapping would appear pseudo-chaotic, namely without any structure. As a result, it would be impossible to fit accurately the training samples using a parametric model unless the number of parameters grows with the number of samples. If not, then it would lead to an averaging effect worsening the prediction error compared to using fewer samples paradoxically.

Non-parametric approaches are preventing the averaging effect but are not addressing the root cause of the problem. More precisely, *K-Nearest Neighbors* (KNN) would perform better than any other method because it sticks to the training samples. It ensures the feasibility and safety of the predictions apart from training samples, which is not true as soon as interpolation is involved since the feasible set $\mathcal{D}_\tau$ is non-convex. Besides, the problem can be locally ill-conditioned for some isolated tasks. Trajectory optimization is likely to fail when it happens, preventing fitting accurately the mapping in their vicinity. The work of Dantec et al. (2021) supports this analysis: KNN performs better than a few parametric models for bipedal locomotion with HRP-2. It is not sufficient to be used directly on the real device, so they

are using the predictions to speed-up computations by warm-starting the solver. It enables running fast enough for generating trajectories online without approximation but after simplifying the planning problem if necessary.

Neither the standard regression nor non-parametric methods provide any mechanism to enforce all the prescribed conditions for certifiability at training time. It is tempting to give up on it since it is cheap to check at runtime if the predicted trajectories are feasible and fallback on the aforesaid warm-starting strategy if not. However, not being able to certify the function approximation suggests that the optimal trajectories are hardly predictable, which is problematic for human-robot interaction even if perfectly learned.

5.3 Guided Trajectory Learning

5.3.1 Unifying Trajectory Planning and Learning

The objective is to compute the mapping from tasks to optimal trajectories. NLP can solve the trajectory planning problem (5.1) provided that that task is known in advance. On the contrary, Parametric Programming aims at solving an optimization problem with respect to some decision variables as a function of others. In general, it can be formulated as follows,

$$\forall \tau \in \mathcal{D}_\tau,\ Y^*(\tau) = \underset{y \in \mathcal{C}_\tau}{\arg\min}\, L_\tau(y). \tag{5.14}$$

Pistikopoulos et al. (2007) presents a generic framework to handle a subset of nonlinear convex optimization problems for which the optimal mapping is linear. However, it is too restrictive for real-world applications.

We propose to reduce this problem to a classic NLP in order to solve it. To this end, we introduce a function approximation $\hat{Y}$ and consider the subsequent optimization problem,

$$\begin{aligned} W^* = \underset{W \in \mathbb{R}^n}{\arg\min} \underset{\tau \sim U(\mathcal{D}_\tau)}{\mathbb{E}} \left[L_\tau(\hat{Y}(\tau, W)) \right]. \\ \text{st. } \forall \tau \in \mathcal{D}_\tau, \hat{Y}(\tau, W) \in \mathcal{C}_\tau \end{aligned} \tag{5.15}$$

It admits a solution if and only if it exists a set of parameters W st. the predicted trajectories are in their respective feasible set $\mathcal{C}_\tau$ for all tasks. Assuming it holds true, the mapping tasks to trajectories is given by $\tau \mapsto \hat{Y}(\tau, W^*)$. It matches exactly the solution $Y^*(\tau)$ of problem (5.14) if and only if the function approximation does not induce any coupling between the sub-problems associated with each individual task. Equivalently, the Lipschitz constant of the function approximation must be larger than the one of the original optimal mapping $Y^*(\tau)$. Typically, it is guaranteed for non-parametric models since their Lipschitz constant can grow unboundedly. If not, then the solution is different since minimizing the sum is no longer equivalent to minimizing each term separately. Yet, it only has an effect in terms of optimally, not

feasibility, which is the main concern. Anyway, modifying the optimal trajectories originally generated by problem (5.1) to some extent is unavoidable to enforce the necessary conditions for certifiability.

The first condition was to cancel out the total prediction error for training samples in the case of standard regression, but it can be relaxed to only require satisfying the feasibility constraints. The second one was to regularize the global smoothness of the mapping from tasks to trajectories by bounding its Lipschitz constant over the whole task space. Even though the solution to the original planning problem may change completely for arbitrary small variations of the task, it is sufficient to bound the Lipschitz constant of the function approximation $\hat{Y}$. How much the original solution is altered by this constraint is directly related to how restrictive the threshold K is compared to the original problem. This side effect is discussed thoroughly thereafter.

Computing quantities and enforcing constraints on a continuous task domain is theoretically possible via basis functions just like the temporal dimension. However, it has very little advantage over discretizing the task space with a finite collection $\{\tau_i\}_{i=1}^N$. First, it does not hinder certifiability as for standard regression. Next, the constraint on the Lipschitz constant determines the maximum constraint violation, and the total cost is barely modified if the number of samples is large. Formally, the trajectory planning cost function L is continuously differentiable and the task space is compact. Thus, the expectation can be estimated via MC integration,

$$I_N = \frac{1}{N} \sum_{i=1}^{N} L_{\tau_i}(\hat{Y}(\tau_i, W)), \tag{5.16}$$

where N is the number of samples and $\{\tau_i\}_{i=1}^N$ are sampled uniformly on $\mathcal{D}_\tau$. According to the law of large numbers,

$$I_N \xrightarrow[N\to\infty]{} \mathop{\mathbb{E}}_{\tau\sim U(\mathcal{D}_\tau)} \left[L_\tau(\hat{Y}(\tau, W)) \right] \tag{5.17}$$

$$\mathrm{Var}(I_N) = \frac{1}{N(N-1)} \sum_{i=1}^{N} (L_{\tau_i}(\hat{Y}(\tau_i, W)) - I_N)^2. \tag{5.18}$$

The standard deviation of the cost estimator I_N decreases linearly with the number of samples regardless the dimensionality of the task space, hence discretizing the task space marginally affects optimality. It gives

$$\begin{aligned} W^* = \mathop{\arg\min}_{W\in\mathbb{R}^n} & \sum_{i=1}^{N} L_{\tau_i}(\hat{Y}(\tau_i, W)). \\ \text{st. } 1 \le i \le N,\ & \hat{Y}(\tau_i, W) \in \mathcal{C}_{\tau_i}, \\ & \|\nabla_\tau \hat{Y}(\tau_i, W)\|_\infty \le K \end{aligned} \tag{5.19}$$

Problem (5.19) is an extremely large, non-sparse, optimization problem. Solving it iteratively requires evaluating the gradient of all the trajectories $\hat{Y}(\tau_i, W)$ at once

for every update step of the parameters W. It is clearly intractable since the number of tasks N ranges from thousands to hundreds of thousands. An approximate solution can be computed accurately and efficiently using the ADMM. The idea is to solve the sub-problems associated with each task independently, as it was originally the case, while gradually enforcing to reach a consensus between them. Doing so is not straightforward under the previous formulation, so we propose to rewrite it as follows,

$$\begin{aligned} Y^* = & \underset{Y \in (\prod_{i=1}^N \mathcal{C}_{\tau_i}) \cap \mathcal{Z}}{\arg\min} \sum_{i=1}^N L_{\tau_i}(y_i), \\ \text{st. } \mathcal{Z} = \{ Z \in \textstyle\prod_{i=1}^N \mathbb{R}^p \mid & \min_{W \in \mathbb{R}^n} \sum_{i=1}^N R_{\tau_i}(z_i, W) = 0, \\ & \text{st. } 1 \leq i \leq N,\ \|\nabla_\tau \hat{Y}(\tau_i, W)\|_\infty \leq K \} \end{aligned} \tag{5.20}$$

where $Y = (y_1, y_2, \ldots, y_N)$ and $Z = (z_1, z_2, \ldots, z_N)$ gather trajectories for all tasks. It will give rise to the consensus optimization algorithm presented in section 5.4.

5.3.2 Trade-off Between Generalization Ability and Optimality

Not being able to solve the problem (5.14) exactly is not a big deal as long as the generated trajectories are certified, i.e. guaranteed to be feasible up to some acceptable margin to be defined. The optimality is not a real concern since finding a global minimum is out of reach in the first place for such large NLPs as trajectory planning problems. Reducing the computational cost is often more important than optimality. This is typically done by stopping the optimization at the early stage once the constraints are satisfied. Indeed, empirically the solution barely changes after this point due to the large number of constraints restricting the search space. The cost function L_τ is mostly here to give some insight to the solver in which direction trajectories with suitable properties lie. It helps to converge faster, more reliably, and avoid obviously inappropriate motions. One of the most common cost functions for bipedal walking is the total energy consumption of the system, another one is the L^2-norm of the jerk. These choices are motivated by studies on what humans are doing while walking. Yet, this argument is partially valid because of the mismatch between the motion of the robot and the patient in practice.

As mentioned before, limiting the expressiveness of the function approximation is essential for certifiability. One option is to bound its Lipschitz constant explicitly. Several technics for *Artificial Neural Network* (ANN) have been presented in chapter 4. They are cheap and easy to implement for small networks with up to a few millions of parameters, so this is not a blocking point. Yet another option is to restrict the number of training parameters W. Experimental results on Atalante suggest that it is sufficient in practice (cf. section 5.6.2 for details). As a by-product, it promotes uni-modality and thus predictability since it enforces the continuity of the mapping. All the generated trajectories are in the image of the functional $W \mapsto \hat{Y}(\cdot, W)$ by design no matter its expressiveness. However, no solution may exist if it is too

restrictive. It can be interpreted as a smoothness regularization of the original problem (5.14). In that respect, it is not surprising that it helps to limit the number of samples required to train an accurate model of the mapping as demonstrated in section 5.2.2. Low expressiveness corresponds to a strong regularization that would substantially alter the original solution and conversely.

It should be possible to update the desired task on-the-fly, for instance, the direction and velocity for bipedal robots. The target state and command would undergo an instantaneous jump when it happens, which adds up to the current tracking error. It is essential to minimize it because the capability of the LLC to absorb such discontinuity and cancel it while maintaining balance is limited. We have proven that the magnitude of the jump is upper bounded by the variation of task times the Lipschitz constant of the function approximation. Consequently, imposing the continuity of the mapping is essential for seamless transitions. That is all the more true for a legged exoskeleton because it enhances the comfort of the patient. For large variations of the task, it is likely necessary to split the update into several intermediary steps and wait for the tracking error to go back to normal before moving on. Hence, increasing the smoothness of the mapping would speed up transitions in such a case.

5.4 Iterative Solving

5.4.1 Alternating Direction Method of Multipliers

Let us consider the following separable non-convex consensus problem (Andreani et al., 2008; Magnusson et al., 2016)

$$\begin{aligned} (Y^*, Z^*) = & \underset{\substack{0 \leq i \leq N,\, y_i \in \mathcal{Y}_i \\ Z \in \mathcal{Z}}}{\arg\min} \sum_{i=1}^{N} f_i(y_i) + g(Z), \\ \text{st. } & 1 \leq i \leq N,\ \mathcal{Y}_i = \{y_i \in \mathcal{D}_y \mid \psi_i(y_i) = 0, \phi_i(y_i) \leq 0\}, \\ & \mathcal{Z} = \{Z \in \mathcal{D}_z \mid \theta(Z) = 0, \sigma(Z) \leq 0\} \\ & Y - Z = 0 \end{aligned} \tag{5.21}$$

where $\mathcal{D}_y \subset \mathbb{R}^p, \mathcal{D}_z \subset \mathbb{R}^{Np}$ are closed compact sets. The cost functions $f_i : \mathbb{R}^p \to \mathbb{R}, g : \mathbb{R}^{Np} \to \mathbb{R}$ and the equality and inequality constraints $\psi_i, \phi_i, \theta, \sigma$ are twice continuously differentiable. Such non-convex problems can be handled gracefully by the Augmented Lagrangian Method (Bertsekas, 1976, 1982). This is an exact penalty method (Di Pillo & Grippo, 1989; Han & Mangasarian, 1979), which means that the solution to the minimization of the augmented Lagrangian matches the one of the original problem. Its scaled form L_ρ for problem (5.21) can be stated as

$$L_\rho(Y, Z, \Lambda) = \sum_{i=1}^{N} f_i(y_i) + g(Z) + \frac{\rho}{2} \left\| Y - Z + \Lambda \right\|_2^2, \tag{5.22}$$

where Λ is the vector of Lagrangian multipliers and ρ is a penalty factor.

ADMM is a method to approximately solve optimization problems composed of a collection of sub-problems linked by a single linear equality constraint but otherwise independent (Boyd, 2010). This is done by minimizing the augmented Lagrangian in an alternating Gauss-Seidel manner, optimizing each variable while holding the others fixed (Bezdek & Hathaway, 2003; Boyd, 2010). Therefore, this method is especially suitable when each of the sub-problems has a readily available solving method. It yields to algorithm 3. Similarly to Y, $(z_1, z_2, \ldots, z_N)$ and $(\lambda_1, \lambda_2, \ldots, \lambda_N)$ denote the task decompositions of Z and Λ respectively. α is referred to as the dual step size.

Algorithm 3: ADMM for Non-Convex Consensus Problem

Initialization;
while *stopping criteria not met* **do**
  for $1 \leq i \leq N$ **do**
1 Update the optimization variables y_i^k individually and concurrently:

$$y_i^{k+1} = \underset{y_i \in \mathcal{Y}_i}{\arg\min}\, f_i(y_i) + \frac{\rho^k}{2}\|y_i - z_i^k + \lambda_i^k\|_2^2;$$

  end
2 Update the last optimization variable Z^k:

$$Z^{k+1} = \underset{Z \in \mathcal{Z}}{\arg\min}\, g(Z) + \frac{\rho^k}{2}\|Y^{k+1} - Z + \Lambda^k\|_2^2;$$

3 Update the Lagrangian multipliers:

$$\Lambda^{k+1} = \Lambda^k + \alpha(Y^{k+1} - Z^{k+1});$$

end

The stopping criteria may be a threshold on the L^∞-norm of the consensus error $\|Y^k - Z^k\|_\infty$ or the variation of the predictions over successive iterations $\|Y^{k+1} - Y^k\|_2$. The variation of the multipliers $\|\Lambda^{k+1} - \Lambda^k\|_2$ is often suggested but far from ideal. Indeed, the predictions are guaranteed to converge even if the problem is unfeasible, which is not true for the multipliers. See section 5.4.3 for details.

All the results presented here can be generalized with minor changes to any penalty function ϕ st. $\phi(x) > 0, \forall x \neq 0$ and $\phi(x) = 0$, at the extra condition to have $\phi''(0) > 0$. Basically, any L^p-norm to power p st. $1 < p \leq 2$ is valid. Using a norm of higher-order k may be desirable to be consistent with the certifiability criteria previously introduced that is based on L^∞-norm. The right way to do so is to combine it with a valid L^p-norm, typically the L^2-norm. It gives $\phi(x) = \|x\|_2^2 + \|x\|_k^k$, $k \leq 2$. It is usually counterproductive to use higher order that $k = 4$ because of ill-conditioning. Note that it is necessary to adapt the update rule for the multipliers accordingly, namely $\Lambda^{k+1} = \Lambda^k + \phi'(Y^{k+1} - Z^{k+1})$. See appendix B for details.

5.4.2 Proposed Consensus Optimization Algorithm

Step 1 of algorithm 3 consists in solving the sub-problems associated with the optimization variables y_i individually and concurrently. It corresponds to the original trajectory planning problem (5.1) modified to add a penalty term in the cost function,

$$y_i^{k+1} = \underset{y_i \in \mathcal{C}_{\tau_i}}{\arg\min}\, L(y_i) + \frac{\rho^k}{2}\|y_i - z_i^k + \lambda_i^k\|_2^2, \tag{5.23}$$

Next, the last sub-problem associated with Z is solved in step 2. It boils down to a standard regression problem,

$$\begin{aligned} W^{k+1} &= \underset{W \in \mathbb{R}^n}{\arg\min} \sum_{i=1}^{N} R_{\tau_i}(y_i^k + \lambda_i^k, W). \\ \text{st. } & 1 \leq i \leq N,\ \|\nabla_\tau \hat{Y}(\tau_i, W)\|_\infty \leq K \end{aligned} \tag{5.24}$$

Once done, Z^{k+1} is inferred from the function approximation $\hat{Y}$ as follows,

$$1 \leq i \leq N,\ z_i^{k+1} = \hat{Y}(\tau_i, W^{k+1}). \tag{5.25}$$

One can think of the multipliers λ_i as being the cumulative residual prediction error for task τ_i. They reveal where the function approximation makes repeating prediction errors. They progressively adapt the original trajectory planning and regression problems to give more weight to regions of the task space where errors are consistently made. Over iterations, the trajectories become easier to mimic for the function approximation but less optimal with respect to the original cost function, until a consensus is found. This algorithm reduces problem (5.20) to a sequence of trajectory planning and regression problems, each of which is well-studied and has an efficient solving method. It is summarized by algorithm 4.

It can be interpreted as a consensus optimization. Trajectory planning and learning are jointly solved, so that trajectory planning acts as a teacher rather than a demonstrator. In the view of trajectory planning, the penalty promotes trajectories that can be accurately reproduced by the function approximation despite its limited expressiveness. The smoothness constraint on the Lipschitz constant encourages the mapping from tasks to trajectories to be continuously differentiable with respect to the task. Regarding trajectory learning, the penalty alone is a weighted regression putting more effort into tasks that have been poorly predicted previously. The initialization of the algorithm 4 corresponds to the naive regression presented in section 5.2.1. The stopping criteria on the consensus error $\|Y^k - Z^k\|_\infty$ would be met right from the start unless co-adaptation is truly necessary. Thus, the computational burden is unchanged for problems that can be addressed via the naive regression.

5.4.3 Theoretical Analysis

Existence of a Solution

The existence of a solution depends on both the regularity of the trajectory planning problem (5.1) with respect to the task τ and the image of the functional

Algorithm 4: Guided Trajectory Learning

Generate N uniformly sampled tasks: $\{\tau_i\}_{i=1}^N$ st. $\tau \sim U(\mathcal{D}_\tau)$;
Initialize Y^0 by solving the original trajectory planning problem (5.1) concurrently for each task;
Initialize Λ^0 to zero;
Compute W^0 as a standard regression problem (5.5) and deduce Z^0 from equation (5.25);
while *not converged* **do**
1 Update Y^{k+1} by solving the adapted trajectory planning problem (5.23) concurrently for each task;
2 Compute W^{k+1} as the adapted regression problem (5.24) and deduce Z^{k+1} from equation (5.25);
3 Update the Lagrangian multipliers: $\Lambda^{k+1} = \Lambda^k + \alpha(Y^{k+1} - Z^{k+1})$;
end

$W \mapsto \hat{Y}(\cdot, W)$. Roughly speaking, all it requires for a well-posed trajectory planning problem is the set of training parameters W to be large enough if the function approximation is a *Feedforward Neural Network* (FNN). Theorem 4 proves that our proposed algorithm makes sense and has a strong theoretical grounding.

Theorem 4. *Let the task space $\mathcal{D}_\tau$ be a closed compact set and the function approximation be a FNN with training parameters $W \in \mathbb{R}^n$. Let us suppose that $\forall \tau \in \mathcal{D}_\tau$ the feasible set $\mathcal{C}_\tau$ of the trajectory planning problem (5.1) is non-empty and the Mangasarian Fromovitz Constraint Qualification (MFCQ) is satisfied for all $(\tau, y) \in \mathcal{D}_y \times \mathcal{C}_\tau$: the gradients of the equality constraints h_j are linearly independent at y and there exists a vector ξ st. $\nabla_y g_i(\tau, y)^T \xi < 0$ for all active inequality constraints and $\nabla_y h_j(\tau, y)^T \xi = 0$ for all equality constraints.*

Then, for any number of tasks N and threshold $\epsilon > 0$ on the maximum total prediction error, there exists a number of parameters n and a threshold K on the Lipchitz constant st. the following problem admits a solution:

$$
\begin{aligned}
Y^* = & \underset{Y \in (\prod_{i=1}^N \mathcal{C}_{\tau_i}) \cap \mathcal{Z}_\epsilon}{\arg\min} \sum_{i=1}^N L_{\tau_i}(y_i), \\
\text{st. } \mathcal{Z}_\epsilon = \{ Z \in \textstyle\prod_{i=1}^N \mathbb{R}^p \mid & \min_{W \in \mathbb{R}^n} \sum_{i=1}^N R_{\tau_i}(z_i, W) < \epsilon, \\
& \text{st. } 1 \leq i \leq N,\ \|\nabla_\tau \hat{Y}(\tau_i, W)\|_\infty \leq K \}
\end{aligned}
$$

Proof. Clearly, the trajectory planning problem problem (5.1) must admit a solution for any task $\tau \in \mathcal{D}_\tau$. It holds true for a given task τ if and only if the feasible $\mathcal{C}_\tau$ is

non-empty. Considering a collection of tasks $\{\tau_i\}_{i=1}^N$, it yields

$$\lim_{N\to\infty} \prod_{1\leq i\leq N} \mathcal{C}_{\tau_i} \neq \emptyset. \tag{5.26}$$

The next step is to prove the existence of a continuous function $\tau \mapsto Y(\tau)$ mapping tasks to feasible trajectories, i.e. $\forall \tau \in \mathcal{D}_\tau, Y(\tau) \in \mathcal{C}_\tau$. Twice continuous differentiability of the constraints is necessary but insufficient. In addition, the MFCQ must be satisfied for all $(\tau, y) \in \mathcal{D}_y \times \mathcal{C}_\tau$. Without it, isolated points may appear or disappear after perturbation of the task. This linear independence of the equality constraints can be relaxed in favor of a fixed rank in the neighborhood of x. Still, complementarity constraints are out-of-scope. The interested reader is encouraged to look at (Still, 2018, Chapter 6) for details.

The Universal Approximation Theorems introduced in section 3.1.2 state that some families of FNN can approximate arbitrary well continuous functions on a compact set $\mathcal{K}$ if the number of parameters n is large enough, typically a *Multi-Layer Perceptron* (MLP) with sigmoid activation function. Let us consider a threshold $\epsilon > 0$, a continuous function f defined on $\mathcal{K}$, and a family of FNN $\theta \mapsto \hat{f}_\theta$ with a set of parameters $\theta \in \mathbb{R}^n$ of variable length n. Then, there exists a number of parameters n_0 st. for all $n \geq n_0$,

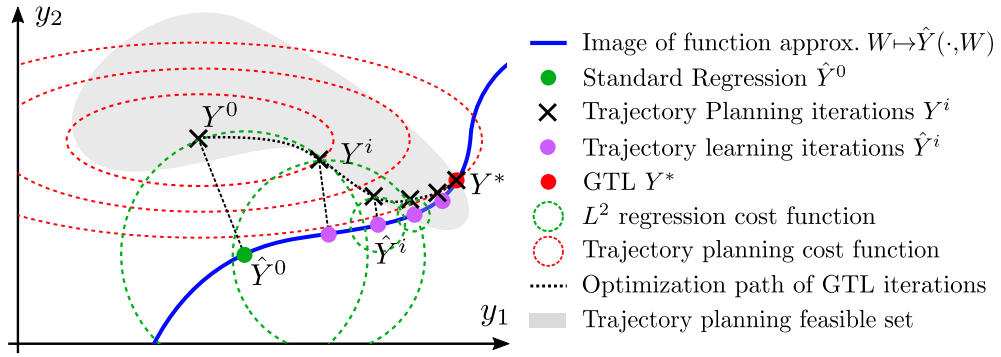
$$\exists\, \theta \in \mathbb{R}^n \mid \forall x \in \mathcal{K},\ \|\hat{f}_\theta(x) - f(x)\| < \epsilon. \tag{5.27}$$

Furthermore, Anil et al. (2019) has proven that FNN combining norm-constrained weight matrices with the gradient norm preserving GroupSort activation function are universal Lipschitz approximators on compact sets. Therefore, it is possible to take into account the Lipschitz constraint if the constant K is large enough. ■

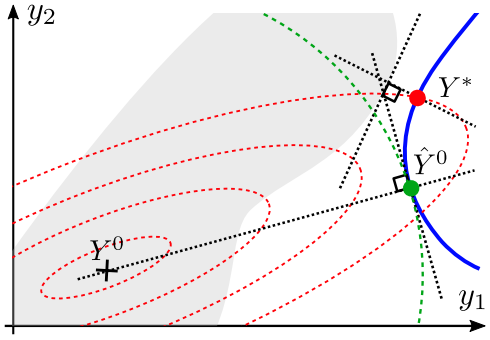
According to theorem 4, there exists a FNN with bounded expressiveness for which the unified optimization problem (5.15) is feasible up to some tolerance ϵ on the total reconstruction error. It cannot be canceled exactly in general. For instance, the feasible set may be degenerated and consist of a single trajectory. The universal approximation theorem still holds, but it proves density and not equality. Why it is not a major issue in practice will be discussed afterward.

Convergence Analysis

Originally, this method was intended to solve convex unconstrained optimization problems, but it has been proven to converge for non-convex consensus optimization problems under mild conditions (Hong & Luo, 2017; Hong et al., 2015; Magnusson et al., 2016). More specifically, Andreani et al. (2008) has proven that algorithm 3 converges R-linearly for α small enough and ρ constant. Strictly increasing ρ^k makes the convergence faster (up to super-linearly), but it is impracticable at some point. Indeed, a high penalty factor leads to ill-conditioning, making the optimization difficult to solve numerically. Successive iterations of GTL are illustrated in figure 5.2a.



(a) Feasible



(b) Infeasible

Figure 5.2: Comparison between standard regression and Guided Trajectory Learning. Y denotes all the trajectories $(y_1, y_2, \dots, y_N)$ associated with a whole collection of tasks. If the consensus optimization problem is feasible, GTL converges to a local minimum, i.e. the predicted trajectories $\hat{Y}$ are feasible and minimize the original total planning cost. If infeasible, the expected distance between the predicted trajectories and their respective feasible set is minimized, ignoring the original planning cost.

Theorem 5. *Algorithm 3 converges to the local minimum* (Y^*, Z^*) *of problem (5.21) that is the closest to its initialization* (Y^0, Z^0) *under the following assumptions:*

- *The consensus optimization problem (5.21) is feasible.*
- $\forall k \in \mathbb{N}$, y_i^k *(resp.* Z^k*) computed at step 2 (resp. step 3) of the algorithm is locally optimal.*
- *Let* $\mathcal{L}$ *denote the set of limit points of the sequence* $\left((Y^k, Z^k)\right)_{k\geq 1}$ *and let* $(Y^*, Z^*) \in \mathcal{L}$*.* (Y^*, Z^*) *is a regular point, i.e. the gradient vectors at* y_i^* *(resp.* Z^**) of the set of active constraints of* $\mathcal{Y}_i$ *(resp.* $\mathcal{Z}$*) are linearly independent.*
- *Let define* $\bar{\rho}$ *such that* $\forall y_i \in \mathcal{D}_y, \forall Z \in \mathcal{D}_z$*,* f_i *and* g_i *have an* L*-Lipschitz continuous gradient. The sequence* $(\rho^k)_{k\geq 1}$ *is increasing and either:*
  - $0 < \alpha \leq 1$ *and* $\exists k_0 \geq 1$ *st.* $\forall k \geq k_0$*,* $\rho^k > \bar{\rho}$*.*
  - $\alpha = 0$ *and* $(\rho^k)_{k\geq 1} \longrightarrow +\infty$*.*

The maximum number of training parameters n is bounded by hardware limitations. Indeed, evaluating the function approximation must be fast enough to enable online execution on the device. Hence, it is likely that the necessary conditions for the existence of a solution are locally not met due to some tasks in particular. In such a case, theorem 5 does not hold. Nevertheless, one can show that algorithm 3 converges to stationary point (Y^*, Z^*) that minimizes the consensus error $\|Y^* - Z^*\|_\infty$ under the assumption that all the trajectory sub-problems in step line 3 are feasible individually. This property makes it straightforward to check a posteriori whether the consensus optimization problem (5.21) is infeasible. It is essential because it is impossible to check beforehand that all the conditions to converge are satisfied.

More generally, the consensus error $\|Y^* - Z^*\|_\infty$ is a metric to compare the performance of different model architectures and assess which one has the lowest runtime cost. In theory, it indicates whether the number of training parameters n or the bound of Lipschitz constant K must be increased. Yet, the results in section 5.6.2 suggest that it is unreasonable to expect canceling out the consensus error perfectly in practice since it would require an extremely large number of samples. The total prediction error was supposed to be zero for certifiability, but it can be relaxed as long as the consensus error is lower than the acceptable safety threshold δ. Indeed, the local prediction error for each task $\|y_i^* - z_i^*\|_\infty$ adds up linearly to the worst-case prediction error between tasks. Therefore, one can modulate the density of samples over the whole task space based on the local prediction error and check after learning that the worst-case prediction is safe.

A numerical solver would fail to solve problem (5.21) directly and stop at an arbitrary infeasible point. Conversely, naive standard regression would not fail but its solution corresponds to the projection of the original optimal trajectories Y^0 onto the image of the functional $W \mapsto \hat{Y}(\cdot, W)$, which has nothing to do with mitigating infeasibility. Hence, the algorithm 3 is preferable over tackling the consensus optimization problem as a whole and the naive standard regression. Still, it may be necessary to stop the algorithm prematurely because the multipliers would grow unboundedly. The trajectory planning sub-problems would be increasingly ill-conditioned and the solver is likely to fail at some point. The infeasible scenario is depicted in figure 5.2b.

It follows from theorem 5 that algorithm 4 can be simplified by setting the dual step size α to 0. By doing this, it reduces the algorithm to an instance of the Alternating Direction Penalty Method (Magnusson et al., 2016). We refer to this variant as GTL-0. Since the multipliers are constant and equal to zero, it is no longer necessary to keep them in memory. Moreover, it enables generating a new set of tasks at every iteration, which is useful to adapt the distribution for certifiability as mentioned previously. Besides, it eases the implementation of the algorithm since the generation of the tasks is completely decentralized. The price to pay is a non-vanishing consensus error unless the sequence of penalty factors goes to infinity.

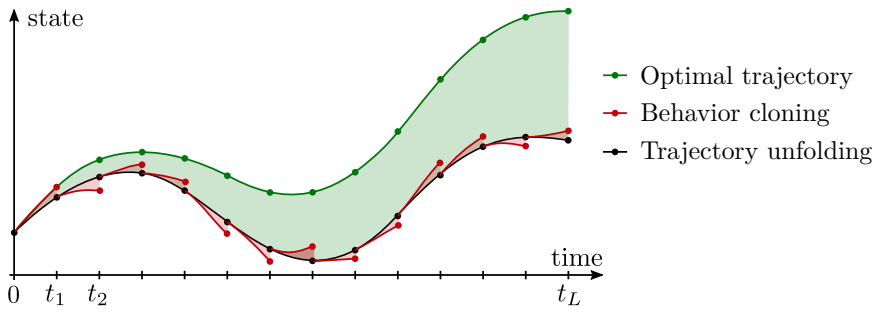


Figure 5.3: Comparison between Behavior Cloning and trajectory unfolding

5.5 Structured Prediction with Neural Network

Learning to predict trajectories is fundamentally a structured prediction problem. On the one hand, the next state x_{t+1} is related to the current state and command (x_t, u_t) via the dynamics of the system $x_{t+1} = f(x_t, u_t)$. On the other hand, all the optimal trajectories result from the same optimal policy π^* mapping the history of states $x_{:t} = (x'_t)_{0 \leq t' \leq t}$ to the next command u_{t+1}. Knowing the closed-loop dynamics f^*_π, the minimal information defining uniquely a trajectory comprises the initial state x_0 and the trajectory duration T. It is much more compact than the raw vector representation comprising the entire sequences of states and commands.

5.5.1 Autoregressive Model

The closed-loop dynamics under the policy f^*_π can be learned with an autoregressive model, which aims at predicting the future outcome of a sequence from the observation of the previous ones. Then, the whole trajectory is generated sequentially, one timestep after the other. Such a model can be viewed as an open-loop controller for which the state at the next timestep is supposed to be realized by the LLC.

Only the state is considered because the command is not involved in the closed-loop dynamics. An auxiliary model of the policy itself $u_t = \pi(x_t)$ may be trained to predict the nominal command if necessary. We put aside this question since they are usually unused in practice, and anyway it is a classical supervised learning problem without any particular challenge. In practice, even though only the states would be involved in recursive computations for trajectory unfolding, the same network would output both the next state and command. Sharing the hidden layers to have a common feature extraction halves the computation cost and improve prediction accuracy by fully leveraging correlations.

Traditionally, an autoregressive model is linear. Instead, we suggest using a generic MLP with the Sigmoid activation function (cf. section 3.1.2). Moreover, we assume that the optimal policy only depends on the current state, so that the dynamics under the optimal policy is Markovian $x_{t+1} = f^*_\pi(x_t, \tau)$. It implies that trajectories cannot cross each other since the command for a given state-task pair are unique. This assumption is reasonable for high-dimension systems.

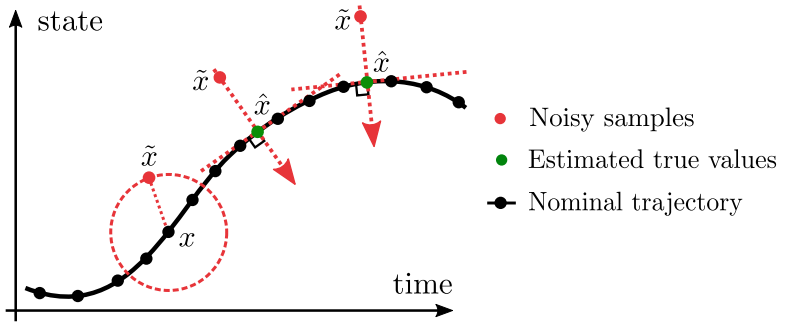
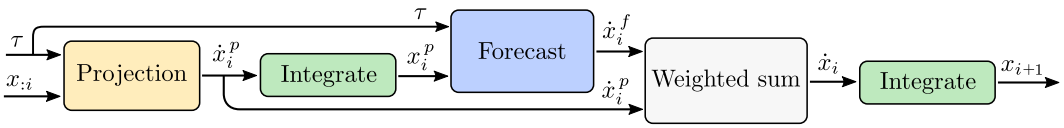


Figure 5.4: Denoising autoencoder. Noisy samples are generated by corrupting the original trajectory samples with additive isotropic Gaussian noise. Then, a classical FNN is trained to recover the original noise-free samples from the noisy ones.

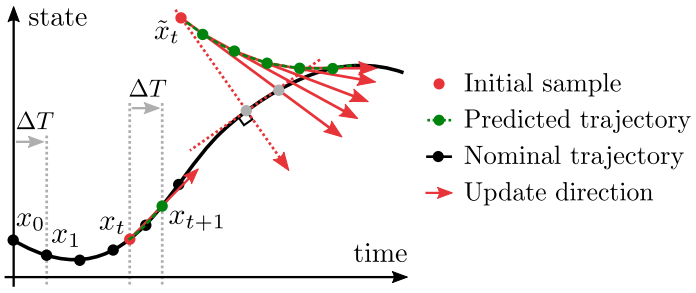
A basic approach to learning f_π^* is to unfold complete trajectories at training time. The gradient flows through the network a large number of times during back-propagation, leading to gradient vanishing problems. It is a well-known issue when training *Recurrent Neural Network* (RNN). It is mitigated by using a residual networks $\tilde{x}_{t+1} = x_0 + f(x_0) + \sum_{i=1}^{t} \hat{f}(\tilde{x}_i, \tau)$. If memory is needed because the Markovian property is not satisfied, then *Long Short-Term Memory* (LSTM) must be used. Yet, it does not help to reduce the computational complexity of learning complete trajectories that is quadratic with respect to their length. Conversely, BC does not face such issue: the current state is taken from the optimal trajectory at every timestep $\tilde{x}_{t+1} = x_t + \hat{f}(x_t, \tau)$. It ignores temporality, and thereby timesteps that can be sampled in any order during training (see figure 5.3).

After training, it remains necessary to unfold trajectories recursively to generate them at runtime. In the case of BC, the prediction would diverge exponentially fast if nothing is done to prevent the compounding of errors. We propose to use a denoising autoencoder that makes sure the trajectory manifold is attractive. This procedure is depicted in figure 5.4. One can show it is equivalent to learning the projection operator onto the trajectory manifold. Galashov et al. (2022) recently proposed a similar technique. They assume some expert policy is available and can be queried at will. It follows that they can update the optimal action after perturbing the original states along the trajectories. Having such an expert policy at one's disposal is very restrictive and not necessary to learn the optimal closed-loop dynamics f_π^*. On the contrary, we just assume that the next optimal state is not affected by the local perturbation that was applied to the current state.

Apart from predicting normal trajectories, learning f_π^* is a promising approach to extrapolate smooth transition between tasks at any point in time. The distance from the trajectory manifold may grow very large when the current task is updated. How fast the current state converges back to the trajectory manifold cannot be adjusted at will if the denoising autoencoder is trained to both project and move forward in time at once. Moreover, it may be useless to assess how far the current state is for various applications, including assistive control. We learn to project and forecast separately



(a) Block diagram



(b) Transition example

Figure 5.5: Trajectory generation with an open-loop policy. A denoising autoencoder computes the gradient to project on the trajectory manifold. Then, a model of the vector field computes the gradient to move forward in time. Finally, the projection and forward gradient are summed up and integrated until the next timestep.

using vanilla behavior clone and denoising autoencoder respectively. Then, both are combined at runtime using a weighted sum. This approach is illustrated in figure 5.5.

5.5.2 Generative Single Forward Pass Model

Here we consider the case where the whole sequences of states and commands are predicted at once. Computation of the whole sequence is very cheap compared to autoregressive models since all timesteps are computed concurrently instead of sequentially, and thereby it can exploit way better the available resources. Despite the computational complexity of learning whole trajectories being quadratic of their length, it remains advantageous to predict them in a single forward pass for short sequences up to several hundred timesteps.

Aside neural networks, many models are available: *Support Vector Regression* (SVR), *Gaussian Mixture Model* (GMM), *Variational Fourier Features for Gaussian Process* (VFF) and *Stochastic Variational Inference for Gaussian Process* (SVIGP) to name a few. Yet, they tend to scale poorly with the number of samples or the dimensionality of the problem, and how to enforce the Lipschitz constant K is unclear. On the contrary, a classical MLP features excellent scalability and versatility. Although effective in practice, it is often necessary to rely on regularization technics to reduce the search space artificially and avoid overfitting. For instance, it helps to enforce temporal smoothness. The maximum number of parameters is upper-bounded by the capability of the embedded hardware and so is the prediction accuracy. In particular, for a given width and depth, the prediction accuracy drops linearly with the length of the trajectories.

It would be beneficial to leverage the actual structure of the problem to further improve accuracy without increasing the computational cost. To this end, we propose to use a *Deconvolutional Neural Network* (DNN) (Dong et al., 2015; Wojna, 2019) as function approximation since it is especially well-suited for generating multidimensional temporal sequences (Tachibana et al., 2018). Notably, the deconvolution operation combines 1D convolution and upsampling, as opposed to the usual transpose convolution that is sensitive to artifacts (Odena et al., 2016). The entire architecture is described in figure 5.6. It completely decouples the task from the time in a way that closely resembles the minimal information representation described above, but it is more flexible. First, a low-dimensional feature encoding substitutes the initial state and duration via a FNN, which has the potential to be more compact. Next, the whole sequences of states and commands are unfolded at once by a DNN without learning the closed-loop system dynamics explicitly. The Markovian assumption is not needed and temporal causality can be ignored by leveraging both forward and backward temporal correlations. This two-stage approach is similar to predicting coefficients for the monomial basis or spline but expressiveness and smoothness are easier to adjust. While the regularity with respect to the task depends on both networks, the regularity with respect to the time only depends on the DNN.

The actual hyperparameters of the model are the width L_h and depth N_h of the FNN. The size of the 1D convolution kernel N_k is usually small, typically 3. It reduces the number of parameters without undermining expressiveness since it would have both short- and long-range effects through the successive upsampling layers regardless. It would be symmetric or not depending on whether enforcing causality is relevant. On its side, the number of upsampling layers determines the temporal smoothness of the predicted trajectories. More specifically, each upsampling layer halves the upper bound on the Lipschitz constant with respect to the time. Such regularization is beneficial since it reduces the search space and mitigates overfitting. However, it must not exceed the inherent regularity of the optimal trajectories to avoid impeding accuracy. The optimal trade-off is problem-specific. Finally, the number of channels N_{ch} and the length of the feature vector L_{seq} derive from other parameters, $N_{ch} = p\, 2^{N_{up}-2}$, $L_{seq} = \text{ceil}(L_T/2^{N_{up}})$.

As mentioned before, the consensus optimization problem is likely to be partially infeasible in practice since trajectory planning and learning may not reach a consensus for some tasks. The associated feasibility constraints would be violated. It is manageable regarding certifiability, but it is nonetheless necessary to enforce the physical limitations $\mathcal{D}_x, \mathcal{D}_u, \mathcal{D}_T$. In particular, the predicted joint position, velocities, and torques must be within hard bounds all along, or it would be impossible to execute the trajectory on the real device otherwise. Taking inspiration from *Interior Point Optimization* (IPO) for policy optimization Liu et al. (2020b), we suggest instead to use exponential barrier penalty function. It would reduce the prediction error compared to clipping at runtime, which is beneficial for certifiability.

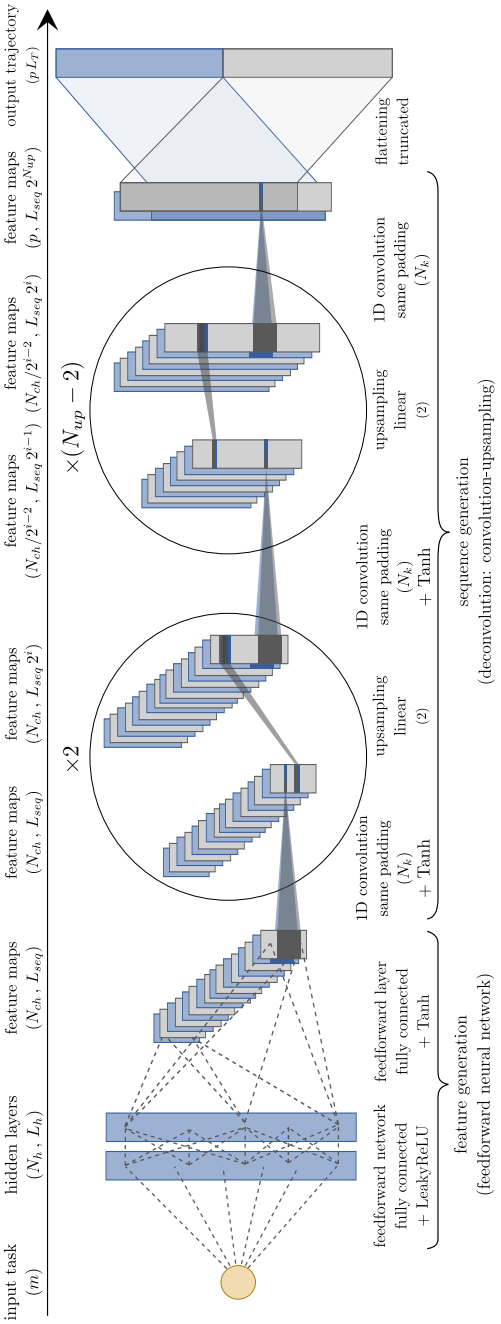


Figure 5.6: Specialized network architecture for continuous-time Markov process: a FNN generates low dimensional features and a DNN produces sequences from them. i denotes the index of the upsampling layer.

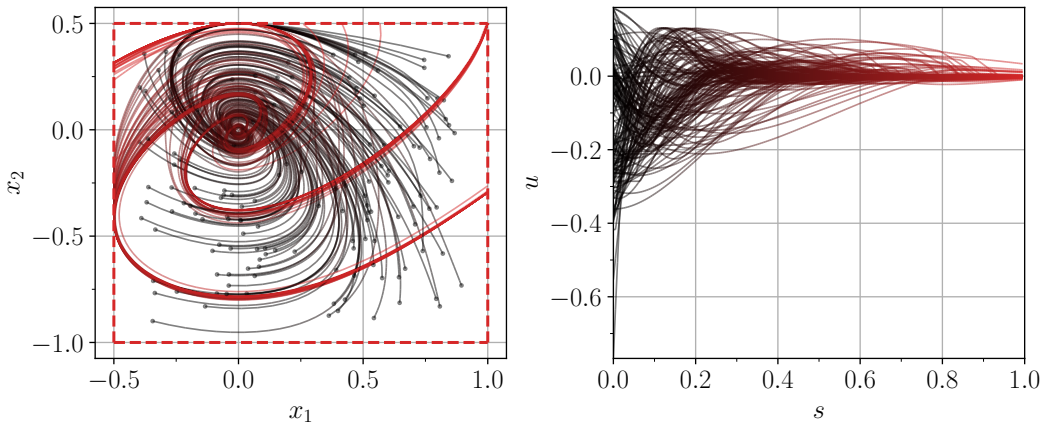


Figure 5.7: Overview of the original optimal trajectories. The red dashed lines are the boundary of the admissible set $\mathcal{D}_x$. The color gradient from black to red represents the flow of time. s denotes the normalized time.

5.6 Experimental Validation

5.6.1 Toy Problem: Van der Pol Oscillator

Let us consider the problem of keeping a Van der Pol oscillator within bounds over a finite horizon while minimizing the commands:

$$\forall \tau \in \mathcal{D}_\tau,\ Y_\tau^* = \underset{Y \in \mathcal{C}_y}{\arg\min} \int_0^T u^2 \,\mathrm{d}t$$

$$\begin{aligned}
\text{st. } \mathcal{D}_\tau &= \{-0.4 \leq x_1(0) \leq 0.9,\ -0.9 \leq x_2(0) \leq 0.4,\ 0.2 \leq u_{\max} \leq 1.0,\ 0.2 \leq T \leq 1.0\} \\
\mathcal{D}_x &= \{x \in \mathbb{R}^2 \mid -0.5 \leq x_1 \leq 1.0,\ -1.0 \leq x_2 \leq 0.5\} \\
\mathcal{D}_u &= \{x \in \mathbb{R}^2 \mid -u_{\max} \leq u \leq u_{\max}\} \\
\mathcal{C}_y &= \{(x, u) \in \mathcal{D}_x \times \mathcal{D}_u \mid \dot{x}_1 = (1 - x_2^2)x_1 - x_2 + 4u,\ \dot{x}_2 = x_1\}
\end{aligned} \tag{5.28}$$

A subset of the original optimal solution over the whole task space is shown in figure 5.7. It gives an overview of the diversity of trajectories to learn. The regularity of the original mapping $\tau \mapsto Y^*(\tau)$ can be analyzed by plotting the pairwise distance in task space vs. trajectory spaces. The line of minimal slope keeping all samples on its right is a tight lower-bound estimate of the global Lipschitz constant K. It appears from figure 5.10a that K is larger than 50, which prevents certifiability.

A basic MLP with a single hidden layer of width 50 and Leaky-ReLU activation function is used as function approximation. Assuming that the norm of the gradient of the activation functions is smaller than 1, one can show that the Lipschitz constant in L^∞-norm of the network K is upper-bounded by,

$$K \leq \prod_{1 \leq k \leq n} \|W_k\| \tag{5.29}$$

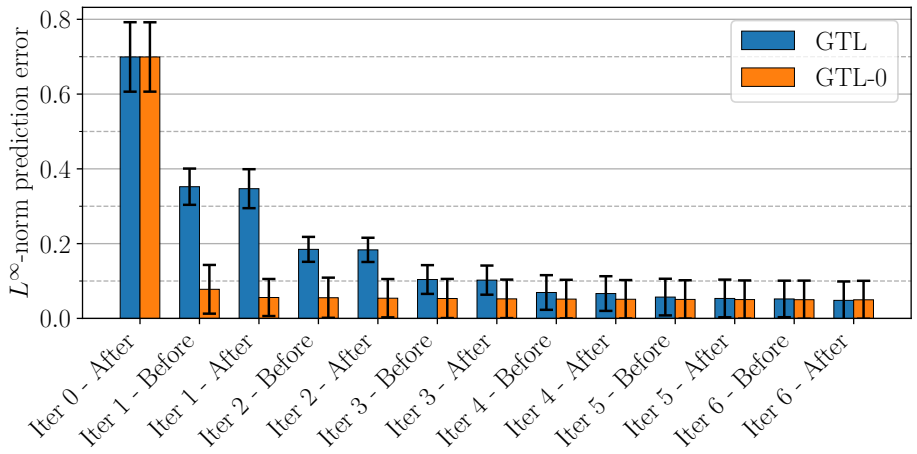


Figure 5.8: Evolution of the total prediction error over GTL iterations

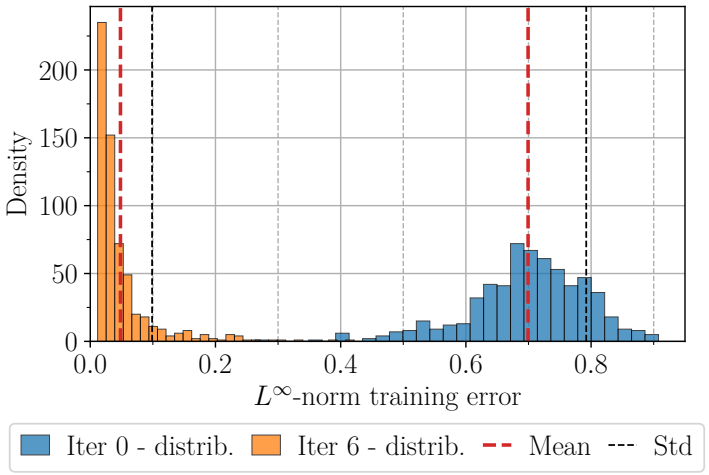


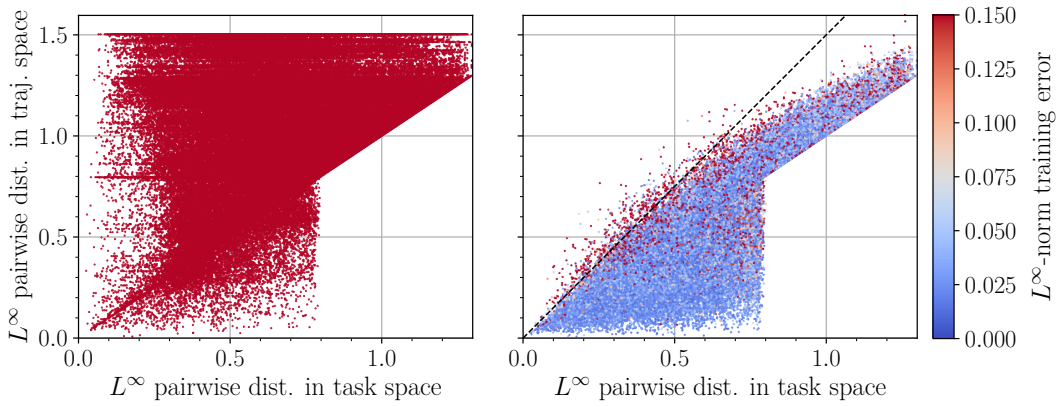
Figure 5.9: Prediction error distribution for initial and final GTL iterations

where n is the depth of the network and W_k denotes the weight matrix of layer k. In practice, this upper bound is very tight for shallow networks, specifically for the L^∞-norm. The spectral normalization of all the weight matrices keeps the Lipschitz constant bounded for the L^2-norm. Applying the same principle to the L^∞-norm is even easier since each row is normalized individually,

$$1 \leq k \leq n,\ 0 \leq i \leq w_{k+1},\ \sum_{0 \leq j \leq w_k} |W_{k,(i,j)}| = K^{\frac{1}{n}}, \tag{5.30}$$

where w_k denotes the width of layer k and $W_{k,(i,j)}$ is the element of the weight matrix W^k on the i-th row and j-th column. In practice, K is set to 1.7 to demonstrate the effectiveness of the method.

The multiplier update step α and the penalty factor ρ are equal to 0.5 and 50



(a) Regression (b) GTL

Figure 5.10: Pairwise distance between training samples. The black dashed line is the Lipchitz constant K of the function approximation.

respectively for all iterations. We refer to the initialization of algorithm 4 either as 'iter 0' or as 'Regression' to highlight that it corresponds to the standard regression method. Similarly, GTL sometimes alludes to its last iteration assuming convergence. Figure 5.8 shows that the total prediction error is initially very large for 'Regression'. Nevertheless, the error reduces significantly over iterations, and the algorithm converges extremely fast: 5 iterations for the exact formulation GTL, and only 2 for the simplified one GTL-0. More precisely, the total prediction error drops quadratically over iterations for GTL while it is instantaneous for GTL-0. This effect is explained by the multipliers smoothing out the progress and avoids premature convergence. It gives a chance to reach a better compromise by letting the function approximation adapts itself, even though most of the improvement is due to the modification of the trajectory planning problem in this example. Either way, the error never vanishes completely because the consensus optimization problem is partially infeasible, essentially due to the very restrictive threshold on the Lipschitz constant. The distribution of errors for the first and last iterations are compared on figure 5.9. While it is very spread at the beginning because of the irregularity of the mapping, it is much more localized at the end, except for a few outliers that are not reproducible.

At the end, most of the optimal trajectories generated by the modified planning problem are compliant with the function approximation. While it is difficult to analyze the effect of the limited expressiveness of the network, it is clear from figure 5.10b that the Lipschitz constant of the mapping has been dramatically reduced. The effect is even more noticeable locally for close-by samples. The certifiable safety threshold can be deduced from the Lipschitz constant and the number of training samples. However, it is not applicable to the whole task space since some areas are poorly reproduced. In theory, those areas of the task space should be excluded. Alternatively, the constraint on the Lipschitz constant can be relaxed to get around this issue, but it would require increasing the number of samples.

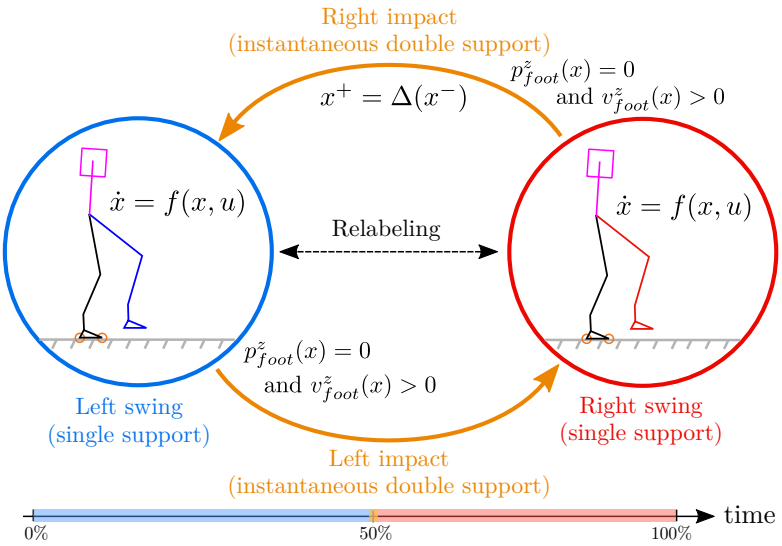


Figure 5.11: Flat-foot walking domains

5.6.2 Application to Atalante: Flat Foot Walking

Optimal Control Problem Formulation

We assume that the patient is rigidly fastened to the exoskeleton, including the upper body. As discussed in section 2.2.1, this assumption is questionable. Nevertheless, it enables considering the system patient-exoskeleton as a usual bipedal robot so that most of the theoretical background can be translated nearly effortlessly. The ensuing discrepancy is mitigated by adding safety margins in the stability conditions when formulating the trajectory optimization problem.

The mechanical structure of the robot has deformation points located at the ankles and hips, and the swing leg is touching the ground earlier than expected because of it. The early impact could be prevented entirely by adding the expected vertical deflection of the swing leg to the task space and generating motions virtually climbing stairs of this height. Then, the deflection parameter would be adjusted manually each time that the patient or gait features change. Although effective, this procedure is tedious. Thus, it was decided to actively compensate for the deformation via robust feedback control methods such as admittance control presented in section 2.2.3. This approach is more versatile and improves overall stability, regardless of whether modelling uncertainties or unexpected events come into play.

This work focuses on flat foot walking in a straight line. Stability is enforced conservatively by making sure that the stance foot remains flat on the ground and does not slip. Formally, it corresponds to keeping the center of pressure far from the edges of the foot. It is supposed for simplicity that the stance foot never rotates around the vertical axis. This hypothesis is sensible since the torsional force should be negligible. Besides, the swing foot is required to stay parallel to the ground all the way. This limits the risk of early impact and scuffing the feet.

Unilateral constraints are used for the ground contact model. This follows that the impact is rigid, and hence the double support phase is instantaneous: the robot is always in single support during the continuous dynamics, with the discrete dynamics modelling the impact (see figure 5.11). More details about the impact model are found in appendices A.1.3 and C.2.2. We take advantage of the symmetry of the gait and the robot across the sagittal plane to optimize the trajectory only for right leg support. The other side is obtained through the relabeling trick described in appendix A.1.3. The system is fully-actuated during the whole motion, thereby the state only has to comprise the position and velocity of the motors.

As mentioned in section 5.3.2, the choice of the running cost is of minimal importance because the constraints are such that any gait satisfying them would be satisfactory. It is unreasonable to expect to find a running cost that leads to consistently superior walking patterns experimentally either in terms of comfort or physiology over the whole task space. The instantaneous power consumption was used in practice. Other common choices such as the L^2-norm of the jerk would be meaningful physically but lead to higher optimization failures on average. Additional regularization terms are added to further alleviate multi-modality issues. This is not necessary since the algorithm 4 can handle it, but it helps to reach a consensus faster without increasing the computational burden. Here, the maximum knee extension of the stance leg over the motion is maximized, which is consistent with the human walk.

The resulting trajectory optimization problem can be solved efficiently under the direct collocation framework, even for systems with a high number of DoFs such as humanoid robots. This approach is presented in appendix A.

Patient Morphology and Gait features Task Space

The task gathers a set of high-level features of the gait that the physiotherapist or the patient might be interested in playing with, namely:

- duration, length, and width of the steps,
- lower-bound height of the swing foot before moving forward,
- upper-bound displacement of the pelvis in the frontal plane,
- upper-bound absolute roll and positive pitch angles of the pelvis,
- upper-bound absolute momentum along z-axis of the stance foot,
- apex of the swing foot trajectory,
- upper-bound velocity of the foot at impact,
- upper-bound excursion of the swing foot behind its take-off position,
- upper-bound excursion of the swing foot above its touch-down position,
- initial-final position and dimension of the bounding box of the CoP,
- upper-bound torque consumption.

We also include the characteristics of the patient in the task space, that is to say:

- morphology of the patient: height and weight,
- settings of the exoskeleton: thigh and shank lengths.

It increases the number of dimensions of the task space but enables computing a

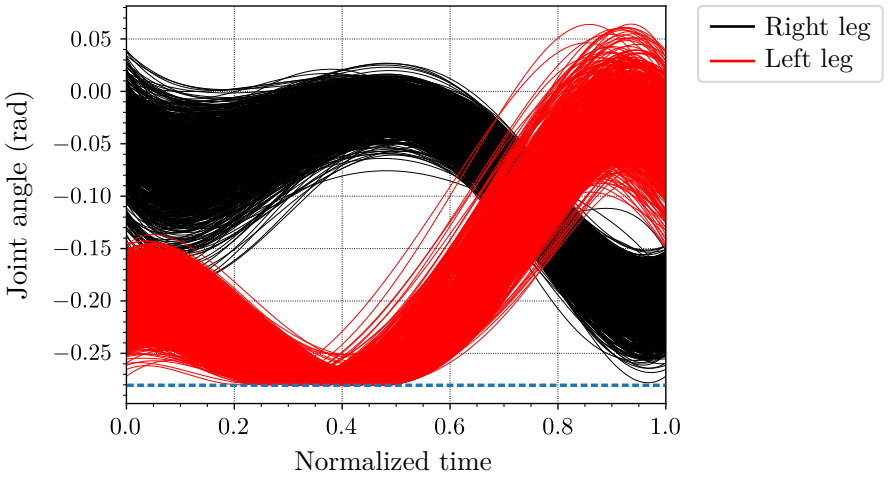


Figure 5.12: Original optimal trajectories of the ankle joints. The stance leg is the left. Accurate ankle trajectory planning is critical since it has a huge impact on the position of the *Center of Pressure* (CoP) of the robot, and the height and orientation of the swing foot.

unique function approximation for all patients and gaits at once. Physically, it makes no sense to change the patient on-the-fly, but it is computationally more efficient since it leverages the correlation between similar patients by nature. Mathematically, it is free of cost because patient features are just additional task variables.

Data Overview

A small subset of the of optimal trajectories associated with the original planning problem is shown in figure 5.12. It focuses on the position of the ankle over time for clarity as it looks very similar to the other joints. The temporal smoothness supports the use of the aforementioned deconvolutional network. In addition, the optimal position at a given timestep does not change much. It never exceeds $0.2 rad.s$ in the worst-case, i.e. the right ankle joint at about 10% of the step. As a result, it should be doable to reproduce them using a function approximation with limited expressiveness and to strongly constrain the Lipschitz constant with respect to the task. This conclusion is supported by figure 5.13 that shows the correlation between the distance in task space and trajectory space. Nevertheless, trajectories that are close in task space can be very far away in trajectory space in a few cases. This dispersion is the outcome of the multi-modality issue. By looking at this figure, it appears that 10.0 seems to be appropriate for the Lipschitz constant in L^2-norm.

Validation criteria

Classical *Proportional-Integral-Derivative controllers* (PIDs) are used on the robot to track the nominal trajectories. They are tuned in a way to guarantee that the

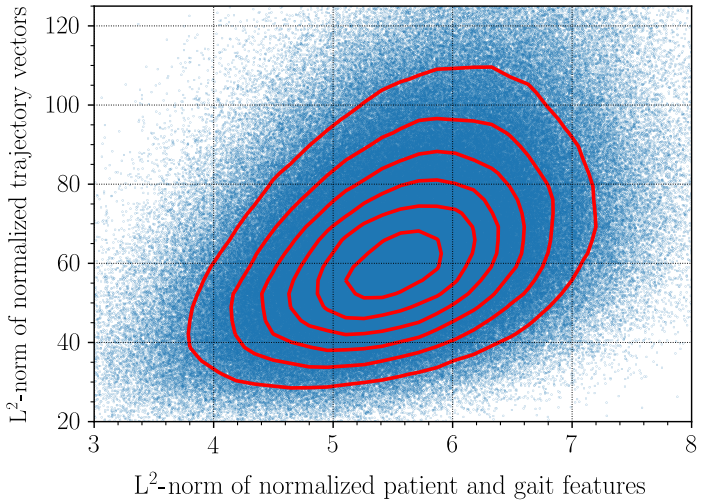


Figure 5.13: Pairwise distance between training samples

maximum tracking error for the joint positions does not exceed 0.01rad in the nominal case. It has proven to be small enough to achieve stable walking and safe for the patient when the trajectories are generated offline through optimization. It enforces an upper bound on the L^∞-norm prediction error, so it concurs with the acceptable safety threshold for certifiability of the function approximation. The effect of the number of training parameters on the prediction accuracy for 'Regression' is evaluated in figure 5.14. It demonstrates it improves very slowly, and it cannot be expected to satisfy the desired validity criteria just by increasing the number of parameters. This result alone confirms the relevance of our algorithm.

The task space is very high dimensional to challenge the scalability of our method. It is critical to make sure the predicted trajectories are safe, but it is not possible to certify the function approximation over the whole task space. Indeed, the condition derived in section 5.2.2 would require a number of samples that is beyond the capability of the available resources despite the apparent smoothness of the trajectories. To circumvent this limitation, we check first that all the prediction errors for the training samples are below the acceptable safety threshold. Then, the task space is discretized and the violation of the feasibility constraints is verified offline for all trajectories. It avoids having to design a fallback strategy since the predictions are guaranteed to be valid at runtime.

Training Performance

Solving the trajectory planning problem is computationally heavy. Hence, it is critical to choose appropriately the number of trajectories to be generated. The number of samples is increased progressively until the prediction accuracy on a testing set does not improve any further. Figure 5.15 indicates that about 40000 samples are sufficient regardless of the number of GTL iterations. It reduces over iterations because the

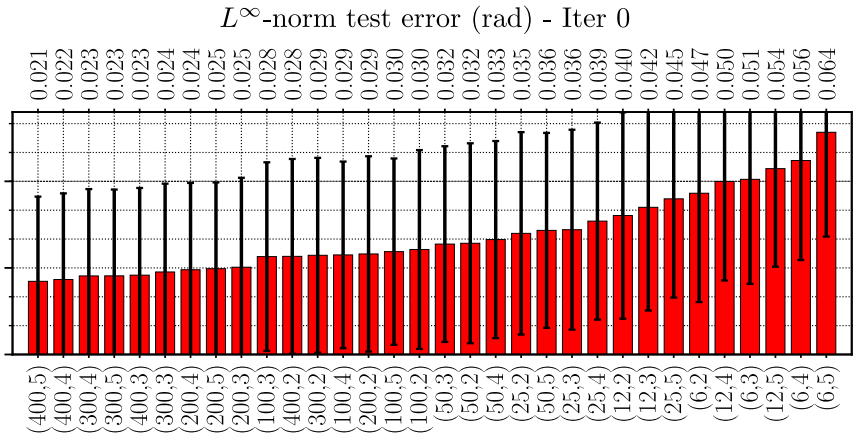


Figure 5.14: Effect of the number of parameters on the training error

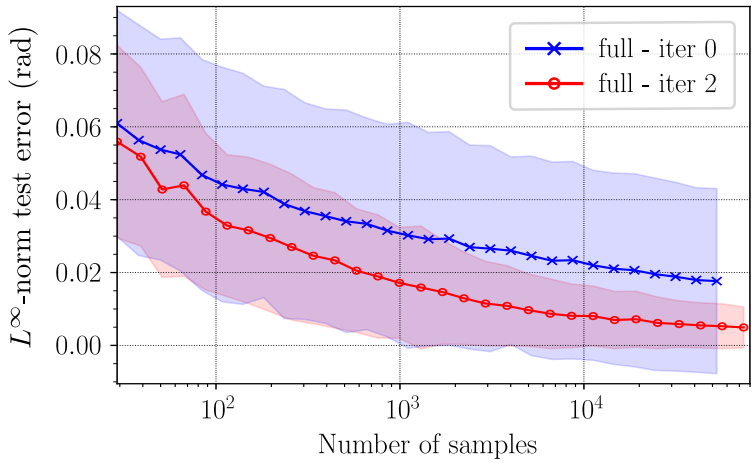


Figure 5.15: Effect of number of samples on the testing error

mapping gets more regular, which is confirmed by the standard deviation reducing. It takes almost 3 minutes to obtain one trajectory on a single core. Grid5000, a French national large-scale grid for computer science, was used to generate the training data on 300 physical cores during about 12 hours per GTL iterations. The convergence rate of the solver fluctuates between 92% and 97%, ending up with a collection of about 70000 trajectories. It is surprisingly small given the dimensionality of the task space as it corresponds to one training sample of all the vertices of the hypercube domain. It suggests that the final mapping is very smooth.

The performance of GTL-0 is solely assessed for practical reasons. First, it converges much faster than GTL. Secondly, it is easier to implement as it is not necessary to keep track of the training tasks and multipliers over iterations. It is not a major limitation because it gives a lower bound on the expected performance of GTL, and it was demonstrated in the Van der Pol example that it does not impede the final per-

m	p	q	L_T
16	24	12	200

(a) learning problem

N_h	L_h	N_{up}
1	200	5

(b) neural network

N	γ	ρ^k	α
70000	1.0	5.0	0.0

(c) GTL

Table 5.1: Parameters summary

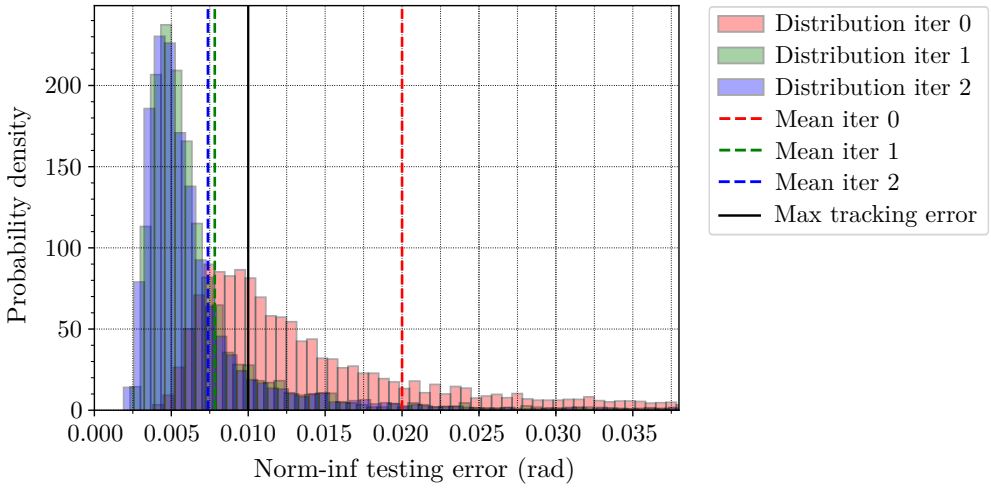


Figure 5.16: Norm-inf test error distribution over iterations of GTL-0

formance if the penalty factor is large. Finally, no Lipschitz constraint is considered because its relation with certifiability was established only later. The penalty factor is kept equal to the highest value for which ill-conditioning does not affect significantly the convergence rate, i.e. $k \geq 1$, $\rho^k = \bar{\rho} = 10$. Indeed, the total computation time already doubles if the convergence rate drops to 90%. Empirically, it approximates a solution to the problem (5.21) sufficiently well in relation with the acceptable safety threshold. The values of the parameters are summarized in table 5.1.

Prediction Accuracy

The accuracy of 'Regression' is compared to GTL-0 in table 5.2. The conclusions are consistent with the Van der Pol example. Unlike 'Regression', GTL-0 shows promising results despite the lack of multipliers. figure 5.16 shows that the error distribution for 'Regression' is very spread and has a long right tail. Therefore, a large part of its predictions has a reconstruction error much larger than the maximum acceptable error of 0.01rad. It is much better for GTL-0, but the prediction error of GTL-0 is not zero after convergence. It is hard to tell whether it could be handled by the multipliers, or the consensus optimization problem is partially infeasible. Increasing the penalty factor ρ may further improve the accuracy, at a cost of worsening the conditioning of the trajectory planning problem.

The efficiency of GTL-0 can be understood in the light of figure 5.17. It reveals

Algorithm	Mean (rad)	Mode (rad)	> 0.01 rad	> 0.015 rad
Regression	2.01×10^{-2}	8.16×10^{-3}	50.3%	16.1%
GTL-0	7.43×10^{-3}	4.25×10^{-3}	10.5%	4.5%

Table 5.2: Testing accuracy in L^∞-norm

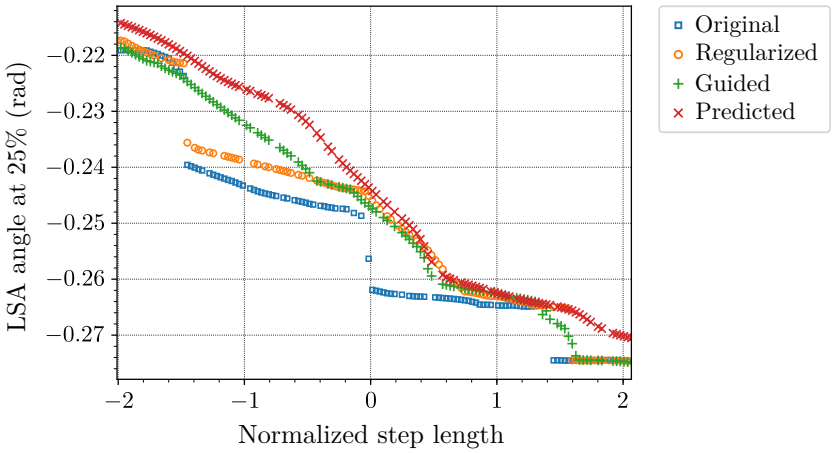


Figure 5.17: Continuity of the trajectories with respect to the task. It shows the effect of the variation of the step length of the walking gait on the angle of the left ankle joint at 20% of the step (see figure 5.12).

several discontinuities for the solutions to the original problem, which are impossible to fit accurately using a continuous function approximation. By contrast, the trajectories generated via GTL-0 are perfectly continuous with respect to the task. A single iteration of GTL is sufficient to enforce the continuity of the solutions, thereby explaining the very fast convergence of the algorithm in 2 iterations.

Real World Validation

We have evaluated our ability to control the average velocity of the exoskeleton. Data are only available for GTL-0, since most predictions were unstable on the real robot using the standard regression. It is not surprising as our validation criterion is not met for half of the trajectories. Although it does not mean that the predicted trajectories are necessarily unstable, such a large ratio is a strong indicator. The situation is much more favorable for GTL-0, and it turns out that many of them were stable in reality. Notably, the prediction accuracy is getting worse as the task is closer to the boundary of the task space. This boundary effect is directly related to the increase of the average distance to the nearest neighbor $D_{nn}(\tau)$. It could be compensating at weighting the training samples accordingly during training of the function approximation. Another option is to simply restrict the span of the task space at runtime. In practice, all the predicted trajectories were stable for GTL-0

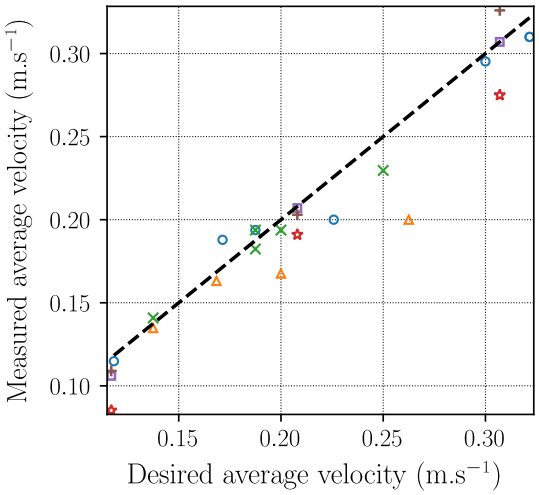


Figure 5.18: From simulation to reality: comparison between the desired and achieved average velocity on 6 valid people with a different morphology. Each pair marker-color corresponds to one patient. The average velocity is a by-product of the step length and duration which are actual decision variables.

after shrinking the range of the decision variables by about 10%. Figure 5.18 shows that the measured velocities are close to the desired ones for every patient.

5.7 Concluding Remarks

In this chapter, we presented a novel algorithm called GTL that learns a function approximation of the solutions to a trajectory planning problem over a task space. Accurate and reliable predictions are ensured by simultaneously training the function approximation and adapting the trajectory optimization problem such that its solutions can be perfectly fitted by the function approximation and satisfy the constraints concurrently. It results in a consensus optimization problem that we solve iteratively via ADMM. We demonstrate its efficiency on flat-foot walking with the exoskeleton Atalante. We believe that our method offers a new scope of applications, such as reinforcement learning, perturbation recovery, or path replanning. Enabling adaptation of the architecture of the neural network itself to further improve its efficiency and usability is an exciting direction for future work.

The whole nominal trajectory is predicted at once, so it does not help to extrapolate stable transitions between them. It means that the task must be updated slowly enough to rely on local attractiveness due to closed-loop control. Besides, motions that require a very faithful model are out of reach using traditional model-based approaches. For example, sliding on purpose is natural for humans for recovering balance or doing sharp turns. These questions are addressed in the next chapter. It presents a more disruptive approach that combines *Imitation Learning* (IL) with RL to learn a robust and versatile control policy.

Chapter 6

Learning Robust and Safe Policy

Contents

6.1 Problem Setup . . . 161
6.1.1 Learning Environment . . . 161
6.1.2 Training Scenarios . . . 165
6.2 Constrained Policy Optimization . . . 166
6.2.1 Implicit Constraints through Early Termination . . . 167
6.2.2 Explicit Constraints through Barrier Functions . . . 170
6.3 Trajectory-Based Imitation using Reinforcement Learning . . . 172
6.3.1 Generalized Space-Time Bounds . . . 173
6.3.2 Scalable Multi-Tasking through Lower-Bound Maximization . . . 182
6.3.3 Task Transitioning via Cross-Initialization . . . 186
6.4 Improving Convergence, Predictability and Safety . . . 187
6.4.1 Reward Engineering . . . 187
6.4.2 Termination Conditions . . . 191
6.4.3 Explicit constraints . . . 192
6.4.4 Smoothness Conditioning . . . 193
6.5 Ensuring Robustness for Bridging the Simulation-Reality Gap . . . 194
6.5.1 Plausible External Disturbances . . . 194
6.5.2 Feasible Initial State Generation . . . 195
6.5.3 Domain Randomization . . . 196
6.6 Results . . . 198
6.6.1 Policy Network Architecture . . . 198
6.6.2 Training Performance . . . 198
6.6.3 Validation in Simulation . . . 198
6.6.4 Standing Push Recovery on Atalante . . . 201
6.7 Concluding Remarks . . . 201

Offline generation of natural motions for bipedal robots has been solved successfully through whole-body optimization. Still, it is meaningless without a controller that is capable to track them while keeping balance on the real robot. In general, a reactive controller is used, e.g. a basic *Proportional-Integral-Derivative controller* (PID) or a more sophisticated admittance controller. Tuning is tedious and gets worse as the number of scenarios to tackle increases. It is a major issue for exoskeletons since the morphology of the patients is unknown and their behavior is unpredictable. Advanced control methods such as whole-body *Model-Based Predictive Control* (MPC)

could alleviate this issue substantially. It unifies planning and control under the same framework, thus the computational cost is much higher, and it does not play well with pre-defined nominal trajectories. The limited embedded computation power requires making assumptions and approximations to run fast enough for online execution, resulting in suboptimal and less natural motions.

On the contrary, *Reinforcement Learning* (RL) is very effective in simulation, and the policy is cheap to evaluate. It is a probabilistic method that is both gradient-based and model-free. It implies that any model for the system and the world can be used without any restriction, so it alleviates the reality gap by being more realistic in the first place. The closed-loop behavior is only a by-product of the maximization of the expected return: complex motions emerge from a small set of fundamental incentives rewarding the agent for every transition step individually. It eliminates human bias when designing gaits and is closer to human reasoning. It is also more generic because different tasks are likely to share most reward components. However, it is hazardous to anticipate the trajectory that the agent will perform. More generally, it is difficult to handle hard constraints or fine-tune the closed-loop behavior if necessary.

Offline trajectory planning with a theoretical model is very effective when it comes to generating a tightly constrained natural gait for a given task. Yet, it is almost impossible to anticipate and characterize mathematically all situations in an ever-changing environment. Falling to do so will certainly cause system failures. In this work, we overcome this limitation by learning a robust policy using RL. We present a method that aims to preserve the nominal trajectories through imitation, while giving enough freedom to go beyond from time to time if necessary. It is more conservative than end-to-end approaches and brings back expert bias, but in a good way. More specifically, it avoids fixing what has proven to be working and thereby prevents regression. The capability to enhance the rehabilitation of the patient has already been validated for the nominal motions, thus seeking to reproduce them whenever possible is reasonable. Besides, imitation guides the agent and enables solving complex tasks without advanced exploration strategies or reward engineering.

Imitation has been mostly ignored but has drawn attention recently in the field of computer-generated animation. Our method builds on the work of Ma et al. (2021). The key idea is to derive space-time bounds acting on a set of high-level features from nominal motions. These hard constraints are enforced for every transition step by aborting the episode as soon as any of them is violated. Hand-crafted reward components are not necessary anymore to reproduce locomotion tasks, thereby simplifying reward engineering and tuning. We generalize this approach to handle never-ending nominal motions and external disturbances. On top of that, we propose a method to reproduce accurately multiple locomotion tasks. Unlike Won et al. (2020), all tasks are learned at once as a single policy through worst-case improvement formulation. First, it inherently leverages common knowledge and correlations between similar tasks, hence it scales much better with the number of tasks. Secondly, it has the advantage to support both discrete and continuous task spaces out-of-the-box. Orthogonally, we also present an approach to infer stable transitions from any task to any other that can be triggered at any point in time. It does not entail an additional

computational burden and is effective regardless of the number of tasks. The main advantage is to avoid systematically going back to the resting state in between.

Safety and predictability are essential preconditions in robotics that are often overlooked in RL. To address this concern, we take advantage of technics that have been initially developed to certify the robustness to adversarial attacks in supervised learning. Unlike Jin and Lavaei (2020), we do not bound the Lipschitz constant of the network to avoid being overly restrictive from the start and penalizing exploration. Instead, we employ spatial regularization in line with the work of Shen et al. (2020), Zhang et al. (2020), and Cooman et al. (2021). As suggested by Mysore et al. (2021), an extra regularization term promoting temporal smoothness is also added. Together, they encourage the agent to focus on a few successful strategies and do nothing if the state is unrecoverable or simply unseen. Besides, they break the excitatory coupling between observation and control causing jerky and sporadic motions. The overall behavior would be safer, more predictable, and transfer to reality more easily. Finally, we enforce hard bounds on the predicted action to make sure the motors are within the well-tested operating range. It prevents large tracking errors that would be dangerous as it induces violent or uncontrolled motions.

Some emerging strategies would be very challenging to reproduce using classical model-based control. Moreover, the policy smoothly transfers to a real bipedal robot without additional tuning. We demonstrate safe and efficient push recovery behaviors for strong perturbations experimentally on the self-balanced medical exoskeleton Atalante carrying different users, as shown in the video[1].

6.1 Problem Setup

6.1.1 Learning Environment

Low-Level Control and Action Space

A distinctive feature in RL is the slow update frequency in contrast to classic control approaches, about 50Hz vs. 1kHz respectively. It gives enough time for the effect of the actions to build up and drift the future state away from the current one, which is beneficial in several aspects. First, it improves the signal-to-noise ratio of the individual transition steps by giving prominence to the underlying system dynamics over random effects that are mostly high-frequency noise. It indirectly speeds up learning because it significantly reduces the variance for the gradient estimate and allows for smaller training batches. Secondly, it greatly improves exploration efficiency when it relies on pure random noise added to the action. Temporally correlated noises such as Ornstein-Uhlenbeck stochastic process (Lillicrap et al., 2016) are not affected by this issue. Alternatively, Raffin et al. (2022) suggests using as noise a randomly parametrized linear function of the features extracted by the policy. The predicted sequence of actions would be smooth during training, enabling online RL from scratch

[1] https://youtu.be/HLx6CHfpmBM

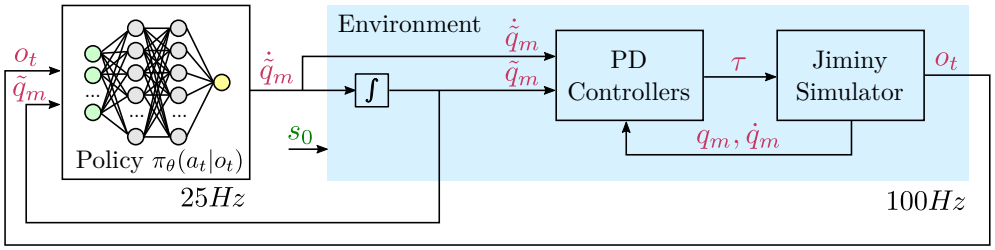


Figure 6.1: Overview of proposed control flow.

on real robots. Various other methods have been presented in section 3.2.2 but none of them is readily supported by the main learning libraries.

Predicting the motor torques u directly at such a low frequency is not recommended. In case of an unexpected event, the current state may have already drifted far away before the policy is given the opportunity to drive it back to the nominal state. This lack of responsiveness bounds the optimal performance. Besides, it is difficult to assert at runtime that the command torques are valid since it involves the dynamic model, which is very problematic for safety-critical devices. This issue can be partially addressed by post-processing the output of the policy with some readily available classic controllers. This *Low-Level Controller* (LLC) will be responsible for updating the motor torques at a high frequency based on the difference between the target and current quantities associated with some high-level features. On their side, these target quantities are a by-product of the action predicted by the policy. This way, the latter can keep being updated at a low frequency without rendering the closed-loop dynamics unstable.

In this work, we use simple decentralized low-level PIDs without integral term (cf. section 2.2.3). It follows that the command torque for each motor is proportional to their own tracking error in position and velocity in isolation. Altogether, it yields,

$$u_m = K_P((\tilde{q}_m - q_m) + K_D(\tilde{\dot{q}}_m - \dot{q}_m)), \tag{6.1}$$

where $\tilde{q}_m, \tilde{\dot{q}}_m$ (resp. $q_m, \dot{q}_m$) denotes target (resp. current) positions and velocities of all motors at once, and K_D, K_P are vectors of proportional and derivative gains. These gains are tuned to trade tracking accuracy for compliance. Notably, it limits internal shear forces and bending moments in double support that cause premature wear. Decoupled impedance control is well-known for its robustness to model uncertainties. Thus, this hybrid control architecture sensibly improves transferability.

There should be a temporal integration constraint between the target positions and velocities,

$$\tilde{q}_m(t_i + \Delta t) = \tilde{q}_m(t_i) + \tilde{\dot{q}}_m(t_i)\Delta t, \tag{6.2}$$

where t_i corresponds to the last time the target positions and velocities were updated by the policy, and Δt can be any positive duration before the next one. Satisfying this constraint is a prerequisite for accurate tracking, which is itself one of the most effective criteria to detect whether something is going wrong at runtime: the system

enters some degraded operation mode whenever the tracking error in position exceeds a given safety threshold since this situation is classified as potentially dangerous for the user. Yet, this consistency issue was disregarded in previous works, presumably because it does not hinder performance but rather interpretability and safety. Most often, the policy solely predicts the target positions, and the target velocities are all kept equal to zero (Siekmann et al., 2021b). Any consistent target positions and velocities can be rewritten in this way but not the other way around,

$$\tilde{q}'_m = \tilde{q}_m + K_D \dot{\tilde{q}}_m. \tag{6.3}$$

This relation shows that such target positions are plain computational artifacts without physical meaning. In particular, the tracking error in position is expected to be roughly proportional to the current motor velocities. Thus, they are doomed to exceed the acceptable safety threshold discussed in appendix A during fast motions.

Similarly, the target velocity is clipped first followed by the updated position to make sure it stays within the physical limitations of the system assuming perfect tracking. It implies that the motors cannot output a torque forcing against the mechanical stops if already close to them. It is double-edged since pushing within bounds might be needed to oppose external forces, but it prevents hitting and damaging them as long as the tracking is accurate.

Our exact control architecture is described in figure 6.1. The policy predicts target motor velocities $\dot{\tilde{q}}_m$ at $f_h = 25\text{Hz}$. The target motor positions are integrated from their previous values at every update of the LLC according to equation (6.2), whereas the target motor velocities are held constant. The LLC is running at $f_l = 100\text{Hz}$. It slightly impedes the performance compared to 1kHz, but it allows for taking larger integration steps in simulation to speed up learning. This approach is equivalent to a safety layer (Dalal et al., 2018). Assuming the current target motor positions are part of the observation, this safety layer would concurrently output the whole sequence of future target motor positions and velocities for the LLC until the next update of the action, while ensuring the consistency between them.

In place of the velocity, any higher-order derivative could be predicted and integrated. The higher the order, the smoother the target positions and velocities (and subsequently the command torques). However, a discrete integrator acts as a low-pass filter with a lower cut-off frequency as the order increases. This property makes the system less reactive and thereby harder to control because the effect of the action takes more time to build up. The agent would be harder to train as the side effect of random exploration being canceled out more aggressively (cf. section 4.2.3). In practice, any higher order than the velocity impedes performance.

Tracking can be made more accurate without instability by jointly increasing the PID gains and the update rate of the LLCs. Yet, it is undesirable because it slows down the simulation and puts more pressure on the hardware. Alternatively, Siekmann et al. (2021a) are incorporating the PID gains in the action predicted by the policy. It enables adapting the gains based on the apparent inertia to set in motion, effectively turning the decentralized PIDs in coupled *Linear Quadratic*

Regulators (LQRs) at joint level tuned optimally under the local closed-loop whole-body dynamics. This way, the same tracking accuracy and response time for all motors can be achieved without causing instability whatever the system state. This property is very useful for legged robots as the inertia of the subtree is substantially different between stance and flying legs. If a single centralized PID is controlling all motors at once, then it would behave as a locally optimal whole-body LQR. It is even more powerful but the gains would be dense positive semi-definite matrices. Any real-valued vector can be interpreted as a compact encoding of such matrices since the product of any lower triangular matrix and its transpose is always positive semi-definite. However, it increases the dimensionality of the action space, and hence the policy may be challenging to train. Moreover, the command torques would undergo discontinuities each time the gains are modified. It is important to make sure the gains are smoothly varying to avoid inducing jolts in the mechanical structure and loud noise. The smoothness conditioning approach presented in section 6.4.4 is well suited to address this specific concern.

State and Observation Spaces

The observation should characterize uniquely the current state of the agent in the world for the learning environment to be fully observable. Here, the state of the agent gathers the ones of the patient and the exoskeleton, each of them having its own dynamics coupled together by straps. Nevertheless, we suppose as before that the patient is rigidly fastened, so providing only the state of the exoskeleton is sufficient. This hypothesis is dubious for the upper body. Pseudo-periodic external forces are applied at the pelvis to mitigate this discrepancy and improve the robustness to the reality gap. Even so, the full state of the agent s_t is defined by:

- the position p_b, orientation (Roll ψ_b, Pitch θ_b, Yaw ϕ_b), linear velocity v_b and angular velocity ω_b of the pelvis,
- the motor positions q_m and velocities $\dot{q}_m$,
- the internal state of LLCs if any,
- the state of the mechanical deformation and backlash if any,
- the external disturbances if any,
- the surrounding ground profile,

In reality, the state must be reconstructed from raw sensor data. The available post-processed sensor data are presented in section 1.1.3. They are all proprioceptive and many insightful quantities cannot be reliably estimated without exteroceptive sensors, e.g. the pelvis height z_b and linear velocity v_b. They are not included in the observation space because a significant mismatch between simulated and real data may prevent the transfer to reality. Even though the odometry pose $p_o := [x_b, y_b, \phi_b]^T$ is not observable, it is not really limiting as the recovery strategies should be invariant to it. The observation $o_t \in \mathcal{O} \in \mathbb{R}^{49}$ comprises:

- the roll ψ_b, pitch θ_b and angular velocity ω_b of the pelvis,
- the total vertical forces acting on each foot in local frame ${}^rF_r^z, {}^lF_l^z$,

- the current motor positions q_m and velocities $\dot{q}_m$
- the current target motor positions $\tilde{q}_m$

Note that the target motor positions must be part of the observation because they constitute the internal state of the LLC.

Asserting observability of the roll and pitch using *Inertial Measurement Units* (IMUs) is misleading since the raw data are the angular velocity and classical angular acceleration. The orientation can be estimated using the nonlinear complementary filters introduced by (Mahony et al., 2008). Originally, it is only valid in an inertial frame of reference. It induces a state-dependent bias, partially corrected by Vigne et al. (2022) by considering that one foot is fixed in the world at all times. Alternatively, one could only provide this information and let the policy infer the orientation. However, it would impede the performance significantly, unless data are accumulated over several time steps, because the acceleration is very noisy. Similarly, we chose to be conservative and completely ignore the IMUs in the feet, although they could be used to observe indirectly the combined effect of the mechanical deformation and backlash. The orientation estimation is not reliable for a short duration after impact on the ground and modeling this phenomenon is out of reach.

Those quantities are heterogeneous and have different scales. It is essential to normalize them to avoid giving more weight to some features relative to others. It would be hazardous to do it manually because their distribution is conditioned by the current policy, and it may change dramatically over training iterations. Hence, all quantities are independently normalized over training batches.

6.1.2 Training Scenarios

Two very different training scenarios have been considered:

- reactive stepping for emergency push recovery while standing
- nominal gait learning with smooth transitioning

The first scenario is about handling strong external pushes on flat ground without falling. Pushes may not be foreseen in reality, thus we assume it is always the case to be conservative. It means that the recovery strategies must be purely reactive. The nominal standing pose has been designed with comfort in mind rather than stability. It is not the most favorable setup but comes close. Therefore, it mainly serves as a baseline of what can be expected at best in a realistic situation, i.e. while moving around. Pushes have variable magnitude and direction, so that doing steps is occasionally the only way to keep balance. They are applied to the pelvis at the hip level. It does not create momentum since this point is almost co-located with the *Center of Mass* (CoM). On the contrary, the tip of the pelvis would be way more challenging but less sensible. Predicting zero velocity on average maintains the initial position of the motors indefinitely. Depending on how the policy is initialized, it would occur right at the beginning of the learning process. It entails that the robot is already stable in the absence of disturbance granted that the initial pose already is. It facilitates learning in contrast to having to learn how to compensate the gravity

already. RL is well-known to be capable of addressing such local optimization problem in simulation. The main difficulty is being able to transfer the policy on a real device. At the very least, the emerging strategies must be safe no matter what and comply with the specification of the hardware.

The second scenario is more prospective and mixes imitation with classical RL. The objective is to learn multiple locomotion tasks at once and equally well with a single control policy. The set of periodic primitive motions presented earlier in section 2.2.1 has been pre-computed using offline trajectory planning. As for the resting pose, each of them is specially tailored to be comfortable and human-like. Their capability to enhance rehabilitation has been validated clinically. However, some of them are not intrinsically stable without external assistance, starting with the foot rolling walk. Besides, it is not versatile enough to move around naturally on flat ground because of the lack of seamless transitions between them without going back to rest. The agent must reproduce the nominal motions faithfully as long as it does not jeopardize stability to the point of falling and generalize them with transitions that can be triggered at any time. Robustness to model uncertainties and compounding of errors is also expected. It is already challenging on its own, so we set aside the robustness to external disturbances and transfer to reality.

It is not straightforward to merge both scenarios to offer versatile locomotion skills that would be robust to external disturbances. In the standing pose, it is more or less possible to push in any direction and at any time without issue. It is no longer true while moving because the projected support polygon changes dramatically. Overall, the same level of robustness cannot be expected in single vs. double support. It seems necessary to rely on some kind of curriculum learning for scheduling automatically the magnitude of the pushes based on their direction and the current state of the robot. Yet, it has been observed that the learned motions degenerate and the robot drags its feet if pushes are too strong but recoverable. How to get around this issue is an interesting research direction for future work. Anyhow, we propose a unified training framework that is capable of solving both scenarios using the same hyperparameters.

We use the simulator Jiminy (Duburcq, 2019) based on Pinocchio (Carpentier et al., 2019). It has been developed during this thesis and is presented in appendix C. In this work, only the average patient model is studied. It could be easily extended, providing that the morphology of the patient is forwarded as input to the policy.

6.2 Constrained Policy Optimization

Being able to enforce constraints is a central part of our work as they are involved in both imitation and safety (cf. sections 6.3 and 6.4). The main criteria to partition them is whether they are instantaneous, cumulative over complete episodes, or probabilistic. The question is distinct in each case, and approaches that have been developed for one do not translate to others effortlessly. Judging from their ubiquity in trajectory planning and optimal control, instantaneous constraints are arguably more important than cumulative or probabilistic ones. Yet, how to deal with them

has been largely disregarded in the context of RL. Very few methods with limited application scope have been proposed so far. Conversely, dealing with cumulative constraints is well-studied in the literature. It may be explained by the difficulty to handle violation that is bounded to happen in general, and the lack of a reliable gradient estimate as it is state-dependent and thereby cannot be averaged over an episode. We are solely interested in instantaneous constraints in the following. They are said to be explicit if they have an analytical closed-form expression that is known. This way, the constraint function plus its gradient can be evaluated ad lib. for any state. Otherwise, they are implicit, and only their binary feasibility status for all encountered states is accessible. Explicit and implicit constraints are handled differently to get the best out of the available information.

We assume that the reward function is bounded and normalized, i.e. $\forall (s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}, R(s, a, s') \in [0, 1]$. Note that scaling does not affect the solution, as opposed to the mean. The larger the ratio of the mean reward over its scale, the stronger the agent is encouraged to settle for surviving instead of optimizing the average reward. In this case, the maximum improvement and instantaneous reward are both equal to one, and so whether the agent will favor one or the other in practice will depend on the current reward distribution. Yet, the reward is always positive, which implies that the agent is always marginally encouraged to survive rather than killing itself. It is crucial for complex tasks for which the policy has very bad performance initially.

6.2.1 Implicit Constraints through Early Termination

We first study the handling of implicit constraints. There are not many options considering that the actual constraint violations for every transition step are unknown. One approach consists in including the feasibility information in the reward itself to treat the policy optimization problem as if it was unconstrained. The resulting additional sparse reward component would be equal to some positive constant if all the constraints are satisfied and zero otherwise. It can be interpreted as an instance of the classical penalty method. Although intuitive, this method is flawed for several reasons. First, it is said to be inexact, which means that the constraints are not strictly enforced and can be slightly violated here and there. Next, the penalty competes against the original reward directly. Weighting it properly to give prominence to the constraints and adjust the maximal violation is tedious as it is problem-specific and usually sensitive. Finally, states encountered after any violation may not be relevant or meaningful anymore for the rest of the episode if some constraints characterize critical failure. It impedes the sample efficiency and corrupts the gradient estimate of the return. Avoiding such failure via external guidance is only doable in simulation and often challenging to implement. Moreover, it does not play well with learning as it makes it harder for the agent to induce the effect of its actions. For example, naively adding a virtual physiotherapist to avoid failing and penalizing the agent for relying on it when walking never converges to an autonomous policy.

We propose instead to rely on early termination as a generic framework to enforce black-box soft constraints whose analytical formulation is inaccessible. The idea is

to abort an episode as soon as a constraint is violated, preventing the reward from accumulating any longer and capping the return to a fraction of a maximum that depends on the discount factor. This technique is mainly used to avoid bad local minimum and speed-up convergence (Won et al., 2020). The work of Ma et al. (2021) on space-time bounds brought out remarkably that it is much more powerful than that. It circumvents the two limitations of the previous approach: the training batches are inherently free from irrelevant data, and satisfying the constraints has a higher level of priority than maximizing the reward since it is a pre-condition for accumulating it. It has a minimal information access requirement but a slower convergence rate than explicit approaches. Nevertheless, it was never rigorously analyzed to the best of our knowledge, and it raises two interrogations: toward which solution it converges? can it be adjusted to make the constraint hard or soft?

We thereupon demonstrate that it is equivalent to solving a given constrained policy optimization problem and introduce an optional hyperparameter that specifies how tightly the constraints must be satisfied at a cost of worse problem conditioning. Let us suppose that horizon of the *Markov Decision Process* (MDP) is infinite or long enough to be considered as such, and the objective to maximize is the sum of the discounted rewards in expectation. Intuitively, the agent learns to avoid states leading to early termination as it impedes the return. More precisely, the agent becomes risk-averse to some extent because the world is stochastic: it requires extra safety to be confident about preventing such critical failure. Surprisingly, it does not stand out from the math. It is rather the contrary.

As a reminder, the objective $J(\pi)$ is to maximize the expected return $R(\tau)$ over the trajectories τ induced by the policy π or equivalently the expected reward $R(s, a, s')$ over the discounted stationary state distribution ρ_π (cf. appendix E.5),

$$J(\pi) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)] = \mathop{\mathbb{E}}_{s\sim\rho_0}[V_\pi(s)] = \mathop{\mathbb{E}}_{s\sim\rho_\pi}\Big[\mathop{\mathbb{E}}_{\substack{s'\sim P\\ a\sim\pi}}\big[R(s, a, s')\big]\Big]. \tag{6.4}$$

Formally, early termination at time t can be formulated as reaching a virtual absorbing state s_c for which the reward is always zero regardless the action, namely $V_\pi(s_c) = 0$ and $\rho_\pi^{(t')}(s \to s_c) = 1$ if $t' \geq t, 0$ otherwise. It appears that termination happening a while after the episode started barely affects the discounted state distribution and thus has no impact on the objective. To frame it differently, failing is not a big deal once the maximum return has been obtained asymptotically.

Constraints can be enforced by returning a negative reward $-r_c$ at termination. Let ϵ denotes the probability to violate a constraint under the current policy, i.e. $\mathop{\mathbb{E}}_{a\sim\pi}[\mathbb{P}(s_{t+1} = s_c | s_t \neq s_c, a_t = a)]$. The probability to terminate at time t is given by,

$$\mathbb{P}(T = t) = \begin{cases} \epsilon(1-\epsilon)^{t-1} & \text{if } t \leq 1 \\ 0 & \text{otherwise} \end{cases}. \tag{6.5}$$

Assuming that the constraints can be satisfied exactly, the optimal value function is guaranteed to be positive because the reward is always positive. The worst-case

probability to violate the constraints can be derived from this observation:

$$\begin{aligned} 0 \leq V^*(s) &\leq 1 + \sum_{t=1}^{\infty} \left[\left(\sum_{t'=0}^{t-1} \gamma^{t'} - \gamma^t r_c \right) \epsilon (1-\epsilon)^{t-1} \right] \\ &\leq 1 + \sum_{t=1}^{\infty} \left[\left(\frac{1-\gamma^t}{1-\gamma} - \gamma^t r_c \right) \epsilon (1-\epsilon)^{t-1} \right] \\ &\leq 1 + \frac{\epsilon}{1-\epsilon} \left[\frac{1}{\epsilon(1-\gamma)} - \left(\frac{1}{1-\gamma} + r_c \right) \frac{1}{1-\gamma(1-\epsilon)} + r_c \right] \end{aligned}$$

It yields,

$$\forall r_c > \frac{2}{\gamma}, \ \epsilon \leq \frac{2-\gamma}{\gamma(r_c - 1)}. \tag{6.6}$$

The higher the termination reward, the stronger the incentive for the optimal policy to satisfy the constraints. After increasing the termination reward, the probability to violate the constraints for the optimal policy cannot be larger than before. This probability is likely to be lower in practice but there is no guarantee. Either way, any value of the termination reward larger than $2/\gamma$ surely translates into a worst-case probability to violate the constraints for the optimal policy. This threshold increases as the discount factor decreases, which makes sense since long-term effects would have less impact on the objective.

If the above condition is met, then the policy gradient will necessarily steer the parameters in a direction that would reduce the constraint violations. Increasing the termination reward beyond this threshold would further reduce the maximum constraint violations but lead to ill-conditioning at some point. Hence, the constraints cannot be enforced exactly. Besides, bounding the probability to violate the constraints does not bring any guarantee regarding the actual constraint violations as this information is not even accessible. Nevertheless, under sufficient regularity assumption regarding the constraints, reducing the probability to violate them incidentally reduces the actual violations themselves.

It was implicitly assumed the whole time that state-of-the-art policy gradient algorithms converge to the globally optimal policy, which is highly debatable. They are all performing local searches in the first place and are notoriously unstable. But more importantly, their update rules involve a biased estimator of the policy gradient. As discussed in appendix E.5, Nota and Thomas (2020) has shown that nearly all state-of-the-art policy gradient algorithms are in fact following the gradient of the value function in expectation over some weighted stationary state distribution,

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{s \sim \rho_0} \left[\nabla_\theta V_\pi(s) \right] \neq \mathop{\mathbb{E}}_{s \sim \tilde{\rho}_\pi} \left[\nabla_\theta V_\pi(s) \right], \tag{6.7}$$

where $\tilde{\rho}_\pi = \sum_{k=0}^{\infty} \omega_k \mathbb{P}(s_t = s)$ with $w_0 = 1, w_i = 1 - \gamma \ \forall i \geq 1$. This gives the same weight to all k-steps state distributions except the initial one. As a result, these learning algorithms come closer to optimizing the expectation of the average reward

than the discounted one. The discount factor does not vanish completely and still appears in the computation of the value function. This gives the desired sense of short-term vs. long-term profit while reducing the variance of the gradient estimate by putting more trust in recent data (Thomas, 2014).

This unforeseen property turns out to be the key that enables leveraging early termination to enforce soft constraints without termination reward. The value function is considered equal to zero for the final state preceding early termination, i.e. $V_\pi(s_T) = 0$. This way, early termination would have a significant impact on the update direction no matter when it happens. The constraints are not guaranteed to be strictly enforced, as violating them does not necessarily lead to a decrease of the value function in expectation over the weighted stationary distribution. Nevertheless, we observe in practice that policies trained without termination reward satisfy the constraints exactly for our standing push recovery scenario in simulation (cf. section 6.3.1). This includes safety margins in the nominal case so that the constraints do not get violated when an unexpected event occurs. These promising results suggest the worst-case reasoning is pessimistic for real-world applications.

6.2.2 Explicit Constraints through Barrier Functions

We now study the handling of explicit constraints. In that respect, the associated analytical functions and derivatives can be freely evaluated within the learning algorithm. Constrained optimization methods leveraging this additional ability are expected to provide stronger guarantees compared to early termination.

Classically, *Interior Point Optimization* (IPO) performs Newton steps at every iteration until convergence for a given α. It prevents constraint violation after each update and speeds up convergence by scheduling the step size optimally and automatically. It is a second-order method: the Hessian matrix of the problem is evaluated and inverted to compute the step size. As such, it is not viable for deep learning applications for which the models have millions of parameters. Indeed, the time complexity of matrix inversion is at least quadratic, and the Hessian is a square symmetric matrix whose dimension is the number of training parameters. To get around this limitation, Liu et al. rely on the usual *Stochastic Gradient-Descent* (SGD) method. In this case, the feasibility of the policy after its update is not guaranteed. In general, it would be impossible to provide such a guarantee anyway. First, the distribution of states in training batches is often not representative of the true discounted state distribution. Then, we are resorting to the hypothesis of perfect tracking for constraints involving the state of the robot.

They claim that IPO is applicable regardless. The initial policy does not even have to be feasible. They evaluated their method on a few common toy problems. The constraint violation is large at first but decreases asymptotically over iteration. After convergence, the constraints are always satisfied in practice. As a matter of fact, the policy is slightly more conservative than the other constrained policy optimization algorithms for explicit cumulative constraints. It is not surprising because the logarithmic barriers are always pushing toward the center of the feasible domain.

In spite of that, it achieves higher return on expectation. They conjecture that it is because IPO avoids premature convergence and thereby bad local minima. Unlike other algorithms, it does not focus exclusively on reducing the constraint violation if any but keeps optimizing the return on the side.

Sadly, how to handle ill-conditioning or ill-defined logarithms is never explained by the authors and the source code is not available. They are likely clipping the constraint function $J_\pi^{c^i}$ to make sure it is always negative before composition with the barrier function. However, no gradient could be back-propagated when triggered. Some feasible points may be discovered through exploration, so it may eventually improve over time as the gradient would be non-zero in expectation. If it is not sufficient, they suggest splitting the learning algorithm into two stages. First, the original reward is ignored and replaced by a single constraint. Once satisfied, it is made into a penalty in the surrogate loss and another constraint is considered. Secondly, IPO is applied, so that the original reward is taken into account. It is appropriate as long as a feasible policy exists, but it brings back potential premature convergence issues.

We propose the following loss function for handling instantaneous constraints c^i,

$$L^c = \mathop{\hat{\mathbb{E}}}_{\substack{s_t \sim \bar{\rho}_\pi, a_t \sim \pi_\theta \\ s_{t+1} \sim P}} \left[\sum_{i=1}^{p} g(c^i(s_t, a_t, s_{t+1})) \right], \tag{6.8}$$

where $\hat{\mathbb{E}}$ denotes the empirical mean over all the transition steps in a given training batch $\mathcal{B}$. It neglects the dependency between the policy and the task distribution, as the constraint should be enforced equally for all states anyway,

$$L^c = \frac{1}{|\mathcal{B}|} \sum_{e_t \in \mathcal{B}} \sum_{i=1}^{p} g(c^i(s_t, a_t, s_{t+1})), \tag{6.9}$$

where $e_t = \{t, s_t, a_t, r_t, s_{t+1}\}$ is a single transition step. The barrier function g has been moved inside the expectation, and the true state distribution has been swapped for the discounted one. It obviates the discount factor that is irrelevant for constraints, while keeping bounded the expectation.

As intended, it approaches infinity on the boundary of the feasible domain of any constraint at any transition step. Basically, it explodes as soon as a constraint is violated at a single point, which boils down to $\bar{\rho}(s) = 0$ for all infeasible states. Being able to find a trajectory that satisfies all the constraints by local exploration was already questionable when cumulative, but it is far more challenging here. One way to overcome this issue is to rely on a barrier function g that is well-defined and has a non-zero gradient outside the feasible domain. The exponential barrier function is a legitimate candidate,

$$g(x) = -e^{\alpha x}. \tag{6.10}$$

Despite being well-defined for any value of x, we suggest clipping the input $x < 5/\alpha$ for numerical stability. It extends the domain of non-zero gradient beyond the

boundary of the feasible one, which should be enough to find a valid policy by local exploration in most cases. Alternatively, some authors are replacing the tail of the classical logarithmic barrier with another function,

$$g(x) = \begin{cases} \log(-x)/\alpha, & \text{if } x \leq \delta \\ h_\delta(x)/\alpha & \text{otherwise} \end{cases}. \tag{6.11}$$

The two main classes of relaxing functions found in the literature are the polynomial $h_{k>1,\delta}$ and exponential $h_{e,\delta}$ ones (Feller & Ebenbauer, 2017):

$$h_{k,\delta} = \frac{k-1}{k}\left\{\left(\frac{k\delta + x}{k\delta - \delta}\right)^k - 1\right\} - \log(\delta) \tag{6.12}$$

$$h_{e,\delta} = \exp\left(1 + \frac{x}{\delta}\right) - 1 - \log(\delta) \tag{6.13}$$

We prefer the exponential barrier function for simplicity. Cominetti and Dussault (1994) have proven that it has similar properties to the logarithmic one in terms of convergence rate and accuracy of the solution with respect to the original constrained problem.

As a result of this relaxation, the optimal policy may slightly violate the constraints from time to time. In general, it is not blocking since the threshold associated with each constraint is fairly arbitrary in the first place. It is impossible to obtain the same worst-case guarantees as *Constrained Policy Optimization* (CPO) (Achiam et al., 2017). Still, there is a relationship between the maximum constraint violation c^i and the associated probability for the exponential barrier function:

$$-g(\epsilon)\mathbb{P}\left(\max_{1\leq i\leq p}(c^i) > \epsilon\right) < L^c < \frac{1}{1-\gamma} \implies \mathbb{P}\left(\max_{1\leq i\leq p}(c^i) > \epsilon\right) < \frac{e^{-\alpha\epsilon}}{1-\gamma}$$

A much tighter bound could be obtained under the assumption of continuity for both the constraints and the system dynamics. It is left to the interested reader.

6.3 Trajectory-Based Imitation using Reinforcement Learning

As mentioned before, the second case study mixes imitation learning with classical RL. A set of periodic primitive motions have been pre-computed offline for a simplified theoretical model on flat ground. As is, they are not dynamically stable, neither in simulation nor on the real device. The goal is two-fold: mimicking each nominal trajectory as accurately as possible while keeping balance, and designing a unified policy capable of doing so for all motions at once along with transitioning between them. Those two aspects are orthogonal and treated separately in the following.

6.3.1 Generalized Space-Time Bounds

Instantaneous

Ma et al. (2021) have introduced a simple yet impressive method for imitation using RL. In particular, they were able to generate natural motions that are dynamically stable for a humanoid in simulation by leveraging motion capture recordings. The core idea is simply to restrict the deviation between the actual and nominal trajectories. Simply put, it consists in enforcing a set of constraints called space-time bounds acting on the state at unevenly distributed timesteps. These constraints are enforced via early termination without terminal reward. This method has the major advantage to be capable of finding a sensible policy without reward engineering. It is usually enough to provide a sparse reward returning $+1$ systematically until failure or reaching the time limit T_f. On the contrary, coming up with hand-crafted imitation reward components is known to be challenging and time-consuming as they must be specifically tailored for one application in general.

Let $s_\tau \in \mathcal{S}^{T_f}$ denotes the sequence of states $(s_t)_{t=0}^{T_f}$ over a complete episode τ and $\hat{\star}$ refers to nominal quantities. Let us consider a constraint function $b^i : \mathcal{S}^{T_f} \times [0, T_f] \to \mathbb{R}$ defined as the distance between the current and nominal states at one particular timestep in a given feature space,

$$b^i(s_\tau, t) = d(f(s(t)), f(\hat{s}(t))), \tag{6.14}$$

where f is responsible for extracting the feature of interest from a given state and d is a metric. For example, it could be the absolute position of the CoM together with the classical L^2-norm.

The space-time bounds characterize the subset of complete trajectories $\mathcal{B}_\tau^i \subset \mathcal{T}$ that are satisfying the constraint at a set of independent breakpoints I up to some tolerance threshold σ_i for each breakpoint i,

$$\mathcal{B}_\tau^i = \{\tau \in \mathcal{T} \mid \forall i \in I,\, b^i(s_\tau, t_i) \leq \sigma_i\}. \tag{6.15}$$

Assuming the system dynamics is continuous, a finite set of breakpoints is sufficient to limit the maximum deviation globally at any point in time, and it gets tighter as their number increases. Still, this definition can be extended to a continuum easily,

$$\mathcal{B}_\tau^{i'} = \{\tau \in \mathcal{T} \mid \forall t \in [0, T_f],\, b^i(s_\tau, t) \leq \sigma(t)\}. \tag{6.16}$$

They rely exclusively on this second formulation in practice. It is more generic and enables limiting the maximum deviation irrespective of the Lipschitz constant of the system dynamics. They are only considering a pure imitation setup without any disturbance, so they can afford to be more restrictive.

For a policy to be a feasible solution to the RL problem, it must be an element of the subset of policies $\mathcal{B}_\pi^i \subset \Pi$ that induces only valid trajectories for initial state distribution ρ_0. It yields,

$$\mathcal{B}_\pi^i = \{\pi \in \Pi \mid \forall \tau \sim \pi,\, \tau \in \mathcal{B}_\tau^i\}. \tag{6.17}$$

Probabilitistic

Defining bounds that only depend on the current state without history is problematic. It entails that the only way to achieve accurate imitation is confining the actual trajectory within a tube around the nominal, and thus having tight bounds $\sigma(t)$ during the whole motion. Otherwise, the agent will surely abuse it. First, a slowdown in the world plane is expected compared to the nominal motion because it is easier to maintain and recover balance when doing steps as small as possible. Walking in place may even be a local optimum for locomotion tasks on hazardous terrains. Secondly, the robot lacks exteroceptive sensors to locate the ground without touching it, and knowing it would be helpful to keep balance. Hence, the agent is going to walk warily and drag them during the whole motion if permitted. Conversely, handling unexpected events inevitably induce transient dynamics that are deviating locally from the nominal by a wide margin. These two objectives are conflicting and cannot be reconciled. Thus, their formulation is suitable for locomotion on flat ground with 0-step capturable disturbances but does not allow for reactive stepping strategies.

Besides, setting the time-dependent tolerance $\sigma(t)$ for each feature requires prior knowledge. From an imitation perspective, the smaller, the better. However, it shrinks the whole feasible domain for which the constraints associated with the physics and the space-time bounds must be jointly satisfied, and it may be empty at some point. Consequently, it is increasingly difficult to find a valid policy by random exploration. The optimal trade-off between imitation accuracy is feature-specific (not least because it involves different *Degrees of Freedom* (DoFs)), and hence tuning the tolerances accordingly is tedious and time-consuming. For instance, it may be expected to reproduce perfectly the nominal trajectory of the feet in the absence of disturbances but not the one of the relative joint angles, owing to the mechanical deformation among other kinematic discrepancies.

To tackle these limitations, it is necessary to take into account past events implicitly or explicitly. We propose to only require the instantaneous space-time bounds to be satisfied once in a while. Mathematically, it amounts to bounding the minimum deviation over a sliding time window Δ,

$$b^p(s_\tau, t) = \min_{t' \in [t-\Delta T, t]} b^i(s_\tau, t'), \tag{6.18}$$

$$\mathcal{B}_\tau^p = \{\tau \in \mathcal{T} \mid \forall t \in [\Delta T, T_f], b^p(s_\tau, t) < \sigma\}. \tag{6.19}$$

Strictly speaking, it breaks the Markovian assumption since the transition probability now partially depends on past events and not just the current state. Yet, it has no effect apart from seldom early termination events. The actual dynamics is unchanged and remains markovian if it was, so it is not a major concern.

This constraint is more permissive than a finite set of evenly spaced breakpoints. Formally, it is equivalent to letting the agent freely choose breakpoints at which the instantaneous space-times bounds must be satisfied as long as they are not further apart than ΔT. Notwithstanding, it strongly encourages only deviating from the nominal if absolutely necessary and canceling unexpected events as fast as possible.

The reason is that the agent does not have access to the history of past states. It has no other choice than to seek to satisfy the instantaneous space-time bounds all along, otherwise it would risk violating the constraint at some point. In theory, it could also oscillate periodically. The current time is missing in the observation space, but the current nominal state is part of it and could serve as a clock. This contingency is ruled out by smoothness conditioning promoting minimal action (cf. section 6.4.4). Anyway, it is likely easier to discover a policy mimicking accurately the nominal rather than oscillating around it on purpose to trick the constraint.

Our space-time bounds can be expressed as a classical probabilistic constraint. It amounts to bounding the probability to violate the instantaneous space-time bound at every transition step individually and without history,

$$\mathcal{B}^{p'} = \{\tau \in \mathcal{T} \mid \forall t \in [0, T_f], \mathbb{P}(b^i(s_\tau, t) \geq \sigma) < \Delta T^{-1}\}. \tag{6.20}$$

It is a kind of soft relaxation over enforcing an instantaneous hard constraint, giving enough freedom to deviate locally from the nominal without triggering early termination systematically. This formation is convenient for mathematical analysis but impracticable. The actual distribution is computationally demanding to assess during training and completely unknown to the agent.

The interval ΔT must be short to urge the agent to promptly cancel the effect of disturbances. If too long, then the behavior of the policy may feel unnatural and unconformable for the patient. Typically, the robot may stumble for longer than necessary to maximize the return, for instance to avoid hitting the joint bounds (cf. section 6.4). Still, it must be long enough so as not to lower the capability of the agent by preventing doing enough recovery steps.

Our formulation is very robust to the value of maximum acceptable deviation σ as opposed to the instantaneous one. Notably, a feasible policy may exist even if σ is equal to zero. As a result, we make it a constant for simplicity.

Cumulative

The original space-time bounds were designed with trajectories of finite duration in mind. Therefore, it is possible to make sure the initial and final states at time 0 and T_f respectively match accurately the nominal ones. In the context of locomotion, we are intrinsically dealing with an infinite-horizon MDP since the objective is to keep moving indefinitely. A naive approach would be viewing each periodic primitive motion as a usual trajectory of finite duration by considering a fixed number of steps. Somehow it works, but it does not prevent long-term drift. This phenomenon is unacceptable since the motions are specially generated for a set of high-level features that includes the desired average velocity in the world plan.

Tightening the space-time bounds uniformly reduces the drift but does not allow for deviating from the nominal whenever it is necessary to recover balance. Alternatively, tightening them at the endpoints breakpoints only is not any better. First, their exact timings are often meaningless since there are no actual endpoints to periodic motions, but they still have an impact on the resulting policy that is hard to

predict. Next, it would be much harder for the agent to discover a feasible policy as the information provided to the agent is sparse. The agent has no clue whether it is currently drifting until reaching the next breakpoint, where it may fail because of it. Finally, it encourages the agent to catch up on the delay that may have accumulated a while ago due to local shifts following unexpected events, which is undesirable. A better compromise may be achieved by combining both, but it is not solving any of these issues. Increasing the number of steps is not going to help much either. In theory, it prevents drifting at the limit, but the duration of the episodes must not exceed about 30s to maintain healthy sample diversity.

More generally, what happened long ago and where the robot really is in space are irrelevant when dealing with locomotion tasks. Only recent events should be taken into account regardless of the episode duration when assessing the current policy. It follows that any reward component or constraint should be agnostic to the past. One option would be to consider the instantaneous spatial velocity of the features instead of the pose. However, it prohibits fast and short bursts of velocity. They have a limited impact on the drift. Yet, they are necessary to react rapidly and thereby recover balance efficiently. Bounding the deviation of the average velocity over a time window combines the best of all worlds: it prevents long-term drift without impeding the recovery capability nor giving precedence to any point in time in particular. Mathematically, we propose to act on the deviation of the variation over a sliding time window ΔT. It yields,

$$b^c(s_\tau, t) = d([f \circ s]^t_{(t-\Delta T)^+}, [f \circ \hat{s}]^t_{(t-\Delta T)^+}), \tag{6.21}$$

$$\mathcal{B}^p = \{\tau \in \mathcal{T} \mid \forall t \in [0, T_f], b^c(s_\tau, t) < \sigma\}. \tag{6.22}$$

where $\star^+ = \max(\star, 0)$ and $[\star]^{t_2}_{t_1} = \int_{t_1}^{t_2} \dot{\star} = \star(t_2) \ominus \star(t_1)$ denotes the variation along the current trajectory over the interval $[t_1, t_2]$. The latter is linearly increasing until it reaches ΔT. It can be interpreted as a kind of automatic curriculum learning schedule based on the actual capability of the agent. The space-time bounds would be less restrictive at first, giving prominence to dynamic stability. Then, once the robot is able to stand up and move without falling, reducing the drift will be more pressing. The maximum deviation σ is not time-dependent for simplicity as scheduling it would serve the same purpose.

The maximum interval ΔT is not related to the period of the primitive motion. It must be long enough for averaging to take effect, otherwise the space-time bounds would be prone to false positives caused by successive pushes. The effect of unexpected events would also cancel out naturally if homogeneous, so that the agent is no longer encouraged to catch up with local shifts anymore. It is approximately true for a random ground profile that is flat on average or external pushes whose orientation is sampled uniformly. Any value significantly longer than both the pseudo-period of external disturbances and subsequent transients is fine. There is no advantage beyond this point. Then, the maximum deviation is chosen accordingly.

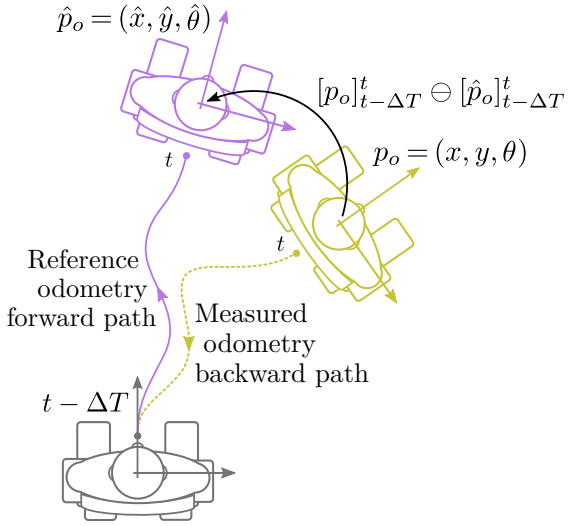


Figure 6.2: Probabilistic constraint on motor position tracking error

Application to robust locomotion

Nominal Dynamics. Long-term drift of the odometry position is inevitable, but it must be limited. The user may have to rectify it manually but this should remain casual. We restrict the odometry displacement through cumulative space-time bounds over a time window of $\Delta T = 20\text{s}$ (see figure 6.2),

$$\mathcal{B}_o^c = \left\{\tau \in \mathcal{T} \mid \forall t \in [0, T_f], \left|[{}^w p_o]^t_{(t-20\text{s})^+} \ominus [{}^w \hat{p}_o]^t_{(t-20\text{s})^+}\right| < [2.0\text{m}, 3.0\text{m}, 180\text{deg}]\right\}, \tag{6.23}$$

where the vector inequality must be understood element-wise. The constraint is satisfied if and only if the condition for all of its components is met.

Transient Dynamics. The robot must track the nominal if there is no hazard, only applying minor corrections to keep balance. Rewarding the agent for doing so is not effective as favoring robustness remains more profitable. Indeed, it would anticipate disturbances, lowering its current reward to maximize the expected future return, primarily averting falling.

We enforce probabilistic space-time bounds at the joint level. Since the configuration of each joint is its relative angle with respect to its parent, the errors at the joint level propagate along the kinematic chains and are maximal at the end-effectors. Considering all the joints at once mitigates this phenomenon (see figure 6.3),

$$\mathcal{B}_m^p = \left\{\tau \in \mathcal{T} \mid \forall t \in [4\text{s}, T_f], \min_{t' \in [t-4\text{s}, t]} \|q_m(t') - \hat{q}_m(t')\|_2 < 0.3\text{rad}\right\}. \tag{6.24}$$

A healthy human being should be able to withstand up to 5-step capturable disturbances but nothing stronger than it, so we set $\Delta T = 4\text{s}$. The maximum acceptable deviation is quite slack. This is to allow a bit of hysteresis when standing at rest after reactive stepping. It is both inefficient and unnatural to do one additional small step

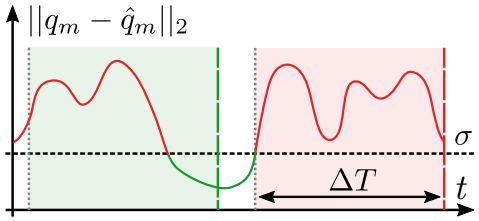


Figure 6.3: Probabilistic constraint on motor tracking error

to go back to the nominal resting pose if the final configuration is not too distant. It is necessary for walking on uneven ground without triggering early termination.

Mimicking the nominal trajectories of the feet accurately is what matters most. First, it plays a significant role in the feeling of natural walking. Secondly, they are designed to reduce the impact on the reality gap – mainly due to the mechanical deformation in particular but not only – and the robot is likely to fall in practice if already not lifting the feet properly in simulation. Thus, we also enforce probabilistic space-time bounds for the feet even if partially redundant. It is not as straightforward as for the joints. The constrained feature must not be subject to drifting, otherwise this effect will interfere and finally dominate, leading to systematic failure. The plain absolute pose of the feet in world frame ${}^{w}X_{r,l}$ is obviously not suitable. Similarly, the poses in base frame ${}^{b}X_{r,l}$ is misguided: poorly mimicking the orientation of the torso would directly impact the reproduction error for both feet, which is not motivated by any physical reasoning. For example, when we turn our torso to look around while walking, our feet keep following the same trajectory. It appears that the reference frame must be independent of the base to avoid any kind of harmful coupling. Picking one of the feet as a reference ${}^{r}X_{l}$ is better but still not satisfactory. It puts more pressure on the reference foot because its orientation affects the relative position in contrast to the other one. This asymmetric is not physically motived either.

One option is to express the pose of the feet in a reference frame that involves them equally. It is referred to as the motion frame and denoted ${}^{w}X_{r+l}$ in the following. Intuitively, the average pose is a good candidate, but it is not well-posed. Several definitions exist, especially for the orientation (Markley et al., 2007). One of them consists in finding the rotation that minimizes the mean square error of the distance with all the others. For any legged robot, it gives

$$\underset{R \in SO(3)}{\arg\min} \sum_{i=1}^{n} d({}^{w}R_i, R)^2, \tag{6.25}$$

where n is the number of feet, ${}^{w}R_i$ is the rotation of the i-th foot in the world frame, and d is a distance metric to be determined. Once again, the distance metric is not unique (Huynh, 2009). The Frobenius norm of the residual rotation is commonly used, i.e. $d({}^{w}R_i, {}^{w}R_j) = \|I_3 - {}^{w}R_i\,{}^{w}R_j^T\|_F = 2\sqrt{1-\cos(\theta_{i\text{-}j})} = 2\sqrt{2}|\sin(\theta_{i\text{-}j}/2)|$ where $\theta_{i\text{-}j}$ is the angle of the axis-angle representation of the residual rotation ${}^{w}R_i\,{}^{w}R_j^T = {}^{w}R_{i\text{-}j} = \exp((\theta_{i\text{-}j}u_{i\text{-}j})\times_3)$. This metric is boundedly equivalent to the angle $\theta_{i\text{-}j} \geq 0$,

so they should be interchangeable. The result would differ whether one or the other is optimized, but very slightly since the angle itself is a very good approximation of the Frobenius norm (less than 1% error for angles up to 15 degrees). For this particular metric, the quaternion of the solution to problem (6.25) is the Eigenvector associated with the largest Eigenvalue of the matrix Q defined as

$$Q = \sum_{i=1}^{n} q_i q_i^T, \tag{6.26}$$

where q_i is the quaternion for the i-th foot. For humanoids, it admits a closed form,

$$q_{r+l} = \sqrt{\frac{1}{2(1 + |q_r^T q_l|)}} \left(q_r + \text{sign}(q_r^T q_l) q_l\right). \tag{6.27}$$

On its side, the reference position is defined as the barycenter of the positions of the feet, $p_{r+l} = (p_r + p_l)/2$. Alternatively, it could also be kept as the origin of the world if one considers only relative positions. Typically, the pairwise relative positions between the feet for legged robots. No information is lost compared to using the barycenter as a reference, and it does not give prominence to any of the feet. For humanoid robots, both formulations are strictly equivalent. Nevertheless, the barycenter is slightly easier to work with as the features can be interpreted as actual frame poses and not just mathematical artifacts. Plus, it gets increasingly robust as the number of feet increases in contrast to the pairwise relative positions.

Extracting relevant transforms is half of the problem, computing the deviation between their actual and nominal values is the other one. Being able to set bounds for each component individually is critical for tightly adjusting the maximum acceptable deviation on each axis. It follows that the deviation must be expressed in the motion frame and the coordinate system easy to interpret. This choice is unrelated to the distance metric involved in the estimation of the motion frame.

As a reminder, any element of the Lie Group of transforms $SE(3)$ can be mapped to an element of its Lie Algebra $\mathfrak{se}(3) \subset \mathbb{R}^3 \times \mathbb{R}^3$ at identity – which identifies to its tangent space – through the pseudo-inverse of the exponential map for transforms (Blanco-Claraco, 2021). Basically, the angular part is the product of the axis by the angle of the relative rotation between the feet. For small rotations, it is approximately the same as the Euler angles (Roll, Pitch, Yaw). It should always be the case since it is the whole point of bounding the deviation through early termination conditions. However, it is much harder to interpret the linear part. It mixes the translation with the rotation, and it does not even preserve the norm of the translation. Hence, this coordinate system is suitable as it is difficult to make sense of all components separately. Still, this coordinate system would be satisfactory if the objective was to cancel the deviation completely without paying attention to intermediary states, e.g. performing a gradient descent on its L^2-norm. As a topological space, $SE(3)$ is homeomorphic to $\mathbb{R}^3 \times SO(3)$, and the coordinate system for the translation and rotation can be chosen independently. The translation is kept as is since it is already a

vector. Then, the pseudo-inverse of the exponential map for the Lie group $SO(3)$ can be applied to the rotation, which would provide the same axis-angle vector as before. Even though it is perfectly fine, we suggest using the Euler angles for convenience. It is easier to interpret for most people and physically meaningful.

Accommodating uneven ground requires allowing large deviations in rotation for the roll and pitch angles. On the contrary, it is helpful to be restrictive for the z-axis for the translation to enforce lifting the feet properly so as temper the reality gap without reward engineering. It is less straightforward for the translation along x- and y-axes. Tolerating some deviation in translation enables adaptation of the theoretical motions to better fit the actual needs and avoid unnecessary repositioning steps as discussed previously. However, being restrictive is helpful to mitigate the drifting phenomenon. Notably, the average velocity of the gait is determined by the step length and thus the deviation in translation along x-axis. The question of long-term drifting is already tackled by probabilistic space-bounds in equation (6.24), but cumulative bounds are intended to be triggered way sooner, thereby discarding bad local minima at an early stage during the optimization process. Besides, if not restrictive enough for the translation along y-axis, the robot would spread the legs to improve stability, which is unconformable for the patient, unnatural, and eventually dangerous for people nearby. More generally, it is recommended to be restrictive if slack bounds are not strictly necessary for solving the task.

The difference between two transforms is characterized by the coordinate vector of a residual. Classically, the residual is the transform moving from one frame to the other, i.e. $X_i X_j^{-1}$ or $X_j X_i^{-1}$. It is still expressed in the motion frame, as opposed to the relative transform between frames ${}^{j}X_i = X_j^{-1} X_i$. Yet, it is not the quantity we are looking for because the residual translation depends on the orientation of both transforms while it should be invariable. It is not a problem if the goal is to cancel the deviation, and this formulation is found in most inverse kinematics algorithms for pick-and-place. But here, we must consider the residual translation and rotation singly, $p_{i\text{-}j} = p_i - p_j, R_{i-j} = R_i R_j^T$. This decomposition is occasionally referred to as *double geodesic* difference. The residual translation and rotation may be aggregated in a transform for convenience, but it is more of a computational artifact since its inverse does not make much sense. Moreover, it is not useful for computing the associated coordinate vector as the translation and rotation are handled independently anyway. It is worth noting that it is also possible to compute the difference between the two coordinate vectors associated with the transforms. It is not well-defined, but it is a good approximation if the two transforms are sufficiently close.

Despite our best efforts, the deviation in translation is biased by the deviation in rotation. Indeed, it is implicitly assumed for computing the difference that the two transforms are expressed in the same reference frame ${}^{o}X_i, {}^{o}X_j$. In our case, it means that the estimation of the motion frame is consistent between the actual and nominal trajectories. It is not true unless the orientations of the feet match exactly in both cases since the motion frame is derived from them, ignoring the base as already

mentioned. The estimation error is bounded according to the deviation in rotation:

$$\begin{aligned}
\| {}^{o_1}p_1 - {}^{o_2}p_2\|_2 &= \| {}^{o_1}p_1 - {}^{o_2}R_{o_1}({}^{o_1}p_1 + {}^{o_1}p_{1\text{-}2})\|_2 \\
&\leq \|(I_3 - {}^{o_2}R_{o_1})\,{}^{o_1}p_1\|_2 + \| {}^{o_2}R_{o_1}\,{}^{o_1}p_{1\text{-}2}\|_2 \\
&\leq \lambda_{\max}(I_3 - {}^{o_2}R_{o_1})\|p_1\|_2 + \|p_{1\text{-}2}\|_2 \\
&\leq \|I_3 - {}^{w}R_{o_2}^T\,{}^{w}R_{o_1}\|_F\|p_1\|_2 + \|p_{1\text{-}2}\|_2 \\
&\leq \|I_3 - {}^{w}R_{o_2}\,{}^{w}R_{o_1}^T\|_F\|p_1\|_2 + \|p_{1\text{-}2}\|_2 \\
&\leq 2\sqrt{2}\,|\sin\left(\theta_{o_1\text{-}o_2}/2\right)|\,\|p_1\|_2 + \|p_{1\text{-}2}\|_2
\end{aligned}$$

It shows that it is a reasonable approximation if the deviation in rotation is small,

$$0 \leq \frac{\| {}^{o_1}p_1 - {}^{o_2}p_2\|_2}{\|p_{1\text{-}2}\|_2} - 1 \leq 2\sqrt{2}\,|\sin\left(\theta_{o_1\text{-}o_2}/2\right)|\,, \tag{6.28}$$

e.g. 10 degrees induces at most 25% estimation error.

The pairwise distances between the feet have the advantage of not being affected by the estimation error of the motion frame since the Euclidean norm is invariant by rotation. Similarly, the angle from the axis-angle representation is independent of the reference frame contrary to the Euler angles. Therefore, defining a reference frame can be avoided altogether by focusing solely on the deviation of the pairwise distances and angles. Information may be partially lost according to the number of feet. This question is related to multidimensional scaling (Borg & Groenen, 2005): estimating the position of points from the matrix of pairwise distances between them. It is under-determined if the number of points is not sufficient. More points are required as the dimension of the embedding space increases. If the measure of the distances is noisy, then adding more points is still relevant even if fully determined as it reduces the variance of the estimation of the positions. For instance, only two configurations exist when 4 points are available in a 3-dimensional space. It entails that almost no information is lost for a quadrupedal robot and none beyond that. It is not true for humanoid robots, but it would be adequate for tasks where penalizing differently the deviation for each coordinate is not essential.

We enforce the following probabilistic space-time bounds for the feet,

$$\begin{aligned}
\mathcal{B}_{r,l}^p = \{&\forall t \in [4, T_f], \min_{t' \in [t-4,t]} \| {}^{r+l}p_{r\text{-}l}(t') - {}^{r+l}\hat{p}_{r\text{-}l}(t')\|_2 < 10\text{cm}\} \\
\cap\, \{&\forall t \in [4, T_f], \min_{t' \in [t-4,t]} \|\text{rpy}({}^{r+l}R_{r\text{-}l}(t')\,{}^{r+l}\hat{R}_{r\text{-}l}^T(t'))\|_\infty < 15\text{deg}\},
\end{aligned} \tag{6.29}$$

where rpy is the function mapping rotation matrices to the corresponding Euler angles, properly restricted to ranges ensuring uniqueness except for a few pathological configurations (cf. section 2.1.1).

All these space-time bounds ($\mathcal{B}_b^i, \mathcal{B}_o^c, \mathcal{B}_m^p, \mathcal{B}_{r,l}^p$) are generic enough to be used as is for any locomotion task with or without external disturbances and uneven ground. Some of them have been specialized for humanoid robots when it was clearer but are applicable to any legged robot. They are designed to be robust to the hyperparameters, so that the exact same values are used for our two training scenarios.

6.3.2 Scalable Multi-Tasking through Lower-Bound Maximization

The second training scenario involves learning a policy capable of performing equally well multiple distinctive tasks by imitation. As before, the task encapsulates both gait and patient features in order to learn a generic policy for all patients or even adapt to changes, e.g. the patient gain weight.

Siekmann et al. (2021a) present a simple method to learn a continuum of gaits and to smoothly transition between all of them by means of a generic policy. It consists in changing the desired gait features multiple times during a single episode. The set of active reward components differs according to the desired type of gait (hopping vs. walking vs. standing ...etc), and the desired gait features are forwarded as input to the policy. This way, it does not break the assumption of stationary MDP. They validate this approach experimentally on the bipedal robot Cassie. Specifically, the gait features are the forward velocity of the pelvis, the frequency of the steps, and the phase ratio between both legs.

The previous approach is only effective if learning to perform each of the desired gaits has similarly difficult for the agent. Depending on the robot of interest, some motions may be significantly more difficult to perform than others. For instance, flat foot walking on Atalante is easier than foot rolling, itself much easier than doing a front flip. If so, then naively learning all motions at once without special care as Siekmann et al. would be unsuccessful: the agent would get stuck in a bad local minimum where it performs a subset of the tasks extremely well while others are completely left out. It comes from the fact that the agent is maximizing the return in expectation over the distribution of tasks. For a discrete set of motions, a naive approach would be to learn an expert policy for each motion individually and unify them as a single policy afterward. It is only applicable for very few motions because it scales linearly with their number and learning any of them is already costly. Moreover, it does not generalize to the continuous domain while most gait features vary continuously.

Won et al. (2020) introduce a more scalable and versatile method that combines preconditioning and warm-starting. The core idea is to flatten virtually the difficulty across motions, thereby enabling learning them all at once. To this end, the procedure is two-stage. First, they partition the motions into several coherent groups in which variation of high-level gait features is supposed to have no discernible impact on difficulty. This is done using an unsupervised clustering algorithm for a fixed number of groups rather than leveraging prior knowledge from an expert. One expert policy is trained for each group of motions. Then, they are fused together by a gating network that acts as an adaptive weighted sum. It outputs normalized weights based on the current observation, and the action is the weighted sum of the predictions associated with all the expert policies. In practice, the network is a simple *Feedforward Neural Network* (FNN) with Softmax activation function as the final layer, and the observation gathers the actual state and some desired high-level gait features. The tasks are evenly distributed for the policy to perform equally well for all of them. The computational cost scales linearly with the number of groups. It is already a great improvement upon the naive approach and generalizes to the

continuous domain. The agent may fail to learn some motions if their clustering is too coarse relative to their actual diversity because of limited resources.

Impressive results were achieved in simulation for one hundred humanoid motions on flat ground and without external disturbances. Nevertheless, their approach is not well-suited for learning to transition between motions at any point in time by changing the high-level gait features, which is one of our goals. It is only allowed within the same group in the first stage, and it has to rely on a linear combination of expert policies in the second stage. Each expert policy is tailored for one specific skill and has only observed the nominal states among their own group, not to mention that optimal transitions may have nothing to do with any of the nominal motions. Although the parameters of the experts are still trainable at the second stage, it is unlikely to be sufficient. RL algorithms are finding local optima so splitting the process into two stages impedes performance. Besides, there is no information aggregation during the training of the expert policies. This segmentation prevents sharing of basic knowledge that could be common to all of them, in particular the feature extraction and the underlying system dynamics. Jointly learning all the expert policies from the start may speed up convergence and improve the robustness to corrupted input or model uncertainties. Still, the overall architecture restricted to linear combinations of policies has less expressiveness than a generic feedforward policy.

Being able to train from scratch a single feedforward policy to perform multiple tasks equally well regardless of their difficulty would overcome these limitations altogether. Let us consider a task space $\mathcal{T}$ that may be discrete or continuous. The task τ is a random variable that is sampled at the beginning of each episode and stays the same until termination. In general, the reward function R^τ, the transition dynamics P^τ and subsequently the discounted stationary state distribution ρ_π^τ all depend on the current task τ. We assume that the task distribution is independent of the conditional expectation of the return J_π^τ given the task τ, such that the *law of total expectation* can be applied. It follows that the total expectation of the return for a given policy J_π can be written as the average of the conditional expectation J_π^τ weighted by the probability $\mathbb{P}(\tau)$,

$$J_\pi = \mathop{\mathbb{E}}_{\tau\sim\mathbb{P}(\tau)}\left[J_\pi^\tau\right] = \mathop{\mathbb{E}}_{\tau\sim\mathbb{P}(\tau)}\Big[\mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi^\tau \\ a\sim\pi}}\Big[\mathop{\mathbb{E}}_{s'\sim P_\tau}\left[R^\tau(s',a,s)\right]\Big]\Big]. \tag{6.30}$$

Assuming infinite expressiveness of the policy π, maximizing the objective function J_π is equivalent to maximizing the expected return for each task individually irrespective of their probability. It is the best performance that could be possibly achieved. In practice, it is a completely different story. First, RL algorithms are not finding the global maximum but rather a local one by iteratively optimizing the training parameters. Updating the parameters to improve the performance for a given task at one point would actually impact them all until convergence to some extent that is hard to anticipate. Poor performance due to premature convergence is a prominent manifestation of this phenomenon. Next, the expressive power is necessarily limited. Improving the performance for one task would affect similar ones, thereby the optimization problem for each task is no longer decoupled but compete

with each other. The agent would give precedence to the tasks leading to the largest improvement weighted by their probability when updating the training parameters, eventually marginally impeding the others. Therefore, maximizing the expected return is not equivalent to performing all tasks equally well and may lead to large discrepancies between them. How to choose the probability $\mathbb{P}(\tau)$ for each task in order to learn a policy that performs equally well for all of them is not straightforward.

We propose instead to maximize explicitly the lower-bound of the expected return for all task J_π^{LB}, i.e. only the task for which the agent is the least successful,

$$J_\pi^{\text{LB}} = \min_{\tau \in \mathcal{T}} J_\pi^\tau. \tag{6.31}$$

Finding the optimal policy under this formulation is a so-called *minimax* problem: maximizing the minimum of the expected return with respect to the policy over the task space. It can be interpreted as pushing away as far as possible a Pareto front. This problem is NP-hard, so trying to solve it as such is extremely challenging whatever the optimization algorithm. First, improving the worst case may degrade the others without affecting the total cost. Classic SGD is not effective for solving such a problem because of the local sparsity of the gradient. It would significantly slow the convergence and may even damage the performance ultimately. Next, computing the minimum on a continuous domain is an optimization problem by itself that would be too costly to solve at every SGD iteration. Anyway, changing the task being optimized at every iteration does not give enough time to the agent to learn any of them efficiently. For learning to solve a given task without forgetting it, it is crucial to focus on this task for a while and still have a chance to draw it later on.

To address these concerns, we only consider a finite collection of tasks $\{\tau_i\}_{i=0}^m$ spanning the whole space, and we relax the minimization using a smooth approximation. A common choice is the real-valued function 'LogSumExp' aka LSE_β with temperature parameter $\beta > 0$,

$$\underset{1 \le i \le m}{\text{LSE}_\beta}(w_i) = -\frac{1}{\beta} \log\left(\textstyle\sum_{1 \le i \le m} \exp(-\beta w_i)\right), \tag{6.32}$$

where w_i is a score associated with task τ_i, here, the conditional expectation of return J_π^i. The optimization of the relaxed objective function J_π^{LSE} in place of the true one is well-motivated mathematically. First, 'LogSumExp' matches the true minimum when the temperature goes to infinity,

$$\min_{1 \le i \le m} w_i - \frac{\log(m)}{\beta} \le \underset{1 \le i \le m}{\text{LSE}_\beta}(w_i) < \min_{1 \le i \le m} w_i. \tag{6.33}$$

This parameter is used to adjust the trade-off between accuracy and sparsity. Then, it was demonstrated in section 5.2.2 that the minimum for a uniformly sampled collection approximates the true minimum over a continuous domain under some regularity assumption about the score function f. As for the temperature, this approximation matches the true minimum when the number of samples goes to infinity.

The objective function J_π^{LSE} involves estimating the conditional expectation of the return J_π^τ for each task separately. It does not fit with the classical framework, and applying any existing RL algorithm with it would require a custom implementation of its gradient computation. Such an inconvenience can be avoided by choosing the task distribution appropriately in the original objective function J_π:

$$\begin{aligned}
\nabla_\pi J_\pi^{\text{LSE}} &= \nabla_\pi \underset{1\leq i\leq m}{\text{LSE}_\beta}(J_\pi^i) \\
&= \nabla_\pi \left(-\frac{1}{\beta}\log\left(\textstyle\sum_{1\leq i\leq m}\exp(-\beta J_\pi^i)\right)\right) \\
&= \frac{\sum_{1\leq i\leq m}\exp\left(-\beta J_\pi^i\right)\nabla_\pi J_\pi^i}{\sum_{1\leq i\leq m}\exp\left(-\beta J_\pi^i\right)} \\
&= \mathbb{E}_{i\sim S(J_\pi^0,\ldots,J_\pi^m)}\left[\nabla_\pi J_\pi^i\right] \\
&= \nabla_\pi \hat{\mathbb{E}}\left[J_\tau(\pi)\right] \\
&= \hat{\nabla}_\pi J_\pi
\end{aligned}$$

where S is the Boltzmann distribution $\mathbb{P}(i) = \exp(-\beta w_i)/\sum_{1\leq i\leq m}\exp(-\beta w_i)$, $\hat{\mathbb{E}}$ is the empirical mean for the observed distribution of tasks on a given training batch, and $\hat{\nabla}_\pi$ is the empirical gradient estimate. Taking the gradient out of the expectation is possible because any potential dependency between the tasks and the policy is implicitly ignored by the gradient estimate.

Actually, maximizing the lower bound of the expected return is not strictly equivalent to optimizing the original objective function $J(\pi)$ for an adaptive task distribution updated at every iteration. Indeed, the last equality is only valid if the law of total expectation can be applied. It requires the task distribution to be independent of the policy, which is obviously wrong if the task distribution is updated at every iteration according to the performance of the policy for each task. Besides, estimating reliably the task for which the agent is the least successful requires observing many of them. It scales already poorly with the dimensionality for the uniform distribution, and it is much worse for the Boltzmann distribution. The number of tasks in a single training batch would hardly be sufficient since its value is sampled once per episode. Moreover, the task distribution affects the observed transition probability and reward function, and updating it breaks the fundamental markovian assumption. We suggest computing the task distribution based on the estimated performance over several training iterations at once. It effectively decouples the task distribution from the current policy if the number of iterations is large enough, but it must not be too large to avoid overfitting some tasks. This approach is closely related to the so-called importance sampling method first introduced by Park et al. (2019) and later applied by Ma et al. (2021). For convenience and to spare memory, it can be implemented as a moving average filter when updating at every iteration the score w_i. This trick is essentially the same as the delayed target updated in Deep Q-learning (cf. appendix E.4), but the probability is smoothed out rather than being updated periodically. It has the additional advantage to improve tremendously the reliability

of the estimated performance for the current policy over the whole task space by increasing the number of samples available. Samples from previous iterations are going to be off-policy. This phenomenon is neglected, and thereby the learning rate is limited by the window size of the filter.

The expected return is not the best metric to assess the performance of the agent for a given task as it takes into account many more aspects such as the comfort of the patient. Instead, its performance should be regarded in terms of the ability to satisfy the hard constraints of the problem. Thus, we rely on the mean relative duration of the episodes before termination to compute the score w_i. This metric is even more sensitive than the expected return: the robot may fall after a few seconds or walk indefinitely by slightly changing the parameters of the policy or just resampling the initial state and the external pushes. This variability is not an issue in practice thanks to the moving average filter.

6.3.3 Task Transitioning via Cross-Initialization

Robust imitation of nominal motions has been addressed in the previous section. It is already a valuable achievement on its own, but it does not answer the question of how to reach these limit cycles in the first place. Resorting on some advanced classic controller to handle specifically the transient dynamics at runtime is not going to play well with the policy because it was not trained for it. We prefer to avoid any additional complexity once embedded in the real device by putting more effort into the training process in simulation. Considering the resting pose as a motion like any other, what is really needed is learning an end-to-end control policy that is capable of smoothly transitioning between nominal motions at any point in time.

It is a logical follow-up to the work that has been done in the previous chapter. Relying on trajectory planning to generate optimal transitions is not suitable either. Considering only two motions, it is already unattainable because time is continuous. It is not ideal to force the transition to happen at one particular moment because it impedes the reactivity of the system and shatters symbiotic coordination: it feels weird to wait for the system to go back to a given configuration in the limit cycle before transitioning. Even so, the total number of transitions scales quadratically with the number of nominal motions m, which is already untractable to generate in most cases. Notably, going from task i to j or vice versa is not strictly equivalent since the dynamics stability is time reversal but not the appreciation of the patient. One conceivable approach consists in using the resting pose as a universal midpoint for transitioning from one nominal motion to any other. This greatly reduces the number of transitions to generate, namely one to start and one to stop for all nominal motions. However, such transitions would feel unnatural for the patient. This may be acceptable when switching between very different motions, but it is clearly not appropriate when the gait features are changing continuously.

We suggest discovering stable and robust transitions using RL without generating the optimal ones. To this end, the initial state of the robot is uniformly sampled among all the states that belong to the nominal motions put together regardless of the

actual task to perform for the current episode. The agent will learn to converge back to the limit cycle as fast as possible while keeping balance all along to avoid triggering early termination because of the generalized space-time bounds. This method is simple to implement and more powerful that the autoregressive model presented in section 5.5.2. The latter extrapolates the transition of maximum likelihood from a prior distribution of optimal nominal motions. Such a transition has no guarantee to be stable if the acceptable safety threshold is exceeded, which is very likely unless the task features are updated by small increments. Moreover, the attractiveness of the limit cycle was fixed and tuned manually, requiring being conservative.

Setting up a time limit instead of running indefinitely until failure is imperative for sample diversity (cf. section 3.2.2). Here, it ensures that each training batch comprises many independent episodes and corresponding initialization steps. Thus, switching nominal motion during the episodes as Siekmann et al. (2021a) would add extra complexity without any advantage.

6.4 Improving Convergence, Predictability and Safety

Imitation of nominal motions through generalized space-time bounds acts as a guide that contributes to ensuring the safety of the user and people nearby while enhancing rehabilitation. It is different from tracking the nominal motion the whole time because allowing a large deviation locally is necessary to recover balance and adapt to uneven ground. The side effect of this extra freedom is a lack of predictability and safety when an unexpected event occurs. Moreover, the hardware has some physical limitations that cannot be ignored, for instance, the position, velocity, and acceleration bounds of the motors. We present several mechanisms to partially address these concerns in the following.

6.4.1 Reward Engineering

Providing an informative reward is optional since the generalized space-time bounds are already doing most of the job. The agent would learn a sensible policy for a sparse reward returning $+1$ systematically until termination. Nevertheless, doing some reward engineering is still beneficial. First, it speeds up convergence in most cases by discouraging poor local minimum earlier during training by providing insight into how to perform the task, i.e. keeping balance. Next, it can complement hard constraints by encouraging being closer to the nominal motions than strictly required if it does not undermine the stability. Finally, it enables discriminating between several suitable behaviors that solve the task. Specifically, we want to trigger reactive steps only as a last resort as it is unpleasant. Other strategies should be employed first, motor compliance being the preferred option. We aim to be generic, so they can be used for both training scenarios and in conjunction with any nominal motion.

The total reward is a real-valued function aggregating individual reward components. These reward components are derived from errors in heterogenous metric spaces. Thus, the preliminary step consists in scaling them uniformly through the

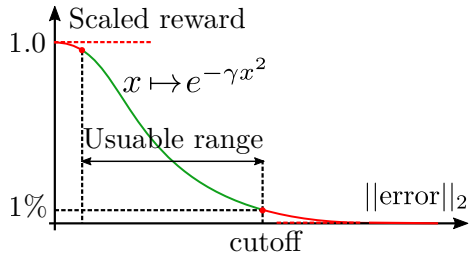


Figure 6.4: RBF kernel for scaling reward components.

widespread *Radial Basis Function* (RBF) (see figure 6.4),

$$K(x, \hat{x}) = \exp\left(-\kappa d(x, \hat{x})^2\right) \in [0, 1], \tag{6.34}$$

where x is a vector-valued feature, d is the distance between its current and target values, and κ is a cutoff parameter. It defines the operating range and must be tuned carefully for each reward component. This procedure is tedious but must be done only once for each environment in most cases. If too small, the margin for improvement is small. Conversely, if too large, then the agent has no idea what to do. In both cases, the agent barely gets any incentive to improve itself because the gradient vanishes. This issue pertains to squeezing the positive real number into the unit interval and is not specific to the exponential transform.

At least two different approaches are found in the literature to mix multiple reward components: additive and multiplicative (Lee et al., 2019). For the multiplication, the agent gets a reward if and only if all components involved are rewarded. It makes sense for components that are mutually dependent and tend to reinforce each other. For instance, tracking the position of the feet and the relative joint angles. On the contrary, the addition is more appropriate if it embodies goals that have nothing in common or even conflicting, so they can be pursued concurrently while searching for a compromise. In principle, the addition could be used in both cases, but the multiplication has extra benefits. The addition is fine with having only a few components being rewarded, which is often undesirable. In contrast, the multiplication urges to balance the total reward among all components. This lack of competition alleviates the need to weigh the components, which is strongly related to the problem at hand. As a result, the multiplication has fewer parameters, and a large variety of problems for a given environment can be addressed without dedicated hyperparameter tuning.

Despite all this, we chose to stick with the addition in this work because it is more usual, and we consider only non-conflicting reward components. The total reward is a normalized weighted sum of individual reward components,

$$r_t = \textstyle\sum_i w_i \exp(-\kappa_i c_i^2) \in [0, 1], \tag{6.35}$$

where $c_i \geq 0$ is a cost and w_i its weight. The weights are normalized, i.e. $\sum_i w_i = 1$. Their values determine the priority level of each reward component. The current time is omitted in the following for simplicity if there is no ambiguity.

Tracking

We define a set of independent reward components all promoting tracking of the nominal motions, which is the resting pose for standing push recovery. Even if one is poorly optimized, it is still possible to improve the others as they are unrelated. The goal is essentially the same as the generalized space-time bounds, so they share most of the features and distance metrics.

Odometry. In a real scenario, the environment is likely to be cluttered. For a recovery strategy to be of any use, it is important to make sure the robot stays within a limited action radius. As mentioned before, the cost impacts the deviation of the velocity because the position in the world plane is allowed to drift.

$$\|\bar{v}_b - \hat{\bar{v}}_b\|_2, \tag{6.36}$$

where $\bar{v}$ denotes the average velocity since the previous timestep. The average velocity is obtained by backward differentiation of the transform, i.e. $(p_b(t) \ominus p_b(t - \Delta t))/\Delta t$ where $\ominus$ is the geodesic difference in $SE(3)$. It reflects what truly happened since the agent took action for the last time, and the corresponding reward component does not break the markovian assumption. It is also much smoother than the true instantaneous velocity, which would be misleading if the dynamics is jerky. Thus, this reward should be easier to optimize for the agent. However, the average velocity cannot be inferred from the current observation.

Motors. The agent should track the nominal already stable. Besides, it must not take any action once falling is inevitable. It is essential to prevent dangerous motions in unexpected situations. Rewarding the agent for tracking the nominal at the joint level fulfills this goal because this metric is completely agnostic to the actual stability and can be maximized despite being falling,

$$\|q_m - \hat{q}_m\|_2. \tag{6.37}$$

End-effectors. The feet should be flat on the ground and at a specific distance from each other. Without promoting this via a reward component, the agent would learn to spread the legs as much as possible to improve the robustness to external forces, which is both unnatural and unpleasant. Considering the norm of the residual transform as a whole to compute the error is not advisable. Indeed, a large error for a single coordinate would be enough for the whole gradient to vanish because of the rescaling by the RBF kernel. The translational and rotational parts of the residual transform are independent, so it is more appropriate to penalize them separately,

$$\| {}^{r+l}p_{r\text{-}l} - {}^{r+l}\hat{p}_{r\text{-}l}\|_2, \tag{6.38}$$

$$|\text{angle}({}^{r+l}R_{r\text{-}l} \, {}^{r+l}\hat{R}^T_{r\text{-}l})|. \tag{6.39}$$

Stability

We define reward components that aim to give a sense of balance and encourage the agent to deviate from the nominal for the sake of stability. They are insightful

regardless of the state of the system and the magnitude of the disturbance. Notably, they are well-defined even if the robot is flying.

Dynamic stability. The *Zero-tilting Moment Point* (ZMP) should be kept inside the support polygon for dynamic stability,

$$\| {}^{w}p_{zmp} - {}^{w}p_{sp} \|_2, \tag{6.40}$$

where sp is the center of the projected support polygon instead of the true one. It anticipates future impact with the ground and is agnostic to the contact state. The rationale behind it has been presented in section 2.2.2.

Foot placement. We want to move the feet as soon as the Capture Point goes outside the support polygon. We encourage moving the pelvis toward the Capture Point to get it back under the feet,

$$\| {}^{o}p_{cp} - {}^{o}\hat{p}_{cp} \|_2, \tag{6.41}$$

where ${}^{o}p_{cp}$ is the relative position of the Capture Point in the odometry frame.

Safety and Comfort

Safety needs to be guaranteed during the operation of a bipedal robot. Comfort is also important for a medical exoskeleton to enhance rehabilitation.

Balanced contact forces. Distributing the weight evenly on both feet is key in natural standing,

$$| {}^{w}F_r^z \hat{\delta}_r + {}^{w}F_l^z \hat{\delta}_l - mg |, \tag{6.42}$$

where $\hat{\delta}_{r,l} \in \{0, 1\}^2$ are the right and left nominal contact states, and ${}^{w}F_{r,l}^z$ are the right and left total vertical forces in world frame.

Ground friction. Reducing the tangential contact forces limits internal constraints in the mechanical structure that could lead to overheating and slipping. Moreover, exploiting too much friction may lead to unrealistic behaviors,

$$\max_{i \in \mathcal{C}_r \cup \mathcal{C}_l} \frac{\| {}^{i}F_i^{x,y} \|_2}{{}^{i}F_i^z}, \tag{6.43}$$

where $\mathcal{C}_{r,l}$ are the right and left active sets of contact points, and ${}^{i}F_i^{x,y}, {}^{i}F_i^z$ are the tangential and vertical forces acting on the i-th contact point in local ground frame. It is upper-bounded by the friction coefficient with the ground mu.

Pelvis momentum. Fast pelvis motions are unpleasant. Besides, reducing the angular momentum helps to keep balance,

$$\left\| [\dot{\psi}_b, \dot{\theta}_b] - [\dot{\hat{\psi}}_b, \dot{\hat{\theta}}_b] \right\|_2, \tag{6.44}$$

where $\dot{\hat{\psi}}_b, \dot{\hat{\theta}}_b$ denote the average velocity of the roll and pitch of the pelvis since the previous timestep respectively.

6.4.2 Termination Conditions

We present here carefully designed instantaneous space-time bounds that ease transfer from simulation to reality and forbid unsafe behaviors. Incidentally, it speeds up convergence by reducing the search space. The numerical values of the hyperparameters are specific to the environment. They were obtained via a qualitative study in simulation unless stated otherwise.

Unrecoverable State

Stopping the simulation if the state is surely unrecoverable avoids gathering useless transition steps and thereby speeds up the learning process.

Pelvis Height. The height of the pelvis is a fast and conservative heuristic,

$$\mathcal{B}_z^i = \{\forall t \in [0, T_f], 30\text{cm} < z_b(t)\}. \tag{6.45}$$

Power Consumption. We limit the power consumption to fit the hardware capability and prevent motor overheating,

$$\mathcal{B}_p^i = \{\forall t \in [0, T_f], \langle u_m(t), \dot{q}_m(t)\rangle < 3\text{kW}\}. \tag{6.46}$$

Safety and Comfort

We must prevent behaviors that may cause premature wear or damage the hardware if possible. Hurting the patient must be avoided at all costs, and ideally, the motion must be gentle enough to not frighten the user and people nearby.

Pelvis Orientation. We restrict the orientation of the pelvis. For example, the upper body is never leaning back. It yields,

$$\begin{aligned}\mathcal{B}_b^i = &\{\forall t \in [0, T_f], |\psi_b(t)| < 0.4\text{rad}\} \\ &\cap \{\forall t \in [0, T_f], -0.25\text{rad} < \theta_b(t) < 0\}.\end{aligned} \tag{6.47}$$

Joint Bounds Collisions. Hitting the mechanical stops q_m^-, q_m^+ is inconvenient, but forbidding it completely is undesirable. It constrains the problem too strictly and avoiding falling is the highest priority. Never hitting bounds requires safety margins that would dramatically impede performance. Still, the maximum velocity at impact must be restricted to prevent destructive damage or injuries due to the shockwave. An acceptable threshold has been estimated from real experiments. It is enforced separately for each motor i,

$$\begin{aligned}\mathcal{B}_i^i = &\{\forall t \in [0, T_f], q_i^- < q_i(t) < q_i^+\} \\ &\cup \{\forall t \in [0, T_f], |\dot{q}_i(t)| < 0.6\text{rad}\,\text{s}^{-1}\}.\end{aligned} \tag{6.48}$$

Foot Collisions. Foot collisions need to be forestalled,

$$\min_{p_r \in \mathcal{CH}_r, p_l \in \mathcal{CH}_l} \|p_r - p_l\|_2 > 2\text{cm}, \tag{6.49}$$

where $\mathcal{CH}_{r,l}$ are the convex hulls of the right and left footprints respectively. It has no closed-form expression but can be computed efficiently (Kaown & Liu, 2009), e.g. using *Gilbert-Johnson-Keerthi* (GJK) algorithm.

Impact Forces. Like hitting the mechanical stops, violent impacts on the ground cannot be avoided entirely but must be limited as much as possible,

$$\mathcal{B}_f^i = \left\{\forall t \in [0.5\text{s}, T_f], \max_{i \in \mathcal{C}_r \cup \mathcal{C}_l} \frac{\| {}^iF_i \|_2}{mg} < 4.5\right\}, \tag{6.50}$$

where ${}^iF_i$ is the linear force acting on the i-th active contact point in the local frame.

Transferability

Some behaviors may be very effective in simulation but poorly executed on the robot due to the reality gap. We mitigate this phenomenon by prohibiting such behaviors.

Contact Persistence. Jumping knowingly with the robot from the resting pose is a challenge. This is even worse when pushed since the feet often tilt and stick on the ground. Supposedly, this is caused by the dynamic model of the mechanical deformation being off. Anyway, the flying phase is disturbing for the user. Alternative strategies such as ankle control or stepping are harder to perform (also for humans!) but do not face these issues. Losing contact for a short instant is unavoidable after strong pushes. Thus, we relax the condition of strict contact by checking the distance,

$$\mathcal{B}_c^i = \left\{\forall t \in [0.5\text{s}, T_f], \min_{\substack{p_a \in \mathcal{CH}_r \cup \mathcal{CH}_l \\ p_w \in \mathcal{G}}} \|p_a - p_w\|_2 < 3\text{mm}\right\}, \tag{6.51}$$

where $\mathcal{G}$ denotes the geometry of the ground. The first 0.5s are skipped to prevent instant failure following randomized initialization.

6.4.3 Explicit constraints

The action gathers the target velocity of all the motors. Reducing the motor tracking error is important for predictability, safety and transfer to reality (cf. section 6.1.1). Typically, a large error would trigger a preemptive emergency stop on the real device as this situation can be dangerous. The lower-level PIDs have their own dynamics that are unknown. Therefore, the only reliable way to limit the tracking error is by bounding the maximum acceleration of the motors.

More generally, it is advisable to prohibit excessively fast motions. First, it should be unnecessary in most cases. Secondly, the behavior must be as slow as possible to mitigate the risk of injuries because of muscle spasticity and osteoporosis. Finally, it drives the hardware into a corner for which it is not certified: motors may overheat, mechanical parts may break, and the power supply must collapse.

We bound the target accelerations $\tilde{\ddot{q}}_m$. This way, it only involves the policy and not the actual dynamics, so it can be enforced analytically using exponential barrier functions as discussed in section 6.2.2. This approach is not always appropriate. For

instance, it cannot be used to prevent hitting the mechanical stops as the target position $\tilde{q}_m$ cannot be out of bounds in the first place. This phenomenon happens precisely because of the discrepancy between the target and current motor positions.

In practice, we consider the variation of the target velocities $\tilde{\dot{q}}_m$ over successive timesteps. The rationale is the same as for the odometry reward: it is the average acceleration since the last time the agent took action that matters. Anyway, the true target acceleration $\tilde{\ddot{q}}_m$ is unknown because it depends on the time derivative of the current observation and necessitates the model. Moreover, the actual target velocity $\tilde{\dot{q}}_m$ is replaced by the expectation of the policy for the corresponding state $\mathbb{E}[\pi(s_t)]$ to avoid penalizing exploration during training. The variation of target velocity cannot exceed 40% of the operational range,

$$c_m^i(s_t, a_t, s_{t+1}) = \frac{|\mathbb{E}[\pi(s_{t+1})] - \mathbb{E}[\pi(s_t)]|}{\dot{q}_m^+} - 0.4, \tag{6.52}$$

where $\dot{q}_m^+$ denotes the maximum velocity of the motors. The division and subtraction must be understood element-wise. The action is normalized, so it requires at least 5 steps (200ms) to reverse the direction of rotation at full speed. It corresponds to a maximum acceleration of about $20\mathrm{rad\,s^{-2}}$.

The slope of the exponential barrier function α is set to 10.0. This is slack on purpose as there is no hard constraint on the maximum acceleration. It is preferable to exceed the prescribed threshold from time to time if necessary to avoid falling.

6.4.4 Smoothness Conditioning

Jerky commands are not accurately simulated, hardly transfer to reality, and shorten the lifetime of the hardware. In the case of a medical exoskeleton, smoothness is even more critical as vibrations are annoying and can cause anxiety for its user. Ensuring smoothness during training mitigates these issues without impeding performance if tuned correctly. Beyond this, it leads to more predictable behaviors.

In RL, smoothness is commonly promoted by adding regularization as reward components, such as the total power consumption or the norm of the motor velocities (Ferigo et al., 2021; Yang et al., 2018). However, components in the reward function have no guarantee to be optimized as they are a minor contribution to the actual loss of learning algorithms. Injecting the regularization as extra terms in the loss function directly gives us control over how much it is enforced during the learning.

The policy is generally a dense neural network that has thousands or even millions of training parameters. It follows that the optimization problem is ill-posed. Global regularization approaches including smoothness conditioning are known for substantially reducing the variability of the solutions so that the final policy would be roughly the same between several runs after changing the random seed. This property is important as it reduces the number of experiments on the real platform that is required for tuning hyperparameters. Besides, it tends to slightly improve the convergence speed and discard pathological behaviors that would be considered

valid otherwise, e.g. oscillating periodically to trick the generalized space-time bounds instead of mimicking the nominal motion.

As suggested by (Mysore et al., 2021), we use both temporal and spatial regularization to promote smoothness for the learned state-to-action mappings of the neural network controller. The temporal and spatial regularization terms are given by,

$$L^T(\pi) = \|\mathbb{E}[\pi(s_{t+1})] - \mathbb{E}[\pi(s_t)]\|_1 \,, \tag{6.53}$$

$$L^S(\pi) = \|\mathbb{E}[\pi(s_t)] - \mathbb{E}[\pi(s'_t)]\|_2^2, \tag{6.54}$$

where $s'_t \sim \mathcal{N}(s_t, \sigma_S)$. As a reminder, the state is supposed to be normalized. They are added to the original surrogate loss function of the training algorithm L^R,

$$L(\pi) = L^R(\pi) + \lambda_T L^T(\pi) + \lambda_S L^S(\pi), \tag{6.55}$$

where λ_T, λ_S are hyperparameters than must be tuned to adjust the smoothness.

Mysore et al. choose σ_S based on expected measurement noise and/or tolerance. However, it limits its effectiveness to robustness concerns. Its true power is unveiled when smoothness is further used to enforce regularity and cluster the behavior of the policy (Shen et al., 2020). By choosing the standard deviation properly, in addition to robustness, we were able to learn a minimal yet efficient set of recovery strategies, as well as to adjust the responsiveness and reactivity of the policy on the real device. A further improvement is the introduction of the L1-norm in temporal regularization. It still guarantees that the policy reacts only if needed and recovers as fast as possible with minimal action, but it also prevents penalizing too strictly short peaks corresponding to very dynamic motions. It is beneficial to withstand strong pushes and reduce the number of steps. Finally, it is more appropriate to penalize the mean of the policy $\mu_\theta(s_t)$ instead of the actions $\pi_\theta(s_t)$ as originally stated. It provides the same gradient with respect to the parameters θ but is independent of the standard deviation σ, which avoids penalizing exploration.

Although the temporal regularization acts on the same quantity as the acceleration bounds previously introduced, their effects are complementary. While regularization shrinks the acceleration globally regardless of its magnitude, the explicit constraint suppresses peaks without penalizing large but otherwise valid values. The combination of both enables fine-tuning the behavior of the policy more easily. In practice, we set the hyperparameters to $\sigma_S = 0.13, \lambda_T = 0.2, \lambda_S = 0.4$.

6.5 Ensuring Robustness for Bridging the Simulation-Reality Gap

6.5.1 Plausible External Disturbances

Patient Momemtum

The user has a huge impact on the stability of the robot (cf. section 1.1.4). Usually, they have limited control of their upper body, so they are not standing upright in the

exoskeleton as expected in the model but rather leaning in front. This discrepancy may be sufficient for the robot to fall backward or on the side. It may also walk in place or even stumble before falling forward if the feet stick to the ground. Being robust to external pushes does not help. Unlike reactive stepping, compensating the patient disturbances requires continuous adaptation and online estimation of the stability to some extent.

We apply a pure momentum at the root of the pelvis between the hips. This is not representative of a user in the exoskeleton, the true dynamics is almost impossible to model. It is supposed to encompass the worst case if tuned properly. The value of the momentum is slowly varying over time to further improve the robustness. The temporal evolution is defined by an aperiodic random Perlin process with a wavelength of 5s and 6 octaves. It ranges from $-50\,\mathrm{N\,m}$ (right) to $50\,\mathrm{N\,m}$ (left) and $-10\,\mathrm{N\,m}$ (back) to $90\,\mathrm{N\,m}$ (front) for x- and y-axes respectively.

Extrinsic Periodic Impulse Force

To learn sophisticated recovery strategies, the external pushes in the learning environment need to be thoughtfully scheduled. They must be strong enough to require stepping most of the time, but pushing too hard would prohibit learning.

As suggested by (Ferigo et al., 2021), we apply forces for a short duration periodically on the pelvis, and the orientation is sampled from a spherical distribution. The pushes are bell-shaped instead of uniform for numerical stability. They are applied during 400ms (interval of time during which the magnitude of the force is larger than 1%). In this work, the peak magnitude of the pushes is not constant over time but gradually increased until it reaches the time limit T_f. It can be interpreted as a kind of curriculum learning (cf. section 3.2.2). The most effective recovery strategies may be utterly superfluous for small pushes. Therefore, the initial magnitude must be large enough for all the desired recovery strategies to be useful right from the start, otherwise premature convergence to a suboptimal policy would be observed. In this work, it ranges from 800N to 1600N. The pushes are applied every $3s$, with a jitter of $1s$ to not overfit to a fixed push scheme and learn recovering consecutive pushes.

The benefits of allowing some hysteresis after pushes are clearly identified but introduce unexpected side effects. If the direction of the successive pushes is sampled independently, then they would most probably be roughly opposite. Thus, the effect of hysteresis is going to cancel itself without any effort from the agent to prevent its accumulation. Later on, the real robot would find itself in an increasingly bad configuration when pushed repeatedly in the same direction and finally fall. To mitigate this undesirable phenomenon, successive pairs of pushes are systematically sampled with the same orientation during learning in simulation.

6.5.2 Feasible Initial State Generation

The initial state distribution ρ_0 must ensure that the agent has to cope with a large variety of situations to promote robust recovery strategies.

Naive uniform random sampling for the whole state space is not suitable. First, it is not representative of reality as many of them would never occur in practice. Secondly, it would generate many unrecoverable states or even trigger early termination instantly. The resulting episodes have limited value and can make learning unstable. Ideally, it should span all recoverable states, but even the theoretical model of the system is too complex to compute this set analytically. One option would be to limit ourselves to the 1-step capturable states and to rely on an approximate model for which computations are tractable, e.g. the *Linear Inverted Pendulum Model* (LIPM). Yet, it would be preferable to be able to determine the recoverable states without requiring expert knowledge. Another approach consists in initializing the state randomly and discarding the ones that can lead to instant failure. However, it may be extremely inefficient for high-dimensional systems such as legged robots.

We propose to sample the initial state uniformly among the nominal motions and to add Gaussian noise for increasing the diversity. Its standard deviation must be large enough for robustness and versatility, but small enough to mostly output recoverable states. This approach would be effective if the set of nominal motions is rich enough. The current target velocity $\tilde{\dot{q}}_m$ is the predicted action, and the current target position $\tilde{q}_m$ is obtained by integration of its previous value according to the observation. Therefore, an initial distribution of previous target positions must be specified. In theory, it depends on the current policy π, which is unknown to the environment. Thus, we define a fixed prior distribution $\tilde{q}_m \sim N(\hat{q}_m, 1\text{deg})$. It is not clear whether this distribution has an impact on the training process.

6.5.3 Domain Randomization

The combination of the various techniques previously introduced is sufficient to transfer the policy from simulation to reality. It is nonetheless relevant to also rely on domain randomization because it is complementary. This avoids overfitting the physical parameters that are unknown or may change over time, e.g. the dynamic parameters of the mechanical deformation or the ground friction coefficient.

Domain randomization has no computational cost and is easy to implement. Yet, excessive randomization does more harm than good. First, it is increasingly unlikely to learn a policy that is effective for all values as the range increases. Next, it trades performance over robustness when randomizing parameters that are not observable since the agent has to rely on the exact same behavior no matter their current values. In such a case, it is recommended to settle for their true values.

System Dynamics

The relative CoM placement p_i, mass m_i and inertia matrix I_i of each body i in the kinematic tree are slightly modified around their respective theoretical values $\hat{p}_i, \hat{m}_i, \hat{I}_i$. It is a bit tricky for the inertia since its positive semi-definite property must be preserved. The principal axes $\hat{Q}_i$ and moments $\hat{\lambda}_i$ are extracted first through Eigenvalue decomposition, then the moments are perturbed and a small rotation is

applied to the principal axes. The inertia is given by $Q_i \text{diag}(\lambda_i) Q_i^T$. It yields,

$$p_i \sim N(\hat{p}_i, 0.15\%), \tag{6.56}$$
$$m_i \sim N(\hat{m}_i, 0.2\%), \tag{6.57}$$
$$Q_i \sim \hat{Q}_i \exp_3(N(0_3, 0.1)), \tag{6.58}$$
$$\lambda_i \sim N(\hat{\lambda}_i, 0.1\%), \tag{6.59}$$

where $\exp_3$ denotes the exponential map for the Lie group $SE(3)$. The position ${}^{i-1}\hat{p}_i$ of each body i relative to its parent $i-1$ is also slightly altered,

$${}^{i-1}p_i \sim N({}^{i-1}\hat{p}_i, 0.01\%). \tag{6.60}$$

In addition, the stiffness parameter of each deformation point is completely randomized. Then, the damping parameter ν_i is derived from it to ensure critical damping, and the inertia I_i is adjusted for numerical stability. The χ^2 distribution with 4 degrees of freedom is used for sampling the stiffness parameter k_i. It is always positive and has a long tail, so that large values are likely to be sampled. It follows,

$$k_i \sim 3000(1 + \chi_4^2), \tag{6.61}$$
$$I_i = k_i \left(\frac{\pi}{3} f_l\right)^{-2}, \tag{6.62}$$
$$\nu_i = 2\sqrt{k_i I_i}. \tag{6.63}$$

Finally, the viscous friction ν_i of each joint is also randomized, with gamma distribution to be always positive,

$$\nu_i \sim \hat{\nu}_i \Gamma(10, 0.1). \tag{6.64}$$

Hardware

The communication between the main board and the peripherals has a jitter: the update of the motor commands and sensor measurements occurs with a random delay. The policy tends to generate high-frequency vibrations on the real robot. Smoothness conditioning is sufficient to mitigate this phenomenon, but adding a jitter in simulation can only be beneficial. In practice, we update the sensor data with a small delay ranging from 0 to 4ms and uniformly sampled at every timestep.

Environment

The friction coefficient varies with the material of the ground. Randomizing it during training prevents overfitting one particular value. The agent would have to learn recovery strategies that are effective regardless of the friction coefficient and therefore transfer to reality more easily. We uniformly sample its value from 0.5 to 2.5. The maximum friction is unrealistic but discourages the agent to slide on purpose.

6.6 Results

6.6.1 Policy Network Architecture

The policy and the value function have the same architecture but do not share parameters. The only way to ensure safety is to limit their expressiveness, i.e. minimizing the number of parameters. It guarantees good generalization ability to avoid overfitting, and thereby leads to more predictable and robust behavior on the real robot, even for unseen or noisy observations. However, it hinders the overall performance of the policy. Different networks with varying depths and widths have been assessed by grid search. The final architecture has 2 hidden layers with 64 units each. Below this size, the performance drops rapidly.

6.6.2 Training Performance

We train our policy using the open-source framework RLlib (Liang et al., 2018) with refined PPO parameters. The episode duration is limited to $T_f = 60s$, which corresponds to $T = 1500$ time steps. In practice, 100M iterations are necessary for asymptotic optimality under worst-case conditions, corresponding to roughly four months of experience on a real robot. It takes about 6 hours to obtain a satisfying and transferable policy, using 40 independent simulation workers on a single machine with 64 physical cores and 1 GPU Tesla V100.

The training curves of the average episode reward and duration in figure 6.5 show the impact of our main contributions:

- Smoothness conditioning slightly slows down the convergence but does not impede the asymptotic reward.
- Using a simplistic reward, namely +1 per time step, similar training performance can be observed until 25M thanks to the well-defined termination conditions. After this point, the convergence gets slower and the policy slightly underperforms at the end. This result validates our convergence robustness and that our reward provides insight into how to recover balance.
- Without the termination conditions for safety and transferability, faster convergence in around 30M is achieved. It is consistent with (Ferigo et al., 2021; Li et al., 2021; Melo et al., 2020). However, it would not be possible to use such a policy on the real device.

6.6.3 Validation in Simulation

Closed-loop Dynamics

Figure 6.6 shows that smoothness conditioning improves the learned behavior, cancels harmful vibrations, and preserves dynamic motions. Moreover, it also recovers balance more efficiently, by taking shorter and minimal actions.

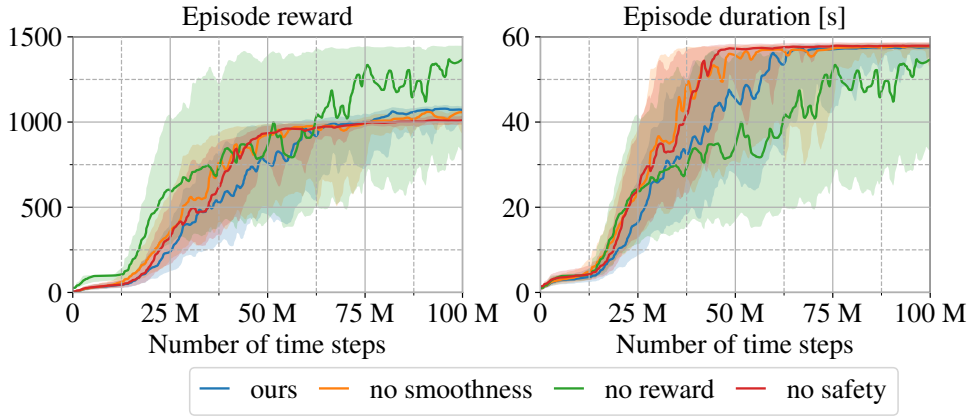


Figure 6.5: Ablation study of our proposed reward formulation, policy regularization for smoothness, and termination conditions for safety. The standard deviation is taken over episodes per batch.

Vibrations in Standing

Our regularization is a key element in avoiding vibrations in standing on the real device and promoting smooth actions. The effect is clearly visible in the target velocity predicted from our policy network, see figure 6.6. The target velocity without regularization leads to noisy positions and bang-bang torque commands, whereas our proposed framework learns an inherently smooth policy. Moreover, using L^1-norm for the temporal regularization enables us to preserve the peaks at 13.8s and 17.5s, which is critical to handle pushes and execute agile recovery motions.

Steady-state Attractiveness and Hysteresis

Once pushed, the robot recovers balance with minimal action, see figure 6.6. It quickly goes back to standing upright close to the nominal. A small hysteresis is observed, see figure 6.9a. It does not affect the stability and will vanish after the next push. It avoids doing an extra step to move the feet back in place.

Analysis of Learned Recovery Strategies

Different recovery strategies are generated and tested in simulation for horizontal pushes on the pelvis. We build an interactive playground in Jiminy, to test the trained policy with pushes from all sides. Depending on the direction and application point, a specific recovery is triggered and a different maximal magnitude for the force can be handled, see figure 6.8. In-place strategies, like the ankle or hip strategies, are sufficient for small pushes from any direction. Strong front, back, and diagonal pushes are handled with reactive stepping strategies, and even jumps, while side pushes activate a different behavior performed with the ankles, see table 6.1. For side pushes, the robot twists the stance foot alternating around the heel and the

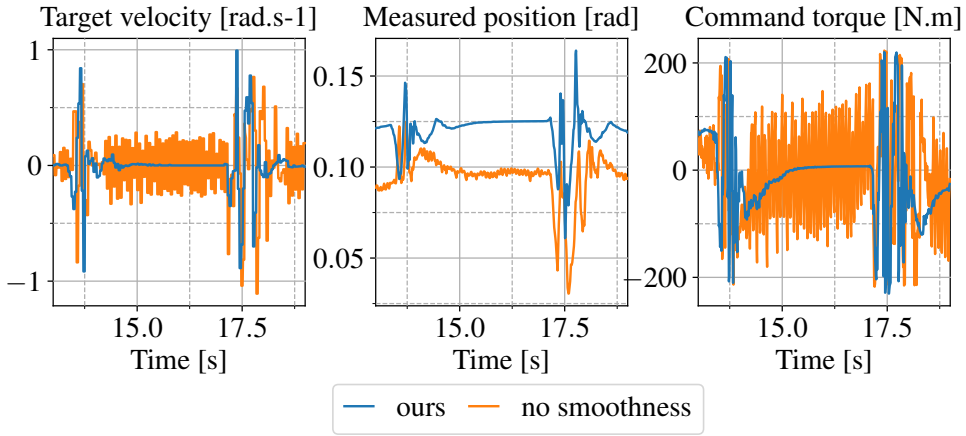


Figure 6.6: Comparison between the predicted target velocity, measured position, and command torque for the left knee joint with and without smoothness conditioning.

Figure 6.7: Strong impact kick, 1 recovery step per frame in the highlighted section.

Emerging strategy	front	back	diagonal	side	small
Ankle control	✗	✗	✗	✓	✓
Hip control	✗	✗	✗	✓	✓
Stepping	✓	✓	✓	✗	✗
Jumping	✓	✗	✓	✗	✗
Foot tilting	✓	✓	✗	✗	✓
Angular momentum	✗	✗	✗	✓	✗

Table 6.1: Overview of emerging strategies for pushes from all sides

toes while balancing the opposite leg, figure 6.9d. This dancing-like behavior avoids weight transfer, which was found the most difficult to learn. A larger variety of push recovery strategies on the real device are displayed in the supplementary video[1].

Recent results in simulation on the Valkyrie robot (Yang et al., 2018) invite for comparison. The humanoid has roughly the same weight as the exoskeleton Atalante carrying a dummy ($135kg$) and the height of an average human person ($1.8m$). To compare impulses, the applied forces are put in relation to the duration of the push. We can handle impulses of about $190Ns$ for sagittal pushes on the pelvis with our

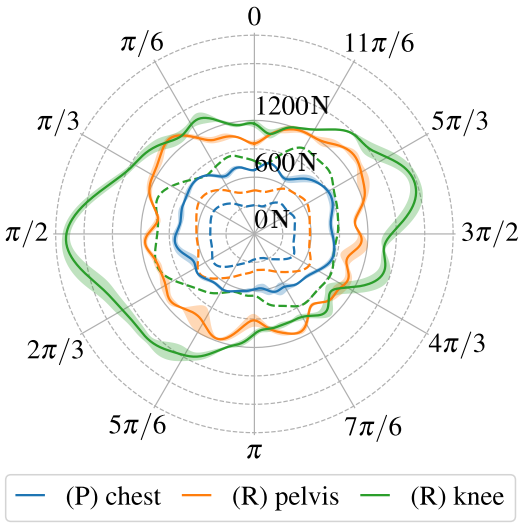


Figure 6.8: Maximum recoverable force magnitude F (bell-shaped with duration of 400ms) applied from any direction in plane $(\cos(\varphi), \sin(\varphi), 0)$ at several locations on the patient (P) and the left side of the robot (R). Solid and dashed lines are associated with our policy and PD tracking of the nominal standing pose respectively.

safe policy shown in figure 6.8, compared to $240Ns$ for Yang *et al.* (Yang et al., 2018). This is satisfactory considering that the motor torque limits are about 50% lower on Atalante and knowing that the safety constraints are limiting our performance.

6.6.4 Standing Push Recovery on Atalante

The trained control policy is evaluated qualitatively for both valid users and dummies of different masses on several Atalante units in the Wandercraft offices. Even though the robot is only pushed at the pelvis center during the learning, figure 6.5 strongly suggests that the policy can handle many types of external disturbances. We push the robot in reality at multiple application points and obtain impressive results, see figures 6.7 and 6.9. The recovery strategies are reliable for different push variations and pulling. The transfer to Atalante works out-of-the-box despite the wear of the hardware and modelling uncertainties, notably the ground friction, the mechanical deformation of the structure, and the patient disturbances.

6.7 Concluding Remarks

We obtain a controller that provides robust and versatile recovery strategies. Several techniques are combined to promote smooth, safe, and predictable behaviors, even for unexpected events and unrecoverable scenarios. As theoretical guarantees are limited, our method was only verified empirically in simulation and reality. The policy was easily transferred to the exoskeleton Atalante. Even though trained for a

(a) front push shoulder (b) strong kick pelvis

(c) random user moves (d) side push hip

Figure 6.9: Scenarios showing the robustness of the push recovery policy

single average patient model, our policy was validated experimentally with different users and dummies. It performed successfully a variety of recovery motions against unknown perturbations at various locations. To our knowledge, we are the first to demonstrate reactive push recovery at rest on a real humanoid robot using RL.

Our framework is generic and could theoretically be applied to any nominal motion in order to stabilize it and provide recovery capability. For now, walking was put aside because it is more challenging than standing. Indeed, the stability is precarious during single support phases and the mechanical deformation of the structure becomes problematic. We are planning to unify walking and push recovery in future works. Besides, our framework can be adapted to other bipedal robots, and it would be interesting to compare the performance on other platforms. Further research directions include switching to more sample-efficient off-policy algorithms and enhancing exploration via curiosity-based intrinsic reward.

Chapter 7

Conclusion

Contents

7.1 Summary of the Contributions 203
7.2 Discussion 205
7.2.1 Online Trajectory Planning 205
7.2.2 Robust and Safe Control Policy 205
7.3 Perspectives and Future Works 205
7.3.1 Human-Like Locomotion 205
7.3.2 Data Efficiency 206
7.3.3 Global Optimally 206
7.3.4 Transfer to reality 206

7.1 Summary of the Contributions

The agile and versatile locomotion of biped robots has been an active research topic since the 70s. The theoretical analysis of the controllability of these systems despite their inherent instability has been the primary matter of interest in early studies. Today, these aspects are well understood, and it is rather the limited capability of the embedded hardware (actuators, sensors, and computing power) related to the necessity of being commercially profitable that contributes for the most part to making biped locomotion a real challenge.

In this thesis, we have developed motion planning and control algorithms for legged robots that can be readily applied to real devices. They yield a strict yet modest improvement over the existing methods. All our contributions have been validated experimentally on the exoskeleton Atalante. In this context, enhancing the quality of life of the users while guaranteeing a safe and predictable behavior of the robot has been major a concern throughout this work.

Our first contribution focuses on motion planning. The concrete objective is to enable the user to change high-level gait features on-the-fly. It is a step toward versatile locomotion since stability is disregarded. We achieve this by training offline a parametric function approximation of the trajectories generated with a motion planning framework over a continuous task space. The large prediction error resulting from the application of the naive standard regression prevents us from guaranteeing

the safety of the user when blindly tracking the predicted trajectories. To overcome this limitation, we propose the *Guided Trajectory Learning* (GTL) algorithm that ensures high accuracy regardless of the expressiveness of the function approximation. Our algorithm is very generic and can be used for solving any multi-parametric optimization problem beyond the robotic field. The function approximation takes the form of a deconvolution neural network. This architecture is specially designed for generating smooth time series. It significantly reduces the computation cost at runtime and guarantees the desired regularity properties of the predicted trajectories. A simplified variant of our algorithm has been successfully validated on Atalante with valid users despite having weaker guarantees in terms of accuracy.

Our second contribution focuses on robust control using reinforcement learning. The ultimate objective is to achieve human-like navigation in an unstructured environment while avoiding falling. For simplicity, two basic scenarios have been considered: smooth transitioning between nominal motions, and emergency push recovery while standing. We propose an original formulation closely related to imitation learning. A pre-defined set of primitive motions that have been validated clinically is used to guide and constrain the optimization of the policy. First, we introduced generalized space-time bounds that ensure minimal deviation from the nominal trajectories while giving enough freedom to deal with external disturbances or adapt to the real environment. We demonstrate mathematically that early termination is sufficient to enforce these implicit constraints. Secondly, we designed a lower-bound maximization algorithm for solving multiple tasks of heterogenous difficulty over a continuous domain. Together, they enable training from scratch a single policy capable to perform all nominal motions at once equally well. Safety and predictability are encouraged using a combination of spatial and temporal regularizations called smoothness conditioning. This approach is traditionally found in supervised learning for robustness against adversarial attacks. We observe that it leads to the clustering of the closed-loop behavior for standing push recovery: the agent learns a minimal set of highly generic and effective strategies. In addition, we present a method inspired by the penalty and barrier function methods to enforce explicit constraints that only depend on the policy by taking advantage of their analytical gradient. It is used to bound the maximum acceleration of the motors, which eases transfer to reality.

To support this work, an open-source simulator of poly-articulated robots called Jiminy has been developed. It is heavily optimized for reinforcement learning applications. In particular, several parameters are available to trade-off between realism and regularity of the physics to ease or speed up learning. Internally, it relies on a novel analytical contact formulation that does not involve computing impulse forces. Besides, it takes into account many of the hardware limitations and side effects, among them backlashes, communication delays, the inertia of the rotors, and the mechanical deformation. All simulations are perfectly repeatable and Jiminy offers fine-grained monitoring and visualization utilities to facilitate their analysis.

7.2 Discussion

7.2.1 Online Trajectory Planning

Our GTL algorithm has several limitations that may hinder its applicability. First, our certificate of safety is based on the prediction error to be completely agnostic to the dynamic model of the system. This is convenient but overly restrictive since a large prediction error does not entail infeasibility, which is what really matters. Moreover, it puts more pressure than necessary on the global Lipschitz constant of the function approximation and the number of samples to collect. Secondly, the image of the function approximation and the feasibility domain of the motion planning problem may be disjoint for a given task. If so, the corresponding prediction error may be arbitrarily large theoretically, invalidating our certificate of safety.

7.2.2 Robust and Safe Control Policy

A policy trained for standing push recovery as part of our second contribution has been assessed experimentally. It transfers to reality and attains satisfactory performance without any kind of adaptation, which is very promising. However, we were unable to provide results for the second training scenario related to versatile locomotion, which is disappointing. In practice, we observe that even learning to walk in a straight line on flat ground does not transfer to reality. The robot keeps walking in place without moving forward because its feet stick to the ground. Supposedly, this is caused by the dynamic model of the mechanical deformation being off. However, the domain randomization technique is not sufficient to handle this discrepancy, probably because our simulation environment is not challenging enough. This indicates that the agent has never learned what it means to move forward, which raises concerns about our ability to transfer the policy to the reality of more sophisticated behaviors. How to design a simulation environment for which maximizing the expected return urges the agent to learn the expected skills is an open question.

7.3 Perspectives and Future Works

7.3.1 Human-Like Locomotion

Much remains to be done regarding robust human-like locomotion in unstructured environments. The logical follow-up would be to combine our work on learning multiple trajectories at once and transitioning with our emergency push recovery while standing. If successful, then it would be worthwhile to enable changing the high-level gait features on the fly over a continuous task space. To this end, one could train the policy for discovering suitable trajectories on its own. This approach has already been proven effective by Won et al. (2020). However, it may be hard to reconcile with our imitation learning paradigm that aims to minimize the deviation for the nominal trajectories. A better option may be to simply replace the finite set of gaits with continuous function approximation generated using GTL. Concurrently, studying the

emergence of human-like locomotion without pre-defined motion to guide the policy is an interesting research direction. Great results have been obtained by Lee et al. (2019) in the field of computer-aided animation using realistic muscle-based dynamic models of the human body. Yet, it has never been applied to a real robot for now.

7.3.2 Data Efficiency

The wall time for training a policy for push recovery using *Proximal Policy Optimization* (PPO) – a state-of-the-art on-policy learning algorithm – is about 6 hours on a single server-grade computer. This is mostly due to the need of collecting more than 50 million samples to fully converge. Off-policy learning algorithms such as TD3 could substantially improve the sample efficiency and hence reduce the wall time, but they are known to converge to highly sub-optimal solutions.

A more promising approach may be to reduce the variance of the on-policy gradient estimate using an action-dependent control variate based on stein identity (Liu et al., 2018). This would reduce the training batch size per iteration, which is very high in our case as it corresponds to 50 minutes in simulation time. It is also possible to mix on- and off-policy gradient estimates using hybrid algorithms, e.g. Q-prop.

7.3.3 Global Optimally

We have been able to learn very effective and robust strategies for push recovery while standard. They are nonetheless fairly basic and only locally deviate from the resting pose. For instance, none of them involves transferring the weight from one leg to the other on purpose, typically by bending the knee of the supporting leg. Yet, this is essential for doing sidesteps when already falling. This may be slightly mitigated by using a more advanced optimizer than *ADAptive Moment estimation* (ADAM) that is ubiquitous in machine learning, notably K-FAC (Martens & Grosse, 2015). However, this is likely to be insufficient as the main limitation is generally the exploration. Relying on curiosity-driven intrinsic rewards to promote behavioral novelty is more relevant. It has already demonstrated its efficiency for evolution strategies (Lehman & Stanley, 2011). It is an active research topic in *Reinforcement Learning* (RL), with encouraging preliminary results (Badia et al., 2020).

7.3.4 Transfer to reality

The disputable realism of the physics engines is a major issue. For example, it is extremely difficult to train a policy for moving around without entirely lifting the feet. The trick is usually to lift the feet much higher than necessary, but it is far from satisfactory. Indeed, being overly restrictive on what is considered acceptable behavior in simulation impedes the overall performance of the policy. Moreover, it is often necessary to rely on the domain randomization technique or the like, which leads to sub-optimal policies in reality. Yu et al. (2017) overcome this limitation by learning a generic policy taking as input the unknown parameters and identifying them online

on the real robot later on. It would also be helpful to reduce the reality in the first place by improving the simulator itself. In the particular case of the exoskeleton Atalante, one may consider the patient as an independent system having its own dynamic model and interacting with the robot through non-penetration conditions and spring-damper mechanisms for the straps. We could also perform a system identification of the unknown parameters once and for all using a dedicated apparatus such as motion capture. In addition, the simulation could be made hybrid and integrate data-driven building blocks trained using supervised learning, e.g. the transfer function of the actuators (Hwangbo et al., 2019). Finally, the impact of the reality gap could be reduced by learning a policy that predicts high-level features to an advanced traditional low-level controller, typically the placement and timing of the future footprints for a whole-body *Model-Based Predictive Control* (MPC).

Appendices

A Trajectory Planning Using Whole-Body Optimization

Being able to generate theoretically stable primitive motions for the exoskeleton is an essential building block of the thesis. The work on stable dynamic walking with paraplegics for the exoskeleton Atalante started before the beginning of this thesis but was refined in the meantime to make the gait pattern more physiological, further improve stability, and extend it to other motions. The methodology is based on the *Hybrid Zero Dynamics* (HZD) theory. The latter has been developed for the planning and control of under-actuated bipedal robots (Westervelt et al., 2003). Nevertheless, it is relevant for any bipedal robot as a way to automatically adapt to disturbances. This approach has proven effective for the control of planar bipedal robots with point feet. However, its naive application to realistic humanoids was unfruitful (Finet, 2018). Later, Hereid et al. (2018) successfully validated the HZD theory on many bipedal robots with practical value, including DURUS, MARLOS and Cassie.

A sparse *Non-Linear Program* (NLP) is obtained via the Direct Collocation transcription to ensure scalability to very large problems. Huynh et al. (2021) generate trajectories involving more than 10 independent domains coupled together for the exoskeleton Atalante using this framework in less than 5 minutes, corresponding to about 50000 variables. Many other planning methods exist. The most successful alternatives are based on *Differential Dynamic Programming* (DDP) (Budhiraja et al., 2019b; Mastalli et al., 2020). The interested reader is encouraged to look at (Goswami & Vadakkepat, 2019, Part VII) for details.

A.1 System Modeling

A.1.1 Hybrid System Model

Let us consider a robot with a free-floating base and n joints. The classical generalized coordinates for such a system is

$$q = (p, \phi, q_J) \in \mathcal{Q}_q, \quad u \in \mathcal{U}_q \tag{A.1}$$

where $p \in \mathbb{R}^3, \phi \in SO(3)$ denote the position and orientation of the base frame $\mathcal{R}_b$ in world frame, $q_J = (q_1, q_2, \cdots, q_n) \in \mathcal{Q}_{J,q}$ is the vector of joint configurations and

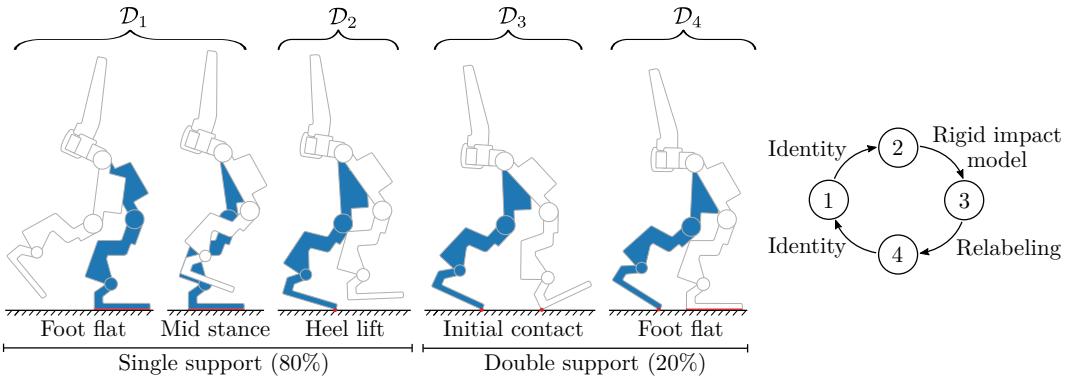


Figure A.1: Directed graph of foot rolling walking pattern for the exoskeleton Atalante. The motion is only optimized for left leg support and the relabeling trick is used to skip right left support. It half the number of continuous domains of the problem, so that it is easier to solve it numerically.

$u \in \mathcal{U}_q$ denotes the *controls*. For instance, q_J is simply the vector of the relative angles of all joints $q_i \subset \mathbb{R}$ for a robot having only bounded revolute joints, and the controls u are the torques applied by the motors to all the actuated joints individually. $\mathcal{Q}_q, \mathcal{U}_q$ encompass the physical limitations of the joints of the robot. In general, they are hypercubes that do not depend on the current configuration q, but it can get more complex if the transmissions between the actual actuators and the modelling joints are nonlinear, as for the ankle of the exoskeleton Atalante. The subscript is dropped for the ease of notation. Similarly, the velocity of the joints $\dot{q}_J$ is likely to be bounded, in particular the actuated joints because of the viscous friction. Apart from that, it makes sense to, restrict artificially the admissible velocity for the ease of the patient and safety, while improving convergence rate by discarding irrelevant state upstream. The state of the system x simply gathers the position and velocity,

$$x = (q, \dot{q}) \in \mathcal{D} \subseteq \mathcal{TQ} \tag{A.2}$$

where $\mathcal{D}$ represents the admissible states, $\mathcal{T}_q\mathcal{Q}$ is the tangent space of the manifold $\mathcal{Q}$ at point q and $\mathcal{TQ} = \bigcup_{q \in \mathcal{Q}} \mathcal{T}_q\mathcal{Q}$ is its tangent bundle.

The dynamic behavior of bipedal locomotion is characterized as a hybrid system consisting of a sequence of continuous and discrete domains. Each continuous domain is related to a specific contact phase, e.g. single support with flat foot, double support with both feet flat, or rolling contacts around the heels and toes. Discrete events happen when transitioning between continuous domains. The mapping linking the final state of previous continuous domain to the initial state of the next one is called *reset map* and denoted Δ. It is the *switching surface* $S \in \mathcal{D}$, also called *guard surface*, that determines the specific condition that triggers the transition from one domain to next one. Formally, it constitutes a cyclic directed graph $\Gamma = (V, E)$ whose nodes $v \in V$ are continuous domains and directed edges $e = v \to v+ \in E$ are discrete events. The directed graph for foot rolling walk is depicted in figure A.1.

As extensively explained in appendix C.3.2, the continuous dynamics can be written as an *affine autonomous control system*,

$$\dot{x} = f(x) + g(x)u, \tag{A.3}$$

where x is the system state. The functions f, g depend on the active contact points, which are specific to the current contact phase. Mathematically, it implies that the control system is different for every domain.

In the particular case of flat foot walk generation, there is one continuous domain corresponding to single support on right leg, and one discrete event when the swing leg touches the ground flat. Note that the rigid impact model does not allow for having a double support phase in planning, but there is one in practice because of the mechanical deformation of the structure. The symmetry of the gait relative to the sagittal plane is leveraged to avoid optimizing for both side. It has the advantage to guarantee the symmetry of the gait without dedicated equality constraint. The resulting hybrid system dynamics can be written as

$$\Sigma : \begin{cases} \dot{x} = f(x) + g(x)u(x) & , x \notin S \\ x^+ = \Delta(x^-) & , x \in S \end{cases}, \tag{A.4}$$

where the vector fields f, g are from the continuous-time swing-phase dynamics and u is the feedback controller that depends on the current state.

A.1.2 Continuous Dynamics

As thoroughly explained in appendix C.2.1, the interaction of the robot with the environment can be formulated as a holonomic constraint $f_c(q) = 0$. This constraint is the same for every single contact point: it gathers all the locked *Degree of Freedom* (DoF) of the corresponding frame, typically the three translations. Then, the overall constraint is obtained by concatenating the one acting at every contact point.

Although approach is mathematically valid, it induces many redundancies in practice. The ensuing optimization problem would be ill-posed, and the numerical solver would be prone to optimization failure. Notably, enforcing the position of several points that are fixed relative to the same body is problematic. There is no easy way to get around this issue in simulation. Nevertheless, the contact phase is usually known in advance in planning since it is part of the specifications for the motions being generated. For instance, one foot may be fixed on the ground while the other one is rolling around its heel during a given phase. This property can be leveraged to get rid of the redundancies. It follows that the constraints applied locally on the DoF of every contact point can be unified to operate directly on the DoF of their respective parent body. Doing so, the overall constraint is now obtained by concatenating the one acting at each body being in contact in at least one point. This approach effectively avoids any redundancy for the vast majority of legged robots, in particular if the interaction with the environment is limited to the end-effectors (i.e. the feet and hands for a humanoid robot).

To enforce this holonomic constraint, it is sufficient to set its acceleration to zero, along with its initial position and velocity,

$$\forall (q, \dot{q}) \in \mathcal{D}, \forall \ddot{q} \in \mathcal{TQ}, J_c(q)\ddot{q} + \dot{J}_c(q, \dot{q})\dot{q} = 0 \qquad \text{Domain acceleration} \tag{A.5}$$

$$J_c(q_0)\dot{q}_0 = 0 \qquad \text{Initial velocity} \tag{A.6}$$

$$c(q_0) = 0 \qquad \text{Initial position} \tag{A.7}$$

where $(q_0, \dot{q}_0)$ is the initial state of the robot. In simulation, Baumgarte stabilization is used to drive the velocity toward zero exponentially instead of the enforcing the initial position and velocity. This difference stems from the fact that the contact is not necessarily stable in simulation while it is a prerequisite in planning.

Let us recall the *Karush-Kuhn-Tucker* (KKT) conditions of the constrained optimization problem:

$$\left(J_c H^{-1} J_c^T\right) \lambda = a_{c,\text{free}} \tag{A.8}$$

$$a_{c,\text{free}} = J_c(q)\ddot{q}_{\text{free}} + \dot{J}_c(q, \dot{q})\dot{q} \tag{A.9}$$

$$\ddot{q}_{\text{free}} = H^{-1}(u - C(q, \dot{q})\dot{q} - G(q)) \tag{A.10}$$

where λ is the vector of lagrangian multipliers. As mentioned in section 2.1.2, it contains the efforts associated with the locked DoF due to the holonomic constraints. The vector of lagrangian multipliers is abusively referred to as the contact wrench f for the sake of simplicity.

One can show that the vector of lagrangian multipliers can be computed in closed-form as long as the Jacobian matrix $J_c(q)$ is full-rank. If so, then the closed-form expression of the contact wrench f is given by,

$$f = A_c + B_c u, \tag{A.11}$$

where $\Xi = \left(J_c H^{-1} J_c^T\right)^{-1}$ is the *Operational Space Inertia Matrix* (OSIM), $A_c = \Xi(J_\eta(q)H^{-1}(C(q, \dot{q})\dot{q} + G(q)) - \dot{J}_c(q, \dot{q})\dot{q})$ and $B_c = \Xi J_c(q)H^{-1}$. The jacobian J_c is full-rank if and only if there is no redundancy in the holonomic constraint, otherwise the solution for the contact wrench is not unique. Physically, it means that an infinity of contact wrenches would satisfy the holonomic constraint, all resulting in the same generalized acceleration $\ddot{q}$. A pseudo-inverse could be employed to pick one solution arbitrarily, but it is unlikely to be the only one that is physically meaningful.

The unified formulation of the contact at body level is preferred over the naive concatenation for every contact point. This is so because the Jacobian $J_c(q)$ of the unified formulation is guaranteed to be full-rank for the exoskeleton Atalante (ignoring side effects due to the joint bounds). It is true even in double support as there is at least 6 independent DoFs in the kinematic chain going from one foot to the other plus 6 additional DoFs for the floating base. Consequently, it is possible to control the position and orientation of both feet completely independently.

At this point, one can substitute the closed-form expression of the contact wrench in the whole-body dynamic equations of motion (equation (2.33)). This resulting

equation of motion is referred to as the *reduced dynamics* because the system becomes mathematically analogous to a fixed-based robot: the contact forces does not appear anymore and the state is restricted to the lower-dimensional manifold satisfying the holonomic constraint. Although appealing, the reduced dynamics is rarely used in motion planning as it makes the whole optimization slower empirically. There are mainly two reasons: computing the reduced dynamics is expensive, and more importantly, it impedes the sparsity of the problem. Thus, the general equation of motion and the holonomic constraint for the contact are usually enforced as two separated equality constraints.

Anyhow, additional constraints must enforce contact stability since it is a prerequisite. The classical *Zero-tilting Moment Point* (ZMP) criteria presented in section 2.2.2 cannot be used because the support polygon could be degenerated during some phases of the motion. To get round this limitation, inequality constraints ensures that the pressure force at any contact point is always positive and Coulomb friction law is satisfied (cf. appendix C.2.2).

A.1.3 Discrete Dynamics

Rigid Impact Model

The rigid impact model has been formulated rigorously by Hurmuzlu and Marghitu (1994). It states that at impact, the configuration of the system q remains unchanged, but the velocity $\dot{q}$ undergoes a discrete jump due to the instantaneous changes in momentum. The jump in velocity is captured by the reset map Δ, which represents the relationship between the pre- and post- states x^-, x^+, i.e. $x^+ = \Delta(x^-)$.

Under the assumption of rigid impact, it is possible to compute the post-impact velocity q^+ given the pre-impact velocity q^-. The velocity of the swing foot is zero instantly after touching the ground and the former stance foot lift-off immediately. Formally, the velocities after impact verify,

$$J_c^+ \dot{q}^+ = 0, \qquad J_c^- \dot{q}^+ > 0, \tag{A.12}$$

where J_c^-, J_c^+ are the Jacobian of the pre- and post-impact holonomic constraints f_c^-, f_c^+ respectively, which are different since the stance foot has changed.

In the case of rigid impact with no friction in rotation along z-axis, Hurmuzlu and Marghitu (1994) have proven that the angular momentum is conserved. We have

$$H\dot{q}^+ - H\dot{q}^- = {J_c^+}^T \tilde{f} \tag{A.13}$$

where $\tilde{f} = lim_{t^- \to t^+} \int_{t^-}^{t^+} f dt$ is called ground reaction impulse and corresponds to the intensity of the contact wrench over the infinitesimal duration of the impact event.

The system of equations comprising equations (A.12) and (A.13) can be inverted in order to compute the post-impact velocity $\dot{q}^+$ in closed-form. It yields,

$$\dot{q}^+ = \dot{q}^- + H^{-1} {J_c^+}^T \tilde{f}, \qquad \tilde{f} = -\Xi^+ J_c^+ \dot{q}^-, \tag{A.14}$$

where $\Xi^+ = \left(J_c^+ H^{-1} {J_c^+}^T \right)^{-1}$ is the post-impact OSIM.

Periodicity Condition and Relabeling

All primitive motions as 2-steps periodic by definition, ignoring the offset of the freeflyer consequent to the actual displacement of the *Center of Mass* (CoM) of the robot in world frame. As a result, it is enough to optimize the motion over a complete gait cycle and enforce the periodicity condition. It must be checked while taking into account the desired displacement of the CoM. Thus, instead of enforcing the initial state of the first domain to match exactly the final state of the last domain, the difference must be equal to an offset that depends on the specification of the pattern, e.g. the step length, the walking direction or the turning angle. Computing the desired offset is trivial for pure translations and rotations in world plane, such as climbing stairs or turning in place, but it is not for more complex motions, such as turning while walking. How to compute this offset in the general case is out of scope.

It is possible to go even further by leveraging the symmetry of the pattern if any. It avoids optimizing the trajectory over a complete gait cycle to focus only on either right or left leg support, thereby reducing the computational cost and improving the convergence rate. For the pattern to be symmetric, the total displacement of the freeflyer must be a pure translation that is consistent with the mirroring plane of the robot. More precisely, the total displacement must be orthogonal to the sagittal plane of the robot, which coincides with y-axis of the local frame of the robot. It holds true for forward and backward walking, as well as stair climbing. Let us consider a motion primitive that is symmetric relative to the sagittal plane of the robot, so that the roles of the legs can be swapped. Swapping the role of each leg is called *relabeling*. It is a linear transformation that can be written as the product of two operations: swapping the state of each joint, and mirroring the state of each joint individually. The former is just a mapping the state of each joint to their counterpart on the opposite leg. It is a matrix that contains a single 1 for each row and column. The latter is more complicated. It is a block diagonal matrix in the general case, whose blocks are is a mirroring operation specific to each joint. Let us consider only single DoF joints. Any joints corresponding to pure translations or rotations can be decomposed in a chain of single DoF joints. It may not be the case for more complex types of joint, but they are ignored as rarely encountered in practice. In the case, the full mirroring matrix is diagonal and contains only the value -1 or 1. Indeed, any single DoF joints is fully specified by a constant spatial axis in parent joint frame. The mirroring operation relative to sagittal plane consists of flipping the components d_y, d_{O_x}, d_{O_z} components using Plücker notations, i.e. d_x, d_y, d_z is the basis for translation and $d_{O_x}, d_{O_y}, d_{O_z}$ is the basis for rotation.

For spherical joints, it is often preferable to use the unified quaternion representation instead of decomposing them in 3 revolute joints and use Euler angle representation, as presented in section 2.1. It is typically the case for the freeflyer state, that is commonly decomposed in 3 linear joints and a spherical joint. It is possible to perform the mirroring operation directly without further decomposition of the spherical joint: the x and z components of the quaternion and angular velocity are flipped, as if it was independent rotations. Nevertheless, the dimensionality is not the same

for the configuration and velocity, and therefore different relabeling matrices must be used for the position and velocity.

Reset Map

The reset map is specific to each transition between continuous domains. It can either be the identity, the rigid impact model, the relabeling function, or a combination of both impact and relabeling into a single map. For instance, the identity is used when lifting one foot in double support phase. The rigid impact model is used when the swing foot strikes the ground, no matter if it lands flat or on the heel. Relabeling is used to optimize only one side of the motion, which could gather several domains, for example foot rolling walking in straight line.

A.2 Gait Parameterization

A.2.1 Virtual Constraints and Trajectory Parameterization

The method of inverse dynamics is ubiquitous in the field of robotics. It consists of defining a set of outputs and then designing a feedback controller that asymptotically drives the outputs to zero. The task is encoded into this set of outputs in such a way that the nulling of the outputs is asymptotically equivalent to achieving the task. It can be any real-valued function of the state of the system x at time t, such as the velocity of the CoM. In this perspective, the controls are no longer free variables but rather a function of the state x and time t in the general case. In the case of bipedal locomotion, it is unnatural to explicit the time-dependency since walking arises from the coordinate movement of the joints across the step progress, called *phase*. Formally, the phase variable s can be any scalar differentiable function strictly monotonous over the step progress which only depends on the configuration of the robot q. Ideally, when the robot is disturbed by some unexpected event, the feedback controller should be responsible for going back to state that belongs to the reference trajectory, without the additional burden of re-synchronizing with time. *Virtual constraints* are used to synthesize such feedback controller.

Virtual constraints are relations between the joints – i.e. constraints – that are imposed through feedback control instead of a physical connection. Like physical constraints, they induce a reduced-order dynamic model called the *zero dynamics* that captures the natural dynamics of the robot under constraints. The outputs of the feedback controller are virtual constraints specifically designed to synchronize the evolution of the joints of the robot in order to create an attractive periodic motion. This type of controller also has the advantage to be time-invariant, which facilitates theoretical analysis. Only velocity-modulating outputs with constant references and position-modulating outputs are considered:

$$y_2(q, \alpha) = y_2^a(q) - y_2^d(\tilde{s}(q), \alpha) \tag{A.15}$$

$$y_1(x, v_d) = y_1^a(x) - v_d \tag{A.16}$$

where $\tilde{s}$ is the normalized phase variable going from 0 to 1 over the whole step, $y_1^d \in \mathbb{R}^{n_1}$ are velocity-modulating or relative degree 1 outputs and $y_2 \in \mathbb{R}^{n_2}$ are position-modulating or relative degree 2 outputs. The reference degree 1 outputs is denoted v_d. It is chosen constant for simplicity, but it is not mandatory. The reference degree 2 outputs are parametric functions of the vector α. α must be optimized to obtain a stable periodic motion. Bézier's polynomials of degree M have been used as parametric functions for reference degree 2 outputs, fully specified by $M+1$ coefficients called *control points*. They are defined as follows

$$y_2^d(\tilde{s}, \alpha) := \sum_{i=0}^{M} \alpha[k] \frac{M!}{k!(M-k)!} \tilde{s}^k (1-\tilde{s})^{M-k}. \tag{A.17}$$

Bézier's polynomials are used because they are smooth, easy to analyze graphically, and well-behaved no matter the value of parameters. More specifically, the convex hull of the control points contains the curve itself. By leveraging this property, it is possible to restrict the optimization to a sensible domain. Indeed, it is usually possible to guess a priori the domain where the outputs are expected to be, which is directly related to bounds on the Bézier coefficients α. It makes the numerical optimization more stable, and it converges faster and more reliability by avoiding looking for the solution where it is known in advance that it cannot be. Moreover, the derivative is guaranteed to be continuous and bounded, which is critical since numerical solvers expecting the gradient and hessian of the problem to be properly defined. The desired output functions y_1^d, y_2 is uniquely defined over the whole trajectory, and thus the same Bézier coefficients α are used across all continuous domains.

Using a feedback controller to drive the virtual constraints to zero is preferable over enforcing the trajectory to them match exactly during the whole motion. Imposing perfect matching of the virtual constraints strongly constraints the NLP, which affects the converge rate. Besides, it is unrealistic in most cases. For example, in the case of a velocity-modulating output on the forward velocity of the pelvis, it would enforce it to move at a constant speed during the whole step. Using a feedback controller instead gives more freedom to the optimization to find complex solution that cannot be represented as Bézier polynomials without significantly increasing their degree, whereas it is favorable to keep it as low as possible. Indeed, higher degree polynomials allow finding solutions that are less smooth, and thus more difficult to transfer on the real device and likely to feel uncomfortable for the user.

A.2.2 Input-Output Feedback Linearization

The objective is to design a feedback controller that is driving the virtual constraints to zero exponentially. Classically, it is achieved by means of *feedback linearization* (Finet, 2018; Hereid et al., 2018). Let us consider the first-order time-invariant formulation of the general dynamic equation of motion,

$$\dot{x} = f(x) + g(x)u, \tag{A.18}$$

where

$$f(x) = \begin{pmatrix} \dot{q} \\ H^{-1}(q)(C(q,\dot{q})\dot{q} + G(q)) \end{pmatrix} \qquad g(x) = \begin{pmatrix} 0 \\ H^{-1}(q) \end{pmatrix}. \tag{A.19}$$

Let h be any differentiable function of the state x of the system

$$\dot{h}(x) = \frac{\partial h(x)}{\partial x}\dot{x} = \frac{\partial h(x)}{\partial x}(f(x) + g(x)u) = L_f h(x) + L_g h(x)u \tag{A.20}$$

where $L_f h(x)$ is called the Lie derivative of $h(x)$ with respect to the differentiable vector field f. If h is holonomic, which means that it only depends on the configuration of the system q, then it follows

$$L_g h(x) = \left(\frac{\mathrm{d}h(q)}{\mathrm{d}q}, 0\right)\begin{pmatrix} 0 \\ D^{-1}(q)B \end{pmatrix} = 0. \tag{A.21}$$

Differentiating twice h, it yields

$$\ddot{h}(x) = \frac{\partial L_f h(x)}{\partial x}\dot{x} = L_f^2 h(x) + L_g L_f h(x)u. \tag{A.22}$$

Therefore, the dynamic equations of the virtual constraints is given by

$$\begin{pmatrix} \dot{y}_1 \\ \ddot{y}_2 \end{pmatrix} = \underbrace{\begin{pmatrix} L_f y_1 \\ L_f^2 y_2 \end{pmatrix}}_{\mathcal{L}_f} + \underbrace{\begin{pmatrix} L_g y_1 \\ L_g L_f y_2 \end{pmatrix}}_{\mathcal{A}_{f,g}} u, \tag{A.23}$$

where $\mathcal{A}_{f,g}$ is called the *decoupling matrix*. Feedback linearization consists of writing the controls u as function of a surrogate v

$$u = \mathcal{A}_{f,g}^{-1}(-\mathcal{L}_f + v), \tag{A.24}$$

where v must be chosen in a way to ensure $y = (y_1, y_2)$ converges to zero exponentially. A typical choice is

$$v = \begin{pmatrix} \epsilon y_1 \\ 2\epsilon L_f y_2 + \epsilon^2 y_2 \end{pmatrix} \tag{A.25}$$

where $\epsilon > 0$ is a free parameter. Under this control law, the dynamic of the virtual constraints is linear and has the form,

$$\begin{aligned} \dot{y}_1 &= -\epsilon y_1 \\ \ddot{y}_2 &= -2\epsilon \dot{y}_2 - \epsilon^2 y_2. \end{aligned} \tag{A.26}$$

It corresponds to critical damping, which is known to converge exponentially. The resulting feedback controller is illustrated in figure A.3.

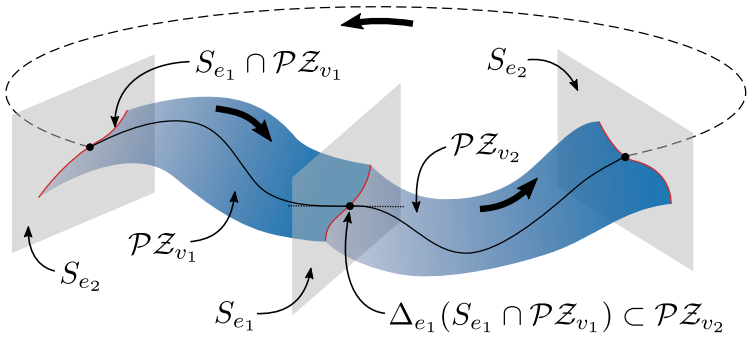


Figure A.2: Illustration of the PHZD periodic orbit for a two-domain hybrid system.

A.2.3 Hybrid Zero Dynamics

Each virtual constraint of relative degree 1 and 2 are locking respectively 1 or 2 DoFs of the system. The number of locked DoFs cannot exceed the total number of DoFs of the system. If they do not match exactly, it gives rise to a *zero dynamics*, which corresponds to the part of the system that is not controlled. The proposed feedback law renders the zero dynamics sub-manifold invariant in each continuous domain. Yet, it is not necessarily invariant through discrete dynamics. Enforcing impact invariance of the relative degree 1 output is too strong due to the velocity change at impact. Hence, invariance is enforced only for the relative degree 2 virtual constraints y_2, resulting in a so-called *partial zero dynamics* surface, given by

$$\mathcal{PZ} = \{(q, \dot{q}) \in \mathcal{TQ} \mid y_2 = 0, \dot{y}_2 = 0\}. \tag{A.27}$$

Moreover, the sub-manifold $\mathcal{PZ}$ called impact invariant, if there exist a set of parameters $\{v_d\}_{o\in\mathcal{O}_1}$ and $\{\alpha\}_{o\in\mathcal{O}_2}$, so that

$$\Delta x \in \mathcal{PZ}, \quad \forall x \in S \cap \mathcal{PZ}. \tag{A.28}$$

A manifold $\mathcal{PZ}$ is said to be *hybrid invariant* if it is invariant over the continuous dynamics and impact invariant through the discrete dynamics, namely, a solution that starts in $\mathcal{PZ}$ remains in $\mathcal{PZ}$, even after impulse effects. If a feedback control law renders $\mathcal{PZ}$ hybrid invariant, then the hybrid control system is said to have a *Partial HZD* (PHZD). The PHZD manifold is illustrated in figure A.2.

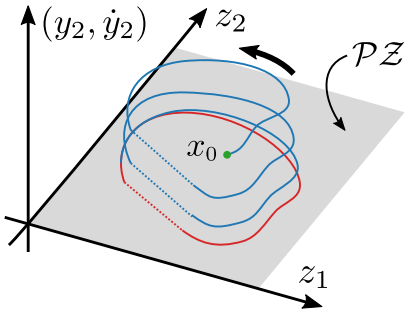


Figure A.3: Illustration of input-output feedback linearization for a one-domain hybrid system. The nominal trajectory is a periodic orbit. This initialized state x_0 is a bit off the nominal trajectory. The relative degree 2 outputs are non-zero at initialization but are driven exponentially to zero over time. The trajectory asymptotically convergences to a limit cycle that belongs to the partial zero dynamics surface $\mathcal{PZ}$.

A.3 Problem Transcription

Let us consider a generic *Optimal Control Problem* (OCP) for an autonomous system of the form $\dot{x} = f(x, u)$. It can be formulated as follows,

$$\begin{aligned} J = &\min_{x(0),u(t)} \int_0^T l(x(t), u(t))\,\mathrm{d}t \\ \text{st. } x(t) = &\int_0^t f(x(t), u(t))\,\mathrm{d}t \\ &\begin{cases} c_{in}(x(t), u(t)) \leq 0 \\ c_{eq}(x(t), u(t)) = 0 \end{cases}, \quad 0 \leq \forall t \leq T \end{aligned} \tag{A.29}$$

where l represents the running cost function, and c_{in}, c_{eq} are path inequality and equality constraints respectively. In our case, the objective is to find a parameterization of the virtual constraints for which the hybrid control system admits a PHZD while minimizing a given cost function.

There are three types of algorithms for solving optimal control problems (Anderson & Moore, 2007; Kalman, 1960; Kelly, 2015; Zhou, 1990): Dynamic Programming, indirect methods, and direct methods. Dynamic Programming consists in solving the Hamilton-Jacobi-Bellman equation associated with the problem over the entire state space. Indirect methods analytically derive the necessary and sufficient conditions for optimality by leveraging Pontryagin's maximum principle, then they discretize these conditions and solve them numerically. By contrast, direct methods discretize first the problem and then derive the conditions for optimality. Both indirect and direct methods yield a single trajectory through state and control space rather than a policy like Dynamic Programming. The conversion of an OCP in a NLP before passing it to a numerical solver is referred to as *problem transcription*. This process is common to indirect and direct methods.

DDP is the most effective *Dynamic Programming* algorithm (Jacobson & Mayne, 1970). DDP supports continuous state and action spaces and allows for fast convergence at a relatively low computational cost. However, it was originally limited to unconstrained problems. Very recently, Kazdadi et al. (2021) extended it to support general equality constraints, while Tassa et al. (2014) can take into account box control constraints. These methods are not mutually exclusive and can be combined, but they still lack versatility. Indirect methods tend to be numerically unstable and are difficult to implement and initialize (Kelly, 2015). Consequently, we restrict focus to direct methods for transcribing and solving the OCP, in particular the direct collocation method. This method scales well to high-dimensional systems and is fast even for offline trajectory planning.

A.3.1 Direct Collocation

The traditional approach to transcribing a trajectory optimization problem is the direct single shooting method. It involves integrating the dynamics via standard time-marching numerical methods, evaluating the constraint violations, and employing a NLP solver to drive them to zero. However, this method scales very poorly with the dimensionality of the problem, which is rapidly growing for multi-domain trajectory optimization. Direct multiple shooting methods based on reduced-dimensional hybrid zero dynamics can be used to improve reliability and speed, but such approaches also run into scalability issues with increasing degrees of freedom, due to the increased complexity of the equations describing the zero dynamics (Hereid et al., 2015). The Direct Collocation method works by replacing the explicit forward integration of the dynamical systems with a series of integration constraints (Hargraves & Paris, 1987; Kelly, 2017). More precisely, the system dynamics is expressed as basic implicit equations instead of complex close-form equations, and therefore the computational cost is not significantly affected by the dimensionality of the system dynamics. While the virtual constraints provide a formal guarantee of stability of the resulting trajectory, the Direct Collocation provides scalability for optimizing large-scale dynamical systems through discretization and approximation of states and inputs. This method has proven to be successful on DURUS (Hereid et al., 2016a) and Atalante (Gurriet et al., 2018).

The key concept of Direct Collocation method is to avoid numerical integration by approximating the continuous solution using a finite set of points. Then a piece-wise polynomial interpolation is used to infer the continuous solution from this discrete representation. Specifically, the time interval $t \in [0, T]$ is divided into a fixed number of distributed intervals. Since the discrete representation of the system must satisfy the differential equation of the system, the spline of the state must respect some condition to be a faithful approximation. Suppose a discrete representation with spline degree K defined as follows,

$$\tilde{x}_i(t) = \sum_{k=0}^{K} \left(a_{i,k} (t - t_i)^k \right), \tag{A.30}$$

where $\tilde{x}_i(t)$ denotes the polynomial approximation of the solution over a time interval $[t_i, t_{i+1}]$ and parametrized by coefficients $a_i = (a_{i,0}, a_{i,1}, \ldots, a_{i,K})$. The coefficients a_i must be chosen such that the approximation matches the solution at the beginning of the interval but not necessarily at the end, and that the derivatives match at K uniformly distributed points τ_j in the time interval $[t_i, t_{i+1}]$, namely,

$$\tilde{x}_i(t_i) = x(t_i) \tag{A.31}$$

$$\dot{\tilde{x}}_i(\tau_j) = f(\tilde{x}_i(\tau_j), \tilde{u}_i(\tau_j)) \tag{A.32}$$

$$\tau_j = t_i + \frac{j-1}{K-1}(t_{i+1} - t_i), \quad \forall 1 \leq j \leq K \tag{A.33}$$

These conditions are known as the *collocation conditions*, the intermediate points τ_j are called *interior nodes* or *collocation points*, and the points t_i are called *cardinal nodes*. No further distinction is made between interior nodes t_i and cardinal nodes τ_j for simplicity. If the coefficients a_i satisfy the collocation conditions, then the polynomials $\tilde{x}_i(t_i)$ are uniquely defined and yield to accurate approximated continuous solution over the time intervals $[t_i, t_{i+1}]$.

The numerical integration scheme used to approximate the state x is the Hermite-Simpson scheme, which is a 2-stage implicit Runge-Kutta scheme that relies on cubic interpolating polynomial. An implicit integration scheme is favored over an explicit one because it is generally more suitable for computing the solution of stiff differential equations such as periodic motions because their region of absolute stability is bigger. For the controls u, a simple trapezoidal scheme was used. It goes the same for polynomial inference $\tilde{x}^i$ and discrete representation x_i. Let assume the step is divided in $2N+1$ timesteps $t_0, t_1, \ldots, t_{2N}$. At each discrete node of $t = t_i$, an approximation of the state variable $x_i = x(t_i)$ and the command torques $u_i = u(t_i)$ is introduced as a set of optimization variables to be solved. The collocation condition specified by equation (A.32) gives

$$\dot{x}_i = f(x_i, u_i), \quad \forall i \in \{0, 2, \ldots, 2N\} \tag{A.34}$$

whereas the collocation condition specified by equation (A.31) becomes

$$x_{i+1} - x_{i-1} = \frac{1}{6}\Delta t_i(\dot{x}_{i-1} + 4\dot{x}_i + \dot{x}_{i+1}), \quad \forall i \in \{1, 3, \ldots, 2N-1\} \tag{A.35}$$

$$x_i = \frac{1}{2}(x_{i-1} + x_{i+1}) + \frac{\Delta t_i}{8}(\dot{x}_{i-1} - \dot{x}_{i+1}) \tag{A.36}$$

where equations (A.35) and (A.36) are obtained by integrating the state using the Simpson's quadrature rule and cubic polynomial inference respectively.

The distribution of cardinal nodes within a domain can be arbitrary, but the interior points have to be uniformly distributed between two adjacent cardinal nodes as a requirement of the integration scheme. The *Chebyshev-Gauss-Lobatto* (CGL) distribution is used to improve accuracy near the boundary conditions, namely,

$$\Delta t_i = \frac{T_f}{2}\left(\cos\left(\pi\frac{2\lfloor\frac{i+1}{2}\rfloor}{2N+1}\right) - \cos\left(\pi\frac{2\lfloor\frac{i+1}{2}\rfloor+1}{2N+1}\right)\right) \tag{A.37}$$

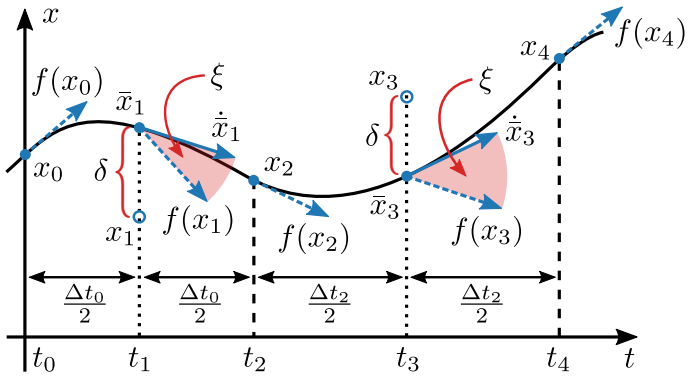


Figure A.4: Illustration of the Direct Collocation with Defect Constraints.

where T_f is the duration of the trajectory over the domain, and $\lfloor . \rfloor$ is floor operator. It is important to refine the spline approximation of the trajectory at the boundaries of the domains because the behavior of the robots tends to be more dynamics around contact phase changes. For instance, the accelerations undergoes a large jump before and after impact when switching support leg in order to transfer the center of pressure from one foot to the other.

A.3.2 Non-Linear Programming

Defect variables are optimization variables that could have been determined by closed-form functions, e.g. x_i and $\dot{x}_i$ at collocation points. Compute these variables explicitly is equivalent to impose constraints between defect variables, but it tends to be preferable to introduce defect variables on purpose. First, closed-form expressions are often expensive to compute, which is slowing down the optimization. For example, computing $\dot{x}_i$ explicitly requires inverting the inertia matrix H, whereas the equality constraint does not. Finally, it decouples the global optimization problem in many independent sub-problems, bound together through affine equality constraints to enforce consistency. Such formulation promotes sparsity and scalability, not to mention that it greatly simplifies the analytical derivation of the Jacobian of the optimization problem. These sub-problems can be tackled separately during the optimization process before consistency is enforced. Roughly speaking, it gives more freedom to the solver to find a valid solution since the equality constraints do not have to be valid all along the optimization process, as it would be the case using their closed-form counterparts. This property is even more beneficial that each constraint violation is driven to zero independently. For instance, it may be more appropriate to satisfy approximately constraints at position and velocity level, before finding an actual solution of the problem in the vicinity. The Direct Collocation method with defect constraints is illustrated in figure A.4.

This formulation significantly increases the number of constraints and optimization variables, leading to a large-scale nonlinear optimization problem. Yet, the Jacobian matrix is very sparse, and the density of the matrix is less than 1%. This

kind of NLP can be solved efficiently using a sparse NLP solver such as IPOPT (Hereid et al., 2016a). Note that such formulation enables to solve the optimization problem using the generalized *Alternating Direction Method of Multipliers* method, though it was not used here since direct solving is tractable.

With the discretization of a continuous domain, let

$$z_i = (T_{f,i}, q_i, \dot{q}_i, \ddot{q}_i, u_i, F_{ext,i}, \alpha_i, v_{d,i}) \tag{A.38}$$

be the vector of optimization variables defined at each node $i \in 0, 1, ..., 2N$, where $T_{f,i}$ is the total duration of the step, and $Z = (z_0, z_1, ..., z_{2N})$. Using this formulation, the collocation conditions (equations (A.35) and (A.36)) are computed using the defect variables instead of evaluating the system dynamics explicitly. The same goes for any other constraints, except the collocation condition specified by equation (A.34). In the latter, the evaluation of the constrained dynamics at each node is performed by the evaluation of the continuous dynamics and the holonomic constraints separately instead of using its closed-form expression. Note that it is unnecessary to explicitly introduce the current time of the node t_i because the current time is uniquely determined by the index of the node i and the step duration T_f through the CGL distribution. Anyway, the closed-loop dynamics is time-invariant because the motion is rather synchronized with a kinematic phase variable than the time.

Once a solution Z^* is found, only of few elements are extracted, namely, q_i^* and $\dot{q}_i^*$ at each discrete time t_i, which are then interpolated to match the high-level control period on the robot. It is irrelevant the extract the nominal torque u_i or the reference outputs parameters $v_{d,0}$ and α_0 since they are not used for control in practice.

A.3.3 Cost Function and Extra Constraints

So far, only the necessary constraints for the problem to be consistent and generate stable periodic motions have been addressed. Some additional constraints must be considered to enforce some desired gait features. In the particular case of flat foot walking in a straight line, the gaits features are:

- the duration of the step,
- the length of the step,
- the width of the step.

The question of the running cost $l(x, u)$ is often overlooked for robotics, for good reason since it usually barely affects the solution in practice. The direction of the optimization gradient is mostly determined by canceling constraint violation. The gradient of the running cost is only dominant at the last stage, where not much can be done to change the solution at this point given the large number of constraints that must be satisfied. In general, energy consumption is minimized in robotics. It is reasonable to make this choice since many robotic systems are powered by batteries. Moreover, the optimization marginally converges faster in practice because the gradient often heads in the direction of interesting solutions, though any sensible running cost should lead to similar performance. The formulation of power consumption depends on the hardware specification. For Atalante, the power generated by

each motor individually during braking ($u_i \cdot \dot{q}_i < 0$) cannot be used to power each other nor to recharge the batteries. Thus, it must be minimized just like the power consumed to facilitate the dissipation by the hardware of the superfluous energy in thermal losses. It yields the following approximate running cost at node i,

$$l_p(x_i, u_i) = \frac{\Delta t_i}{T_f} \|u_i \odot \dot{q}_i\|_2, \tag{A.39}$$

where $\odot$ denotes the element-wise product, also called *Hadamard product*. Note that L^2-norm is used in place of L^1-norm. The latter has a singularity at the origin, which is an issue when computing the analytical hessian. However, in practice, the hessian is not computed analytically by approximating numerically using the *Broyden-Fletcher-Goldfarb-Shannon* (BFGS) iterative algorithm. It is robust to this kind of singularity by smoothing it out, allowing the use of the L^1-norm formulation. Besides, if it is possible to make use of the generated energy to power other motors at the current time but not to charge the batteries, then the Hadamard product is replaced by the classical scalar product.

In addition, it may be beneficial to also maximize the extension of the knee of the stance leg. It is human-like, but more importantly, it reduces the variability of the trajectories. Assuming one of the position-modulating outputs corresponds to the relative knee angle, the extension can be approximately maximized by adding a cost on its reference y_k^d,

$$l_k(x_i, u_i) = \max_{s \in [0,1]} y_k^d(s, \alpha_i). \tag{A.40}$$

It is a continuous and differentiable function of the α_i which can be computed in a closed form by analytical differentiation of the Bézier polynomial y_k^d.

The cost function, which is defined as the integral of the running cost, is computed using Simpson's quadrature rule for irregularly spaced data. Let $L(x, u)$ be a function that needs to be integrated over the continuous domains. It can be stated as

$$\int_{t=0}^{T_f} L(x(t), u(t)) = \sum_{i=0}^{2N} \omega_i l(x_i, u_i), \tag{A.41}$$

where

$$\omega_i = \begin{cases} 1/3\Delta t_1, & \text{if } i = 0 \\ 2/3\Delta t_i, & \forall i \in \{1, 3, ..., 2N-1\} \\ 1/3(\Delta t_{i-1} + \Delta t_{i+1}), & \forall i \in \{2, 4, ..., 2N-2\} \\ 1/3\Delta t_{2N-1}, & \text{if } i = 2N \end{cases}.$$

B Proximal Splitting Method

B.1 Method of Multipliers and Quadratic Penalty

B.1.1 Generalities

Let us consider a general optimization problem

$$\begin{aligned} x^* &= \arg\min_{x \in \mathbb{R}^p} f(x) \\ \text{st. } & x \in \mathcal{X}, \quad h(x) = 0 \end{aligned} \tag{B.1}$$

where $\mathcal{X} \subseteq \mathbb{R}^p$ is a closed set and $f : \mathbb{R}^p \to \mathbb{R}, h : \mathbb{R}^p \to \mathbb{R}^q$ are continuous functions.

The Augmented Lagrangian function $L_\rho : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}$ is given by

$$L_\rho(x, \lambda) = f(x) + \lambda^T h(x) + \frac{\rho}{2} \|h(x)\|^2 , \tag{B.2}$$

where $\rho \in \mathbb{R}$ is the penalty factor and $\lambda \in \mathbb{R}$ is the lagrangian multiplier. $\|\cdot\|$ is the Euclidean norm, i.e. the L^2-norm. All the results presented here can be generalized with minor changes to any penalty function ϕ that is positive-definite in place of the Euclidean norm, i.e. $\phi(x) > 0, \forall x \neq 0$ and $\phi(x) = 0$, with the extra condition to have $\phi^{(2)}(0) = 1$ (Bertsekas, 1976).

The method of multipliers with quadratic penalty function consists in solving a sequence of problems of the form,

$$x^k = \arg\min_{x \in \mathcal{X}} L_{\rho^k}(x, \lambda^k). \tag{B.3}$$

The following assumption is central to the method of multipliers. It makes sure that the coupling constraint is gradually enforced, and satisfied at the limit:

Assumption 1. *The sequence of Lagrange multipliers* $\{\lambda^k\}_{k \geq 0}$ *is bounded in* $\mathbb{R}$, *namely* $\forall k \geq 0, \exists M \in \mathbb{R}$ *st.* $\left\|\lambda^k\right\| < M$, *and the penalty cost satisfies*

$$\forall k \geq 0, \quad 0 < \rho^k \leq \rho^{k+1}, \quad \rho^k \to \infty \tag{B.4}$$

Intuitively, solving problem (B.3) under assumption 1 is the same as solving problem (B.1) at the limit. Rigorously, additional conditions must hold true. Bertsekas (1982) has proven the subsequent theorem:

Theorem 6. *If assumption 1 is satisfied and* $\forall k \geq 0, x^k$ *is a global optimum of problem (B.3), then the limit point of the sequence* $\{x^k\}_{k\geq 0}$ *is a global optimum of problem (B.1).*

Problem (B.3) may not have a global optimum even if problem (B.1) admits one. Still, it is sufficient to have $\mathcal{X}$ compact (as opposed to closed) to prove that problem (B.3) as a global minimum by the Weierstrass Extreme Value Theorem. Another option is to alter f, g to make them positive, i.e. $\forall x \in \mathbb{R}^p, f(x) > 0$ and $g(x) > 0$. Yet, having to converge to a global optimum is very restrictive. This condition cannot be guaranteed beyond *Quadratic Program* (QP) on $\mathcal{X} = \mathbb{R}^p$ under linear equality constraints. It can be relaxed in favor of isolated local optima, which is more attainable. For a given optimization problem, an isolated set of local optima X^* is defined as any non-empty subset of $\mathbb{R}^p$ st. any point in X^* is a local optimum of the problem, and the set $X^*_\epsilon = \{x \in \mathbb{R}^p \mid \|x - x^*\| \leq \epsilon \text{ for some } x^* \in X^*\}$ contains no local optima of the problem. Bertsekas (1982) has demonstrated that:

Theorem 7. *If assumption 1 is satisfied and* X^* *is an isolated set of local optima of problem (B.1) that is compact, then there exists a subsequence* $\{x^k\}_K$ *converging to a point* $x^* \in X^*$ *st.* $\forall k \in K$, x^k *is a local minimum of problem (B.3), where* K *is an infinite subset of positive integers. Furthermore, if* X^* *is a single point* $\{x^*\}$, *there exists a sequence* $\{x^k\}_{k\geq 0}$ *and an integer* $\bar{k} \geq 0$ *st.* $x^k \to x^*$ *and* x^k *is a local minimum of problem (B.3) for all* $k \geq \bar{k}$.

The aforementioned theorem has limited practical interest. The whole point of proximal splitting methods is enabling finding a solution of problem (B.1) without solving it directly, thus its local optima always are unknown. The following corollary is more relevant:

Corollary 7.1. *If assumption 1 is satisfied, and a subsequence* $\{x^k\}_K$, *where* K *is an infinite subset of positive integers, converges to a point* $\bar{x} \in \mathbb{R}^p$ *such that,* $\forall k \in K$, x^k *is a local minimum of problem (B.3), then there exist a local optima of problem (B.1)* x^* *such that* $\bar{x} = x^*$.

Proof. The proof is straightforward. The Augmented Lagrangian $L_\rho(x, \lambda)$ is continuous with respect to ρ, $\{\lambda^k\}_{k\geq 0}$ is bounded, and $\rho^k \to \infty$. Hence, $L_{\rho^k}(x, \lambda^k) \to \tilde{f}(x)$, where $\tilde{f} : \mathbb{R}^p \to (-\infty, \infty]$ is defined by

$$\tilde{f} = \begin{cases} f(x), & \text{if } h(x) = 0 \\ \infty, & \text{if } h(x) \neq 0 \end{cases} \tag{B.5}$$

Consequently, if x^k is a local minimum of problem (B.3) for $\forall k \in K$, and K is an infinite subset of positive integers, then the subsequence $\{x^k\}_K$ converges to a local minimum of $\tilde{f}$, and therefore a local minimum of problem (B.1). Roughly speaking, taking the limit to infinity (whenever the limit exists) is equivalent to converging with respect to K as the latter is an infinite subset. ■

The ill-conditioning of problem (B.3) is characteristic of penalty methods because the cost function is very steep is the penalty factor ρ_k is large and the equality constraints $h(x)$ are violated. It can be mitigated by warm-starting the optimization close enough to a local minimum of problem (B.3) at each iteration, which is doable as long as the increase of the penalty factor ρ_k remains small. There is no theoretical method to find the fastest sequence $\{\rho^k\}_{k\geq 0}$. Bertsekas (1982) states that $\rho^{k+1} = \beta\rho^k$ with $\beta \in [4, 10]$ works well in general.

B.1.2 Multipliers update

The sequence of Lagrange multipliers $\{\lambda^k\}$ can be chosen freely. They are all set to zero in the original method of multipliers, but updating them appropriately can dramatically improve the convergence rate and relax the requirement to have $\rho^k \to \infty$ to converge. The proposal for the update rule of the multipliers is

$$\lambda^{k+1} = \lambda^k + \rho^k h(x^k) \tag{B.6}$$

Formal results and a geometric interpretation in two dimensions for $f, h \in C^2$ on $\mathcal{X} = \mathbb{R}^p$ is explained in details by Bertsekas (1982). A brief summary is presented here to understand why this choice is natural, alleviate the difficulties due to ill-conditioning of (B.3), and relax the requirement to have $\rho^k \to \infty$ to converge. It assumes that $f, g \in C^2$, which is not very restrictive in practice. Nonetheless, this hypothesis can be alleviated if necessary (Bertsekas, 1976).

A vector x st. $h(x) = 0$ is said to be a regular point if the gradients of the constraints $\nabla h_1(x), \nabla h_2(x), \ldots, \nabla h_q(x)$ are linearly independent. If x^* is an isolated local minimum for problem (B.1) that is also a regular point, then it satisfies the standard second-order sufficiency conditions to be an isolated local minimum for problem (B.3). That is to say, the hessian of the Augmented Lagrangian L_ρ is positive definite on the tangent space $T_x^* D$, where D is the surface associated with the equality constraints g. More precisely, there exists a unique vector $\lambda^* \in \mathbb{R}^q$ st.

$$\nabla_x L_\rho(x^*, \lambda^*) = 0 \tag{B.7}$$

$$z^T H_x(x^*, \lambda^*) z > 0, \forall z \in \mathbb{R}^p \setminus \{0\} \text{ with } \nabla h(x^*) z = 0 \tag{B.8}$$

where $H_x(x^*, \lambda^*) = \nabla_x \nabla_x L_\rho(x, \lambda)|_{(x,\lambda)=(x^*,\lambda^*)}$ is the Hessian matrix of L_ρ with respect to x at point (x^*, λ^*).

Let us assume on the contrary that it exists a point (x^*, λ^*) for which the second order conditions are met:

Assumption 2. *Let x^* be st. $h(x^*) = 0$. Assume that there exist a vector $\lambda^* \in \mathbb{R}^q$ st. $\nabla_x L_\rho(x^*, \lambda^*) = 0$ and $z^T H_x(x^*, \lambda^*) z > 0, \forall z \in \mathbb{R}^p \setminus \{0\}$ with $\nabla h(x^*) z = 0$.*

If assumption 2 is satisfied, then x^* is an isolated local minimum for (B.1). Moreover, there exists $x(\cdot), \lambda(\cdot)$ continuously differentiable functions such that $x(0) =$

$x^*, \lambda(0) = \lambda^*$, and $\forall u \in \mathbb{R}$, $(x(u), \lambda(u))$ is a local minimum for

$$\begin{aligned} x^* = &\underset{x \in \mathbb{R}^p}{\operatorname{argmin}} f(x) \\ &\text{st. } h(x) = u \end{aligned} \tag{B.9}$$

Furthermore,

$$\nabla_u f(x(u)) = -\lambda(u), \quad \forall u \in \mathbb{R}. \tag{B.10}$$

Consider the functional p defined as

$$p(u) = \min_{h(x)=u} f(x), \tag{B.11}$$

where the minimization is understood to be local in an open sphere within which x^* is the unique local minimum of (B.1). From the previous proposition, it gives $\nabla p(0) = -\lambda^*$ since $p(0)$ equals $f(x^*)$. The minimization of $L_\rho(x, \lambda)$ with respect to x can be broken down into two stages, first minimizing for x such that $h(x) = u$ for a given u, and then minimizing for u. This yields

$$\min_x L_\rho(x, \lambda) = \min_u \min_{h(x)=u} \left(f(x) + \lambda^T h(x) + \frac{\rho}{2} \|h(x)\|^2 \right) \tag{B.12}$$

$$= \min_u \left(p(u) + \lambda^T u + \frac{\rho}{2} \|u\|^2 \right) \tag{B.13}$$

where the minimization is understood to be local in an open sphere within which $u = 0$. The minimum is attained at the point $u(\lambda, \rho)$ for which the gradient of $p(u) + \lambda^T u + \frac{\rho}{2} \|u\|^2$ vanishes, or equivalently

$$\nabla_u \left(p(u) + \frac{\rho}{2} \|u\|^2 \right) |_{u=u(\lambda,\rho)} = -\lambda. \tag{B.14}$$

In addition, it gives

$$\min_x L_\rho(x, \lambda) - \lambda^T u(\lambda, \rho) = p(u(\lambda, \rho)) + \frac{\rho}{2} \|u(\lambda, \rho)\|^2 \tag{B.15}$$

so the tangent hyperplane to $p(u) + \frac{\rho}{2} \|u\|^2$ at $u(\lambda, \rho)$ intersects the vertical axis at the value $\min_x L_\rho(x, \lambda)$ as shown in figure B.1. It can be seen that if ρ is large enough then $p(u) + \lambda^T u + \frac{\rho}{2} \|u\|^2$ is convex in the neighborhood of the origin. Furthermore, $\min_x L_\rho(x, \lambda)$ is close to $p(0) = f(x^*)$ for values of λ close to λ^* and large values of ρ.

Figure B.1 depicts a geometric interpretation of the multipliers update iterations according to equation (B.6). Note that if x^k minimizes $L_{\rho^k}(x, \lambda^k)$ with respect to x, then the vector u^k given by $u^k = h(x^k)$ minimizes $p(u) + {\lambda^k}^T u + \frac{\rho^k}{2} \|u\|^2$. Hence,

$$\nabla \left(p(u) + \frac{\rho^k}{2} \|u\|^2 \right) |_{u=u^k} = -\lambda^k \tag{B.16}$$

and

$$\nabla p(u^k) = -(\lambda^k + \rho^k u^k) = -(\lambda^k + \rho^k h(x^k)). \tag{B.17}$$

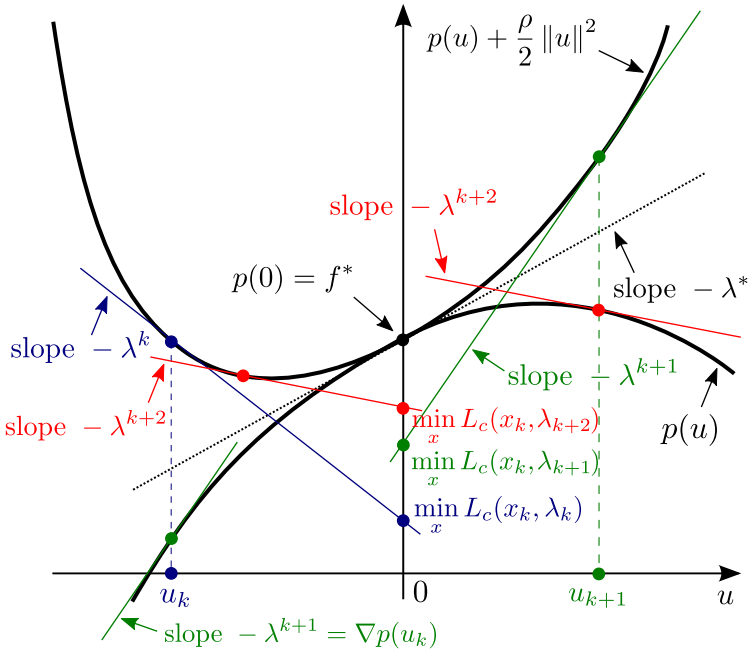


Figure B.1: Geometric interpretation of the first-order multiplier iteration

It follows that the multipliers at the next iteration λ^{k+1} are given by

$$\lambda^{k+1} = \lambda^k + \rho^k h(x^k) = -\nabla p(u^k). \tag{B.18}$$

The figure makes it clear that if λ^k is sufficiently close to λ^* and/or ρ^k is sufficiently large, the next multiplier λ^{k+1} will be closer to λ^* than λ^k is. In addition, the convergence is the fastest if $\nabla p(0) = 0$. It is not necessary to have $\rho^k \to \infty$ in order to converge, but only that ρ^k exceeds some threshold after a certain index.

B.1.3 Convergence in the non-convex case

Results about the convergence rate in the non-convex case problem (B.1) was first exposed by (Bertsekas, 1976). A summary of the most important ones is given here.

Under assumption 2, for any given λ bounded subset of $\mathbb{R}^q$, there exist a unique scalar $\rho^* \geq 0$ such that for every $\rho > \rho^*$ and every $\lambda \in \Lambda$, the function $L_\rho(x, \rho)$ has a unique minimizing point $x(\lambda, c)$ with some open sphere centered at x^*, where x^* is a local minimum of problem (B.1). Furthermore, for some scalar $M > 0$ it yields

$$\|x(\lambda, \rho) - x^*\| \leq \frac{M\,\|\lambda - \lambda^*\|}{\rho}, \quad \forall \rho > \rho^*, \lambda \in \Lambda \tag{B.19}$$

$$\|\tilde{\lambda}(\lambda, \rho) - \lambda^*\| \leq \frac{M\,\|\lambda - \lambda^*\|}{\rho}, \quad \forall \rho > \rho^*, \lambda \in \Lambda \tag{B.20}$$

where the vector $\tilde{\lambda}(\lambda, c) \in \mathbb{R}^q$ is given by $\tilde{\lambda}(\lambda, c) = \lambda + \rho\, h(x(\lambda, c))$.

This proposition induces both sufficient conditions for the convergence and the rate of convergence of the method of multipliers when they are updated via the update rule in equation (B.6). It constitutes the strongest convergence result available for problem (B.1). It shows that if Λ contains λ^* in its interior, the generated sequence $\{\lambda^k\}$ remains in Λ, the penalty parameter ρ is sufficiently large after a certain index, and the minimization of $L_{\rho=k}(x, \lambda^k)$ yields the local minimum $x(\lambda^k, \rho^k)$ which is closest to λ^*, then we obtain $x(\lambda^k, \rho^k) \to x^*, y_k \to \lambda^*$.

Theorem 8. *Consider a non-decreasing positive penalty parameter sequence ρ^k such that for some integer $\bar{k} \geq 0$ we have $\rho^{\bar{k}} \geq \max(M, \rho^*)$. Let $\{\lambda^k\}$ be a sequence such that $y_{\bar{k}} \in \Lambda$ and generated for all $k > \bar{k}$ by the update rule (B.6), where x^k is the local minimizing point of $\min_x L_\rho(x, \lambda^k)$ closest to x^* in the sense of (B.19). Then $x(\lambda^k, \rho^k) \to x^*, y_k \to \lambda^*$. Furthermore, if $\rho^k \to \bar{\rho} < \infty$ and $\lambda^k \neq \lambda^*, \forall k$, then*

$$\lim_{k\to\infty} \sup \frac{\left\|\lambda^{k+1} - \lambda^*\right\|}{\|\lambda^k - \lambda^*\|} \leq \frac{M}{\bar{\rho}} \text{ (linear convergence)} \tag{B.21}$$

while if $\rho^k \to \infty$,

$$\frac{\left\|\lambda^{k+1} - \lambda^*\right\|}{\|\lambda^k - \lambda^*\|} \to 0 \text{ (superlinear convergence)} \tag{B.22}$$

It follows that it is not necessary to increase ρ^k to infinity for the method of multipliers to converge. Therefore, the ill-conditioning effects associated with large penalty parameters can be eliminated or at least moderated in multiplier methods.

B.2 Alternating Direction Method of Multipliers

Let's consider a general optimization problem as follows,

$$\begin{aligned} (x^*, z^*) = &\underset{x\in\mathbb{R}^{p_1}, z\in\mathbb{R}^{p_2}}{\operatorname{argmin}} f(x) + g(z), \\ &\text{st. } x \in \mathcal{X}, z \in \mathcal{Z} \\ &Ax + Bz = c \end{aligned} \tag{B.23}$$

where $\mathcal{X}, \mathcal{Z}$ are closed subsets of $\mathbb{R}^{p_1}$ and $\mathbb{R}^{p_2}$ respectively, $A \in \mathbb{R}^{q\times p_1}$, $B \in \mathbb{R}^{q\times p_2}$, and $c \in \mathbb{R}^q$, and the functions $f : \mathbb{R}^{p_1} \to \mathbb{R}$ and $g : \mathbb{R}^{p_2} \to \mathbb{R}$ are continuously differentiable on $\mathbb{R}^{q\times p_1}$ and $\mathbb{R}^{q\times p_2}$ respectively. The affine constraint $Ax + Bz = c$ is referred to as the coupling constraint.

Problem (B.23) is general in the sense that many interesting large-scale problems, including consensus. Such non-convex problems can be handled gracefully by the penalty and augmented Lagrangian method presented previously. In this case, the augmented Lagrangian is given by

$$L_\rho(x, z, \lambda) = f(x) + g(z) + \lambda^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|^2 \tag{B.24}$$

where x and z are the primal variables of problem (B.23).

The primal variable update is given by

$$(x^{k+1}, z^{k+1}) = \underset{x \in \mathcal{X}, z \in \mathcal{Z}}{\operatorname{argmin}} L_{\rho^k}(x, z, \lambda^k) \tag{B.25}$$

where x^{k+1}, z^{k+1} are the optimal value of x, z at iteration $k+1$ for a given λ^k. Notice that x, z are initialized with the previously computed x^k, z^k.

The challenge of the method of multipliers is to choose appropriately the sequence of multipliers $\{\lambda^i\}$ to converge as fast as possible without needing $\rho^i \to \infty$. Based on the results of the previous section, an appropriate choice is to update the multipliers according to the following recursion,

$$\lambda^{k+1} = \lambda^k + \rho^k (Ax^k + Bz^k - c). \tag{B.26}$$

C Jiminy – Open-source Simulator for Legged Robots

C.1 Modelling of the Mechanical Deformation

The structure of poly-articulated robots appears flexible due to the continuous deformation of all their mechanical parts, including bodies and transmissions. The *Finite Element Method* (FEM) is generally used to model this phenomenon. First, each part is decomposed in a large set of interconnected polyhedra (commonly tetrahedra), each of them having known physical properties that only depend on the manufacturing material, e.g. density, Young's modulus, shear modulus, and Poisson's ratio. Then, the deformation in static equilibrium at the macroscopic level is deduced from the interaction between all these tiny elements given external forces. This technique is very accurate and well-motived physically. For instance, it is used to study soft and deformable materials. However, it is limited to static equilibrium and the computations are very demanding. Thus, this approach is not appropriate in robotics.

We suggest relying nonetheless on a simpler approximate model of the deformation in simulation, as it already brings a significant improvement over the classic rigid model while being much cheaper than FEM. Two different methods are found in the literature (Vigne et al., 2020a; Vigne et al., 2019):

- assuming that the robot is fully-actuated and all its joints are 1-*Degree of Freedom* (DoF), a virtual flexible element is added between the motors and their corresponding joints,
- a body followed by a spherical joint are inserted into the kinematic tree at key locations to fit the overall deformation.

The second approach is way more generic and powerful. In particular, any system is supported without restriction, and any number of deformation points located at joints or inside bodies can be considered. Notably, it gets closer to the FEM as the number of deformation points increases. It has the additional benefit to be supported out-of-the-box in simulation, whereas the first one is not because it requires keeping track of some hidden internal state at the motor level. The only advantage of the first approach is being very convenient for designing model-based observers and controllers. We restrict focus to the second approach in the following since we are more interested in faithful simulation for training robust control policies.

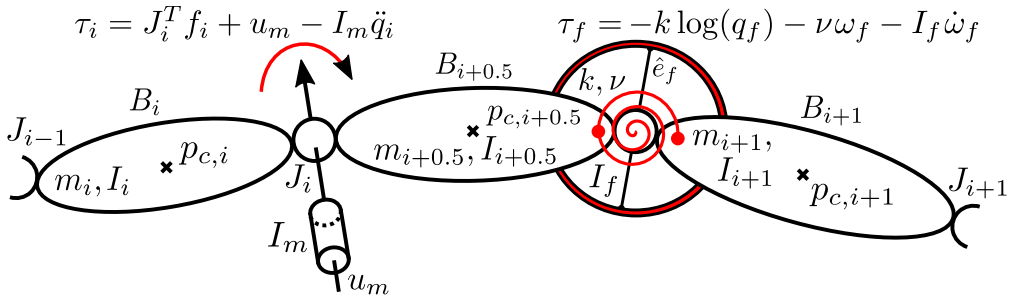


Figure C.1: Modelling of the mechanical deformation in simulation. For each local deformation point, a body followed by a spherical joint J_f are inserted into the kinematic tree. The internal dynamics of the flexible joint corresponds to a spherical mass-spring-damper mechanism with inertia I_f, stiffness k, and damping ν.

For most legged robots, adding deformation points located at the joints is more than enough to accurately fit the actual deformation. More specifically, it is sufficient to add deformation points at the start and end of each sub-chains, namely the hips and ankles for bipedal robots. The resulting model of the system is referred to as *flexible model*, in contrast to the rigid model. The current configuration of the flexible model minimizes the MSE between the true orientation of all the bodies and the theoretical one computed by *Forward Kinematics* (FK). It is mathematically equivalent to projection operation, so it is extremely fast to evaluate. Vigne et al. (2019) validated that deformation points at the sagittal hip joint and the sagittal ankle joint of each leg can explain the overall deformation for the exoskeleton Atalante, using motion capture on nominal motions as ground truth. Once embedded in the real robot, the orientation of the bodies is estimated using IMU sensors, assuming at least one foot is flat on the ground and the contact is stable (Vigne et al., 2020b).

The mass $m_{i+0.5}$ and inertia $I_{i+0.5}$ of the intermediate body $B_{i+0.5}$ depends on the location of the deformation point. If the latter is inside a body, then the mass and inertia of this body are distributed between the original and extra bodies at the user's discretion. Similarly, an intermediate body is still added even if the deformation point is located at a joint, but it is fictitious in this case. More precisely, it is massless and dimensionless, so that the flexible joint is coincident with the true one, i.e. ${}^i p_f = 0, {}^i R_f = I_3$. This is necessary because two joints cannot be connected together in a kinematic tree. By convention, the intermediate body is placed after the original joint in the latter case. As expected, the flexible and rigid configurations match exactly for $\theta_f = 0$ whether the location of the deformation point.

The internal dynamics of the flexible joints are modelled as spherical mass-spring-damper mechanisms (see figure C.1),

$$I_f \dot{\omega}_f = -k \log(q_f) - \nu \omega_f - \tau_f \tag{C.1}$$

where q_f is the orientation of the flexible joint as a quaternion, log is the inverse exponential map on $SO(3)$ mapping quaternions to their axis-angle representation (see equation (2.13)), ω_f is the angular velocity of the joint, τ_f is the torque apply

by its parent body B_f, I_a is a positive diagonal inertia matrix, and k, ν are positive semi-definite matrices representing the stiffness and damping of the deformation respectively. If the extra intermediate body is massless, then $\tau_i = \tau_f$, where τ_f is the force apply by the virtual joint J_f on the intermediate body $B_{i+0.5}$.

The inertia I_f of the flexible joint is implemented as armature inertia in rigid body algorithms to avoid altering the inertia of the whole system. This way, it affects the dynamics of the flexible joint in its local frame but does not propagate downward in the sub-chain composite inertia. Mathematically, armature inertia is a block-diagonal matrix added directly to the mass matrix of the system $H(q)$. If the armature inertia I_f is infinite, then $\dot{\omega}_f = 0$ and the dynamics of the flexible model is the same as the rigid one. On the contrary, if $I_f = 0$ then the numerical integration of the dynamics of the flexible joint is unstable if the intermediary body $B_{i+0.5}$ is massless. In theory, it should be valid regardless, but it reduces the order of the dynamics from two to one: the analytical velocity would change discontinuously if $\nu > 0$, the analytical position otherwise.

The inertia I_f is not well-motived physically. It is rather introduced to smooth out the dynamics and avoid discontinuity in acceleration, which is essential to allow for a large integration time step during simulation. The control period is about 10ms for policy learning applications on legged robots, so ideally the integration step must be as close as possible to this value to maximize the simulation speed and hence reduce the training time. For Atalante, it corresponds to an armature inertia of about $1.0\text{kg}\,\text{m}^2$ on each axis. In comparison, the integration step must be around 10us to avoid numerical instability if $I_f = 0$.

C.2 Ground Contact Interaction

C.2.1 Enforcing Bilateral Kinematic Constraints

Bilateral kinematic constraints are equality constraints involving the current state of the system and time in general. They are ubiquitous in poly-articulated robots. For instance, to model closed kinematic chains or complex transmissions such as mirrored joint or differential gear. In all but a few cases, it can be formulated as holonomic constraints $f_c(q) = 0$, so we restrict our analysis to this particular kind of kinematic constraint in the following. Let P be a frame attached to a body B, and O is a fixed point in space. The constraint $f_c(q) = 0$ resulting from fixing frame P at O is simply the error between the current pose (p_P, R_P) of frame P and the target pose (p_O, R_O) of point O, expressed in an arbitrary reference frame with origin P. It gives,

$$f_c(q) = \begin{pmatrix} p_P - p_O \\ \bar{R}^T \log\left(R_P R_O^T\right) \end{pmatrix} \qquad J_c(q) = J_P \tag{C.2}$$

where $\bar{R}$ is the rotation matrix representing the orientation of the reference frame in space, J_P is the jacobian of the position and orientation of the body B at point P in the reference frame, and log is the inverse exponential map on $SO(3)$ mapping

rotation matrices to their axis-angle representation. As a reminder, the reference frame is always considered fixed even if moving.

This objective is to take into account these constraints when integrating the dynamics of the system. More precisely, we are looking for the acceleration $\ddot{q}$ of the system based on the current position, velocity, external forces, and constraints. The classical forward dynamics algorithm does not handle constraints, and the *Gauss's principle of least constraint* must be used instead. It states that the acceleration of the constrained system is as close as possible to that of the corresponding unconstrained system in a least-squares sense. It was originally formulated for point mass particles, but it is straightforward to generalize for poly-articulated systems by considering the inertia matrix as opposed to the mass of each body. Formally, the subsequent optimization problem must be solved:

$$\ddot{q}^* = \arg\min_{\ddot{q}} \frac{1}{2}\|\ddot{q} - \ddot{q}_{\text{free}}\|_H^2 \tag{C.3}$$

$$\text{s.t. } f_c(q) = 0 \tag{C.4}$$

where H is the inertia matrix of the system, $\|x\|_H = \sqrt{x^T H x}$ is the kinetic metric and $\ddot{q}_{\text{free}}$ is the solution of the forward dynamics without constraints $\ddot{q}_{\text{free}} = H^{-1}(\tau - C(q,\dot{q})\dot{q} - G(q))$. The metric $\|x\|_H$ is a valid norm since the matrix H is positive semi-definite by property of the kinematic energy $1/2\dot{q}^T H \dot{q} \geq 0$. Using this metric, joints having a limited effect on the total kinematic energy of the system have a limited effect on the norm and conversely. The constraint $f_c(q) = 0$ implicitly depends on the acceleration $\ddot{q}$. This dependency must be made explicit by differentiating twice equation (C.4) to be able to solve the optimization problem,

$$a_c = J_c(q)\ddot{q} + \gamma_c = \bar{a}_c \tag{C.5}$$

where $J_c(q)$ is the jacobian of the constraint, $\gamma_c = \dot{J}_c(q,\dot{q})\dot{q}$ is called *drift*, and $\bar{a}_c$ is the target constraint acceleration. γ_c is usually obtained by computing a_c for $\ddot{q} = 0$ using forward kinematics. The target acceleration $\bar{a}_c$ serves as correction term to prevent the constraint $f_c(q) = 0$ from drifting. Without correction, it is only sufficient if the constraint is already exactly satisfied at both the position and velocity levels, which may not be the case. Therefore, a correction term is introduced to reduce the constraint violation over time at the position and velocity levels if necessary. Physically, this approach is questionable, but it is convenient to allow large integration timestep without diverging because of compounding errors during integration. A classical correction is the Baumgarte stabilization (Flores et al., 2011),

$$\bar{a}_c = -k\, f_c(q) - \nu\, v_c(q,\dot{q}) \tag{C.6}$$

where $v_c = J_c(q)\dot{q}$ is the velocity of the constraint and k, ν are free parameters. Loosely speaking, it can be viewed as a kinematic spring-damper mechanism whose stiffness and damping parameters are k, ν respectively. In practice, those parameters are chosen to get critical damping, which means that the constraint violation is

exponentially reducing over time. It yields $k = \omega^2, \nu = 2\omega$ where $\omega = 2\pi/\Delta T$ is the pulsation and ΔT is the time constant.

The optimization problem (C.3) can be solved using the *method of Lagrange multipliers* after replacing the original constraint by its regularization at the acceleration level equation (C.5). The associated Lagrangian is

$$L(\ddot{q}, \lambda) = \frac{1}{2}\|\ddot{q} - \ddot{q}_{\text{free}}\|_H^2 + \lambda^T(a_c - \bar{a}_c) \tag{C.7}$$

where λ is the vector of lagrangian multipliers. The solution to the original problem is a saddle node of its Lagrangian,

$$(\ddot{q}^*, \lambda^*) = \arg\min_{\ddot{q}} \arg\max_{\lambda} L(\ddot{q}, \lambda). \tag{C.8}$$

The optimality conditions, also called *Karush-Kuhn-Tucker* (KKT) (KKT) conditions, can be formulated as a linear system (Carpentier et al., 2021)

$$\begin{pmatrix} H & J^T \\ J & 0 \end{pmatrix} \begin{pmatrix} \ddot{q} \\ \lambda \end{pmatrix} = \begin{pmatrix} H\ddot{q}_{\text{free}} \\ -\gamma_c + \bar{a}_c \end{pmatrix}, \tag{C.9}$$

which can be inverted in a closed-form. It yields,

$$\ddot{q} = \ddot{q}_{\text{free}} - H^{-1}J_c^T\lambda \tag{C.10}$$

$$a_c = -\underbrace{(J_c H^{-1} J_c^T)}_{A}\lambda + a_{c,\text{free}} = \bar{a}_c \tag{C.11}$$

where A is the inverse *Operational Space Inertia Matrix* (OSIM) or *Delassus matrix* (Khatib, 1987), λ are the lagrangian multipliers and $a_{c,\text{free}} = J_c(q)\ddot{q}_{\text{free}} + \gamma_c$ is the acceleration of the constraint if not enforced. It has been mentioned in preliminaries that a spatial force $\hat{f} = (f, \tau_P)$ associated with the bilateral constraint $f_c(q) = 0$ can be expressed in joint space as $\tau = J_P^T f$ because of the duality condition between spatial motion and force spaces. Consequently, the Lagrange multiplier λ can be interpreted as the spatial force vector $\hat{f}$ applied at point P on body B in the reference frame. Similarly, the velocity v_c and acceleration a_c of the constraint are the spatial velocity v_P and acceleration a_P of the contact point P in the reference frame.

Carpentier et al. (2021) have shown that equation (C.11) admit a unique solution if and only if A is strictly positive definite, which can be obtained efficiently using *sparse Cholesky decomposition*. However, in most cases, this condition does not hold. Indeed, the OSIM matrix A is only guaranteed to be positive semi-definite. A is rank deficient if some constraints are redundant figure C.2. In such as case, there is an infinite number of solutions and Cholesky decomposition cannot be used anymore. Several approaches can be considered to circumvent this limitation. Carpentier et al. solves it iteratively using a proximal algorithm, which converges to a solution of the original problem at a linear rate. Alternatively, L^2-norm penalty term can be added to the Lagrangian, so-called Tikhonov regularization.

$$\bar{L}(\ddot{q}, \lambda) = L(\ddot{q}, \lambda) - \frac{1}{2}\|\lambda\|_\Gamma^2, \tag{C.12}$$

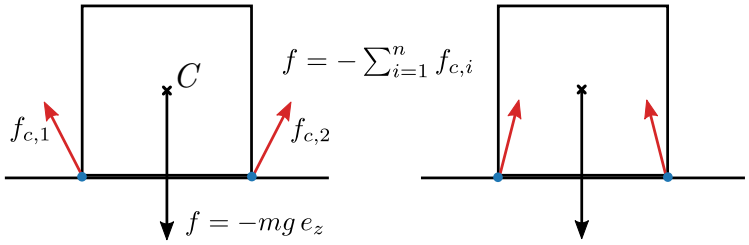


Figure C.2: Non-uniqueness of the solution in case of constraint redundancy. The box is in static equilibrium if and only if the spatial force applied at the CoM C by the n contact constraints $f_{c,i}$ cancels the weight exactly. There is an infinity of solutions to this problem for $n > 1$. Two different solutions are illustrated.

where Γ must be strictly positive definite. It achieves a trade-off between solving the original problem and minimizing the norm of the multipliers, so it does not converge to an exact solution anymore. However, it helps to get smoother forces, by preventing jumping from one to another if it is not unique, whatever the initial guess. It is also faster to compute than proximal methods, and the effect of the regularization is very limited when Γ is properly tuned. It is the preferred approach in Jiminy (Duburcq, 2019). A common choice $\Gamma = \gamma\,\mathrm{diag}(A)$, where diag operator extracts the diagonal of a matrix as a diagonal matrix, and γ is scalar called damping parameter around 1e−3. This relative weighting of the penalty terms is important to be robust to the unbalanced mass distribution along the kinematic tree, and thereby consistently provides a good approximation of the optimal solution. Using this approach, equation (C.11) must be slightly revised. It follows,

$$\underbrace{(J_c H^{-1} J_c^T + \Gamma)}_{A_\gamma} \lambda - \underbrace{(a_{c,\mathrm{free}} - \bar{a}_c)}_{b} = 0. \tag{C.13}$$

where A_γ is called the damped inverse OSIM.

C.2.2 Contact as Unilateral Constraints

Physically, contact constraints are not bilateral but rather unilateral, and the ground reaction force is unbounded. More precisely, the robot can slip on the ground and no torque can be applied by individual contact points figure C.3. According to Coulomb friction law (Moreau, 1988),

$$0 \leq v_n \perp f_n \geq 0 \quad \text{No penetration \& Non-adhesive ground} \tag{C.14}$$

$$0 \leq \|v_t\|_2 \perp \mu f_n - \|f_t\|_2 \geq 0 \quad \text{Coulomb friction cone} \tag{C.15}$$

$$v_t \cdot f_t = -\|v_t\|_2 \|f_t\|_2 \quad \text{Maximum energy dissipation} \tag{C.16}$$

$$\tau_P = 0 \quad \text{No torque} \tag{C.17}$$

where $v \perp w$ is the complementarity condition $v \cdot w = 0$, v_t, f_t are the linear velocity and force at the contact point in the plane orthogonal to e_n axis respectively, v_n, f_n is

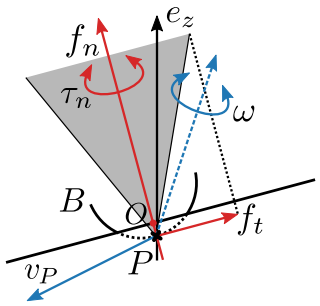


Figure C.3: Ground reaction force at contact point

the linear velocity and force along e_n axis respectively, and μ is the friction coefficient. These conditions must hold during the whole motion. The equality equation (C.17) is equivalent to masking the lines in the jacobian J_P associated with torque coordinates. In this way, the Lagrange multiplier only gathers the linear force vectors at all contact points. This simplification is done in the following for conciseness.

Contact constrained can be formulated as fixed frame constrained whose associated force must satisfy the aforementioned complementarity conditions. Equation (C.11) is only valid for bilateral constraints, and it has no longer guaranteed to hold for contact constraints because of the additional requirements. Instead, it must be replaced by a minimization problem, or equivalently its dual *Non-Linear Complementarity Problem* (NLCP) (Anitescu & Potra, 1997). One can derive this optimization problem using the same framework as bilateral constraints, starting from some holonomic constraint $f_c(0)$ to enforce, even though it cannot be done exactly. In the particular case of contact constraints, the notion of target pose previously introduced is ill-defined. It is not really a fixed point in space but rather a moving target. At every instant, the target position is the projection O onto the surface of the contact point P. Therefore, if it slips in translation or rotation, then the target pose is updated accordingly. The reference frame is defined in a way that its vertical axis is normal to the ground while the two orthogonal axes in the tangential plane are arbitrary. This choice preserves the isotropy of Coulomb friction and eases computations for the complementarity conditions of the contact model. It yields,

$$f_c(q) = \begin{pmatrix} 0 & 0 & d & 0 & 0 & 0 \end{pmatrix}^T J_c(q) = \begin{pmatrix} \bar{R}^T & 0 \\ 0 & \bar{R}^T \end{pmatrix} {}^0J_P \tag{C.18}$$

where d is the depth of the contact point, ${}^0J_P$ is the jacobian of the position and orientation of the body B at point P in the world frame. Using this formulation, Baumgarte stabilization progressively brings the contact point back to the surface without any effect at the position-level in the tangential plane and cancels out the spatial velocity of the contact point. However, it is no longer guaranteed that constraint violation will reduce over time because of bounded forces. If there are n contact points, then the constraint function f_c is the concatenation of the set of individual constraints $\{f_{c,i}\}_{i=1}^n$.

The resulting constrained dynamics is discontinuous in velocity because of this contact model. As a result, it is not possible to integrate the dynamics at the acceleration level anymore. The classical approach is to integrate the velocity directly, dealing with impulses instead of forces. Impulses $\tilde{f}$ are integral of the forces f over the integration timestep h. Impulses associated with contact constraints are inconsistent with their time-continuous counterpart because of the discontinuous nature of the contact model, known as *Painleve's paradox*. The latter states that contact impulses are not equal to the timestep multiplied by the continuous-time forces, not even at the limit when the timestep approaches zero. Indeed, for the time-continuous formulation, forces would grow unboundedly, while the corresponding impulses are always bounded. Only a position shift can occur at the velocity level, which can only be on the normal axis for contact constraints as the reference position p_O in the tangential plane is the contact point itself p_P. Since the contact model enforces the normal velocity to be positive, the contact point will naturally move up to the surface without target velocity to stabilize the constraint. Thus, Baumgarte stabilization can be ignored. Todorov et al. (2012) reformulated equation (C.11) at the velocity level by discretizing the constraint acceleration $a_c = (v^+ - v^-)/h$ where h is the timestep. It results in an *implicit time-stepping scheme* for computing the next timestep velocity v^+,

$$\begin{aligned} v^+ &= -A_\gamma \tilde{f} + h a_{c,\text{free}} + v^- \\ 0 \leq v_n^+ &\perp f_n \geq 0 \\ 0 \leq \|v_t^+\|_2 &\perp \mu f_n - \|f_t\|_2 \geq 0 \\ v_t^+ \cdot f_t &= -\|v_t^+\|_2 \|f_t\|_2 \end{aligned} \tag{C.19}$$

One can demonstrate it admits a unique solution if A is positive definite. It corresponds to the set of first-order KKT conditions of the following *Second-Order Cone Program* (SOCP),

$$\begin{aligned} f^* &= \arg\min_{\tilde{f} \in \mathcal{K}} \frac{1}{2} \|v^+\|^2_{A_\gamma^{-1}} \\ \text{s.t. } v^+ &= -A\tilde{f} + h a_{c,\text{free}} + v^- \end{aligned} \tag{C.20}$$

where $\|v^+\|^2_{A_\gamma^{-1}}$ can be interpreted as the kinematic energy in contact space (Todorov et al., 2012) and $\mathcal{K} = \{x = (x_t, x_n) \in \mathbb{R}^2 \times \mathbb{R} \mid x_n \geq 0, x_n - \|x_t\|_2 \geq 0\}$ denotes the *second-order cone*. The position is then integrated based on the next timestep velocity v^+ using a so-called *semi-implicit* integration. This is essential for numerical stability as the velocity is integrated implicitly and can change discontinuously. Usually, Euler's semi-implicit scheme is used, but nothing prevents from modifying Runge-Kutta 4 to operate on the next timestep velocities rather than accelerations (Stewart & Trinkle, 1996; Todorov et al., 2012).

Another approach is the regularization of the contact model to ensure the forces and accelerations are time-continuous and bounded. Providing smoother dynamics over the velocity-based method is a significant advantage for machine learning algorithms, which are best suited for continuous problems. Indeed, it can be demonstrated that the required number of parameters for a neural network is directly related

to the regularity of the function to approximate through its Lipchitz constant. Neural networks of low complexity are naturally more robust to input noise and easier to train, which is critical for training control policies. Moreover, it makes it harder for the agent to abuse the physics, thereby improving transferability. Yet, the gap between the approximate and exact models should be marginal for the simulation to be realistic. We resort to Baumgarte stabilization to smooth out the system dynamics at impact by gradually enforcing the constraints over time. In particular, there is no temporal discontinuity up to the acceleration level.

Baumgarte stabilization has the additional benefit to cancel any position shift along the normal axis over time. On the one hand, this relaxes the need for enabling contact constraints at the exact impact timing, allowing larger integration timestep. On the other hand, it enables handling arbitrary ground profiles without special care. Specifically, if the ground profile is locally concave, then the depth of the contact point would increase in case of slippage. Stewart and Trinkle (1996) address this issue by replacing the non-penetration condition with a more complex one. The latter is coupling the velocity in the normal and tangential directions to enforce sliding along the surface, which requires having access to the gradient of the ground profile. It is important for the velocity-level formulation, but it is much less of an issue for the acceleration-level formulation thanks to Baumgarte stabilization constantly moving the contact point back to the surface.

The objective is to solve an alternative SOCP,

$$\begin{aligned} f^* &= \arg\min_{f \in \mathcal{K}} \frac{1}{2} \|\delta\|^2_{A_\gamma^{-1}} \\ \text{s.t. } \delta &= a_c - \bar{a}_c = -A_\gamma f + b \end{aligned} \tag{C.21}$$

where $\|\delta\|^2_{A_\gamma^{-1}}$ as no straightforward physical meaning, unlike the velocity-level formulation. This choice can be better understood by considering its dual NLCP,

$$\begin{aligned} \delta = a_c - \bar{a}_c &= -A_\gamma f + b \\ 0 \leq \delta_n \perp f_n &\geq 0 \\ 0 \leq \|\delta_t\|_2 \perp \mu f_n - \|f_t\|_2 &\geq 0 \\ \delta_t \cdot f_t &= -\|\delta_t\|_2 \|f_t\|_2 \end{aligned} \tag{C.22}$$

The target acceleration $\bar{a}_c$ for contact constraints is given by

$$\bar{a}_c = -(\omega^2 d e_n + 2\omega v_n) - 2\omega v_t. \tag{C.23}$$

Typically, $\omega \gg 1$, such that δ is almost colinear with $\bar{a}_c$. Therefore, $v_t \cdot f_t \approx -\|v_t\|_2 \|f_t\|_2$, which complies with the maximum energy dissipation principle unless the acceleration of the contact point is not dominated by its velocity. Using the same reasoning, the tangential target acceleration $2\omega v_t$ should be large enough to saturate the tangential force at the boundary of the friction cone unless the tangential velocity closely approaches zero. It implies that $0 \leq \|\delta_t\|_2$ is equivalent to $0 \leq \|v_t\|_2$ almost every time, as illustrated in figure C.4.

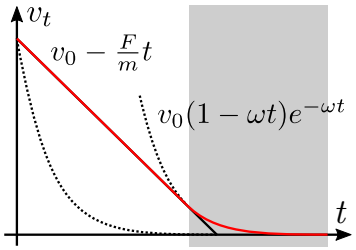


Figure C.4: Theoretical temporal evolution of tangential velocity of a point mass m for the acceleration-level formulation of constrained dynamics. Flat ground and constant normal force are assumed. The maximum tangential force resulting from Coulomb friction is denoted F. It matches the true contact model from the beginning until the very end when the target velocity is too slow to saturate the tangential force. It prevents discontinuity at the velocity level so that it smoothly stops asymptotically. The contact model mismatch transient is highlighted in grey.

The approximate contact model can be quite off during transient phases because the maximum dissipation principle is not strictly enforced. Notably, the ground reaction force would accelerate the tangential velocity instead of slowing it down if the target acceleration is higher than the free acceleration. In practice, this unrealistic phenomenon always occurs right before transitioning from slipping to sticking and slightly delays its timing. This discrepancy is absolutely necessary for the acceleration to be continuous. Indeed, the orientation of the force cannot change instantly from forcibly aligned with the velocity to whatever is necessary to hold the contact point in place. Even though it can be seen as a violation of the laws of physics, the induced extra slippage is insignificant and can be neglected.

The behavior along the normal direction is more tricky to analyze because of the depth d generally being non-zero at impact. Assuming flat ground and positive normal force, critically damped Baumgarte stabilization ensures the normal velocity v_n is first positive during $1/\omega$, then negative until the contact point reaches the surface asymptotically. As expected, it does not match the true contact model during this first phase, else it would be impossible to guarantee the continuity of the acceleration and bounded forces. The complementary condition on the normal axis in problem (C.22) entails that of the true contact model after this short transient. Still, it is more restrictive as it forces the contact point back to the surface (see figure C.5).

Overall, the regularization only has an effect during short transient phases at impact or between sticking and slipping, which should barely affect the simulated trajectory if properly tuned. Lowering the critical time of Baumgarte stabilization would shorten the transient phases. Yet, it would result in a stiff differential equation, leading to numerical instability if the integration timestep h is not reduced. The formulation of the constrained dynamics at the acceleration level through regularization of the contact model is the preferred approach in Jiminy. A more comprehensive case study is presented in appendix C.2.3.

To a lesser extent, complementary-free approaches based on a convex contact

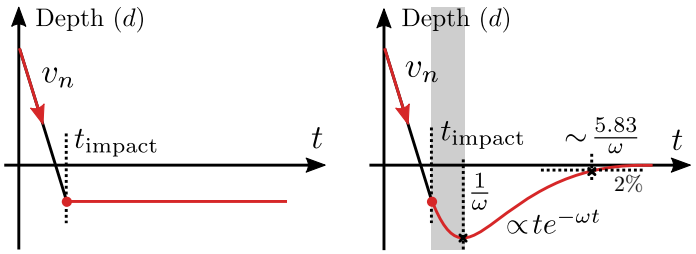


Figure C.5: Theoretical comparison of the temporal evolution of contact depth for velocity-level and acceleration-level formulation of constrained dynamics. The normal velocity of the contact point goes instantly to zero as soon as impact is detected for the velocity-level formulation on the left. On the contrary, both the position and velocity are continuously differentiable for the acceleration-level formulation due to Baumgarte stabilization, and the contact point is forced to slowly move back to the surface. The contact model mismatch transient is highlighted in grey.

model are used by some simulators, Mujoco being by far the most famous simulator relying on such model (Todorov, 2010, 2014). It has the advantage to approximate the original optimization problem that would be NP-hard in the general case without regularization by a convex optimization that can be solved efficiently using newton-based methods. The resulting contact dynamics is analytically invertible, which means that the forces can be recovered from any state and derivative of the system $(q, \dot{q}, \ddot{q})$, regardless of the constraint violations. In the same way, the analytical gradient of the contact model can be obtained efficiently. It is a major advantage for trajectory planning and optimal control that requires access to the gradient of the contact problem. The issue with such a model is that it tends to be constantly unrealistic and not just during transient phases. For this reason, this kind of approach is not well-regarded by the robotics community, for which being able to simulate realistic contact is often critical for transfer from simulation to reality.

Contact models based on phenomenological spring-damper interaction forces were popular in the 90s (Marhefka & Orin, 1996). In these phenomenological models, the ground reaction is completely decentralized, ignoring any coupling between the contact points. Thus, computations are simple to implement, interpretable, and extremely fast (scaling linearly with the number of contact points). However, they have fallen out of favor on account of their many drawbacks. First, it requires exact impact timing detection to prevent forces from exploding due to large penetration depth. Relying on an adaptive timestep integrator alleviates this issue without effort. Yet, the average timestep would hardly exceed 10us for legged locomotion. This may substantially increase the overall computational cost if not mitigated by updating some rigid body quantities at a lower frequency. Moreover, the resulting dynamics lacks smoothness, limiting the performance of policy learning methods. Secondly, viscous friction can be modelled but not static friction, also called *stiction*. It means that the contact points must have some residual tangential velocity for a tangential force to be applied. Consequently, the support feet are constantly moving on the

ground. This is problematic because they are supposed to be fixed on the ground for both planning and control. Besides, the lack of stiction acts as a low-pass filter on the dynamics and may hide some excitatory coupling between observation and control that may occur experimentally. Finally, it requires tedious manual tuning of the stiffness and damping parameters to be numerically stable. If the contact-space inertia is too large, then it will result in a large penetration and oscillations. Conversely, if the inertia is too small, then the contact dynamics will be stiff and difficult to integrate numerically (Todorov et al., 2012). Since the contact-space inertia depends on the current configuration of the robot, it may be impossible to find constant parameters achieving a good trade-off regardless the motion for a given system. Nevertheless, the stiffness and damping parameters can be adjusted automatically to make sure the contact points are critically damped if the diagonal of the matrix A is known, closing the gap with Baumgarte stabilization. Jiminy offers two contact models: a phenomenological one with adaptive critically damped parameters for joint bounds collisions, and an analytical one enforcing unilateral constraints.

It can be demonstrated that both problems (C.19) and (C.22) admit a unique solution as long as A_γ is positive definite, which is always the case for $\gamma > 0$. They both are very similar and can be tackled using the same methods. It could be solved directly using a general-purpose non-linear programming algorithm. It is never done in practice because it is likely to be very slow, and convergence cannot be guaranteed. Another option is to formulate the NLCP as fixed point optimization problems consisting in repeatably solving a well-posed SOCP, much like the proximal formulation aforementioned. Acary et al. (2011) have proven that it converges even if the solution is not unique. The approach is computationally very costly and never used in practice. Instead, the issue is usually overcome by making several approximations, in a way that it can be reformulated as a *Mixed Linear Complementary Problem* (MLCP) (Anitescu & Potra, 1997; Arechavaleta et al., 2009). Then, it can be solved exactly using algorithms with strong global converge guarantees such as *Lemke* (Cottle et al., 2009) or *PATH* (Dirkse & Ferris, 1995). The first approximation is the linearization of the friction cone. It consists of writing the tangential force as a linear combination of forces along several predefined directions and bounding the magnitude of the force along these directions independently. It can be interpreted as a polyhedral approximation of the friction cone figure C.6 (Stewart & Trinkle, 1996). This approximation is anisotropic because the maximum tangential force depends on the orientation of the contact points on the world plane. Increasing the number of directions alleviates this issue, at the cost of increasing the dimensionality of the problem to solve. The canonical form of a MLCP is the following:

$$\begin{aligned} w^+ - w^- &= Ax + b \\ 0 \leq w^+ &\perp x - l \geq 0 \\ 0 \leq w^- &\perp u - x \geq 0 \end{aligned} \tag{C.24}$$

At this point, the NLCP cannot be formulated this way because its complementary conditions are mutually dependent. Indeed, the maximum tangential force depends

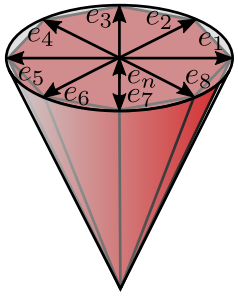


Figure C.6: Example of polyhedral linear approximation of friction cone.

on the normal one despite the linearization of the friction cone. To get around this limitation, the normal force involved in the complementary conditions is usually replaced with a constant estimate. The value resulting from bilateral constraints is a reasonable estimate. This amounts to inverting the linear system $w^+ - w^- = Ax + b$. If the set of active contact points has not changed, then the value at the previous timestep is an even better estimate that is otherwise free of cost. The problem can now be written as a MLCP, but the impact of this approximation is hard to guess.

Solving MLCPs is computationally demanding. It is generally the main bottleneck of the simulation pipeline, even for high-dimensional systems. Thus, choosing the most efficient solver is critical. The splitting methods are iterative methods originally intended for solving unconstrained linear systems. They can be adapted for solving MLCPs. However, they scale poorly with the dimensionality of the problem, and their convergence is slow. Because of that, these iterative methods are disregarded in many fields, in favor of non-iterative methods such as the Lemke and PATH algorithms. Nevertheless, the splitting methods remain competitive in the particular context of physics simulation (Enzenhöfer et al., 2018; Lacoursière, 2003; Servin et al., 2014). This can be explained by several reasons. First, poor scalability is barely an issue for robotics applications. The dimensionality of the problem only depends on the number of constraints, not the kinematic tree of the robot itself. Legged robots are usually bipedal or quadrupedal, corresponding to at most 16 contact constraints. Therefore, the dimension of the problem should be less than 50 components. Next, the complementary conditions are guaranteed to be satisfied after every iteration. More precisely, the force is always inside the friction cone, although the maximum energy dissipation principle may not be satisfied. As a result, the algorithm can be aborted to spare computational resources if necessary, and it would still provide a physically meaningful solution. In addition, splitting methods guarantee that the error is strictly decreasing so that extra iterations always improve the accuracy of the solution. Finally, they can easily be warm-started even if the set of active contact constraints is changing dynamically. In such a case, the Lagrangian multipliers associated with newly active constraints are initialized to zero, whereas the others keep their previous values. It is a major advantage since the solution at the current timestep is likely to be very close to the previous one.

The *Gauss-Seidel* or *Jacobi* methods are the most common splitting methods (La-

coursière, 2003). They are both very similar. While the Gauss-Seidel method relies on the forward substitution, the Jacobi method is performing the very same operations but simultaneously. This difference has an impact on the respective conditions of convergence. The Gauss-Seidel method is guaranteed to converge if the matrix A of the linear system is positive definite. This is the case after linearization, provided that exactly two orthogonal tangential directions are considered for the discretization of the friction cone, corresponding to the well-known *friction pyramid*. Conversely, the condition is more restrictive and not always verified for the Jacobi method: the matrix A must be diagonal dominant. Besides, the Gauss-Seidel method tends to converge significantly faster than the Jacobi method in terms of the number of iterations. Quarteroni et al. (2007) have shown that it converges twice as fast for some pathological scenarios. Thus, the Gauss-Seidel method is preferred in this work.

It is straightforward to adapt splitting methods for handling lower and upper bounds on the unknown x. It simply consists in clipping the current iterate after every operation. The ensuing algorithms are proven to converge under the same conditions as originally if the bounds are fixed, but it gets more complicated otherwise. The extension of the Gauss-Seidel method that handles bounds using this technique is called *Projected Gauss-Seidel* (PGS) Silcowitz et al. (2010). Quarteroni et al. have proven that PGS converges for A is positive definite whatever the initial guess. In general, it may converge for some initial guesses and diverge for others, but it will always converge if the initial guess is close enough to a solution. Moreover, the residual $Ax - b$ always converges even if the solution x diverges, so it is more appropriate to define the stopping criteria in residual space. *Randomized Kaczmarz method* is a variant of the coordinate descent method which closely resembles PGS. It is proven to converge for A positive semi-definite matrix, relaxing the need for regularization. However, it is computationally about twice more expensive per iteration, and it requires much more iterations to converge in practice. Therefore, PGS is more appropriate if regularization must not be avoided at all costs. If A is symmetric positive definite, the Gauss-Seidel method is monotonically convergent for the norm $\|\cdot\|_A$, and the spectrum radius $\rho(A)$ controls the convergence rate. Regularization enforces a lower bound to the spectral radius $\rho(A) > \gamma$ and subsequently the convergence rate. There is no general result about the convergence rate of PGS. Worse still, it is affected by the index ordering when the bounds are mutually dependent. In particular, because the maximum tangential force at each contact point depends on the associated normal force through the friction cone. Thus, updating first all the normal forces and only then all the tangential forces is expected to converge faster than the other way around or even simultaneously.

PGS convergence rate can be further enhanced using *subspace minimization*. It consists of performing PGS iterations on the full problem, followed by fixing the active set and performing iterations on the reduced problem, then starting over. Indeed, PGS is known to be very efficient when it comes to identifying rapidly a good candidate for the active set (Silcowitz et al., 2010). Additionally, *Successive Over Relaxation* (SSOR) can be used to enhance the convergence rate, but it introduces a new parameter tricky to tune to get consistent improvement. Note that increasing

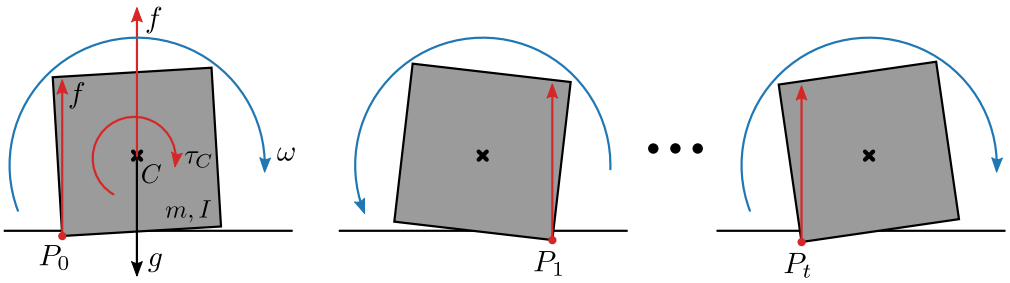


Figure C.7: Pathological issue with collision detection for a box on flat ground using the deepest point as the unique contact point. The ground reaction is jumping discontinuously from one corner to the other, inducing a momentum at the *Center of Mass* (CoM) of the box, therefore never stabilizing flat on the ground.

regularization also makes convergence faster, but it increases the gap between the current and target constraint acceleration, which makes the simulation less realistic.

The friction pyramid approximation is problematic for learning control policy since the orientation in the world plane is usually not observable. This makes it much harder for the agent to take advantage of contact force information as it appears noisy from its perspective, limiting its performance. For instance, the policy cannot predict whether the robot is going to slip. We present a novel algorithm 5 for solving problem (C.22) without any approximation. It is inspired by PGS and Jacobi methods, combined with the projection on a second-order cone.

Assuming it converges, the result must be a fixed point of the algorithm. Examining all individual cases, one can prove A positive definite is sufficient but not necessary. If it converges, then its output is the solution of problem (C.22). The update of tangential forces is done for both axes simultaneously, slightly improving the computational efficiency over vanilla PGS. Modifying this algorithm to take into account the torsional friction would be straightforward.

C.2.3 Contact Detection

Ground contact interaction is usually loosely approximated by a finite set of contact points. Being able to determine efficiently the optimal set of contact points minimizing the discrepancy between reality and simulation is an active field of research. For generic shape, this problem is NP-hard, so heuristics must be used to find a set of points that is good enough with limited computational resources. This problem is related to collision detection. State-of-the-art collision libraries only provide a single contact point for every shape, regarded as the deepest point of intersection between the two mediums. This is not enough to simulate ground contact efficiently. For instance, consider a box on a flat ground figure C.7, the deepest point is jumping discontinuously from one corner to the other at every point in time and never stabilizing whatever the integration step. This kind of issue happens systematically for shapes that are not strictly convex. There are many heuristics to try to overcome this limitation, among them using remanent contact points or limiting the maximum dis-

Algorithm 5: Successive Second-Order Cone Displacement

Input: Linear system to solve: A, b - Initial guess: $f^{(0)}$
Output: Solution: f
Compute initial residual $\delta^{(0)}$: $\delta^{(0)} = Af^{(0)} + b$;
for $k = 0$ **to** k_{max} **do** `// do at most` $k_{\texttt{max}}$ `iterations`
  for $i = 0$ **to** n **do** `// loop over` n `contact constraints`
    Update normal force $f_{n_i}^{(k)}$ using forward substitution:

$$f_{n_i}^{(k+1/2)} \mathrel{+}= \frac{1}{A_{n_i,n_i}} \left(b_{n_i} - \sum_{j<n_i} A_{n_i,j} f_j^{(k+1)} - \sum_{j \geq n_i} A_{n_i,j} f_j^{(k)} \right);$$

    Enforce positive normal force:

$$f_{n_i}^{(k+1)} = \max\left(0, f_{n_i}^{(k+1/2)}\right);$$

    Update tangential force $f_{t_i}^{(k)}$ for x and y axes simultaneously:

$$f_{t_i}^{(k+1/2)} \mathrel{+}= \frac{1}{\max(\mathrm{diag}(A_\gamma)_{t_i})} \left(b_{t_i} - \sum_{j<t_i} A_{t_i,j} f_j^{(k+1)} - \sum_{j \geq t_i} A_{t_i,j} f_j^{(k)} \right);$$

    Project tangential force on friction cone:

$$f_{t_i}^{(k+1)} = \min\left(1, \mu \frac{f_{n_i}^{(k+1)}}{\left\| f_{t_i}^{(k+1)} \right\|_2}\right) f_{t_i}^{(k+1/2)};$$

  end
  Compute updated residual δ^k: $\delta^{(k+1)} = Af^{(k+1)} + b$;
  if $|\delta^{(k+1)} - \delta^{(k)}| < tol_{abs}$ **or** $|1 - \delta^{(k+1)}/\delta^{(k)}| < tol_{rel}$ **then**
    break ; `// Abort if tolerance satisfied for every component`
  end
end

placement of a unique contact point at every integration step. How to do it properly is an active field of research. Alternatively, one can approximate the true body shape by their *convex hull*, which is the smallest convex set that contains the true one. In this way, the deepest point is guaranteed to move continuously along its surface, and therefore the integration is numerically stable if the integration step is small enough. However, the convex hulls can be very different from the original shapes, leading to unrealistic simulations. It is the case of the feet of the exoskeleton Atalante for which bumpers at the corner of the sole plate elevate it about 2 cm about the ground, much like the pathological chair example figure C.8.

In Jiminy, the real shape of the collision bodies is ignored and replaced by a finite set of candidate contact points at the surface. Those candidate contact points must be placed carefully to minimize the corresponding approximation error. Increasing

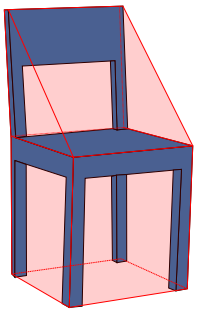
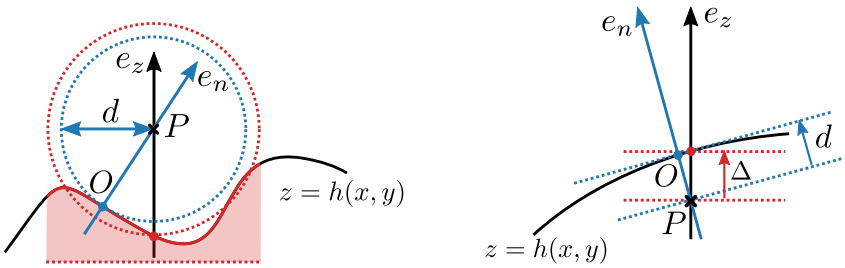


Figure C.8: Illustration of convex hull for a chair

the number of points strictly decreases the error but increases the computational burden as well. Typically, boxes are replaced by the 8 corner points, eventually including the center of each face. Such an approach is satisfactory as long as the regularity of the ground profile is consistent with the chosen set of points. The *Nyquist-Shannon sampling theorem* can be used to assess if the chosen set of points makes sense. Indeed, this theorem establishes a sufficient condition for a discrete sequence to capture all the information from a continuous signal of finite bandwidth. In this case, the discrete sequence relates to the contact points, and the continuous signal is the ground profile. It states that if the lowest period of spatial patterns of the ground profiles is D meters, then it can be approximated by a series of points spaced at most $2D$ apart. In the following, it is assumed that a finite set of candidate contact points has been defined for the bodies in the system that are likely to collide with the ground. It is commonly limited to the end of every sub-chain in the kinematic tree to speed up the simulation. Similarly, handling of internal collisions between bodies can be ignored and replaced by a safety radius associated with each body. The simulation is aborted as soon as the two spheres intersect. Such an approximation is not limiting for most applications since the motion is likely to be unsatisfactory when it happens. For Atalante, there is one contact point at the center of each bumper under both feet. This is necessary because there is one vertical force sensor inside each bumper. Simulating the sensors require having access to the force applied to their locations. The contact problem is not invertible, which means that it is not possible to uniquely decompose the spatial force applied on each foot into linear force on each bumper. Therefore, it is only possible to simulate such sensors if candidate contact points are coincident with them. In such as case, their measurements are the extraction of the vertical forces from the solution of the contact problem directly. This modelling turns out to be quite realistic in practice for Atalante.

The ground profile is defined as a smooth heightmap function $h \in \mathcal{C}^1$ that maps the position in world plane (x, y) to the height of the ground z_h and unit normal axis e_n at this location. The restriction to a heightmap ensures that the search region of the closest point on the ground from the contact point is closed and bounded. The computation of the closest point is tractable since it can be formulated as a global optimization problem of a non-convex smooth function with bounded Lipschitz



(a) Exact (b) Approximation

Figure C.9: Exact and approximate methods to compute the closest point to a smooth heightmap ground profile h. Both methods are equally valid no matter if the candidate contact point P is above or below the ground profile.

constant over a closed bounded region (see figure C.9a). Algorithms exist to solve exactly this problem, but they are computationally costly. Instead, a very fast first order approximation method is used in Jiminy (see figure C.9b). First, the height z_h and normal axis e_n of the ground is computed at the position in world plane (x_P, y_P) of a candidate contact point P, then the signed depth d is estimated projecting the signed vertical height difference $\Delta = z_h - z_P$ on the normal, $d = \Delta(e_n \cdot e_z)$. A contact constraint is activated if the contact point is below the ground profile. However, this constraint is not deactivated as soon as the contact point gets above the ground, but a hysteresis parameter d_{thr} is introduced to improve the numerical stability. Its value is around 1mm. This procedure is summarized by algorithm 6. We assess its performance for a falling box on flat ground (see figure C.10).

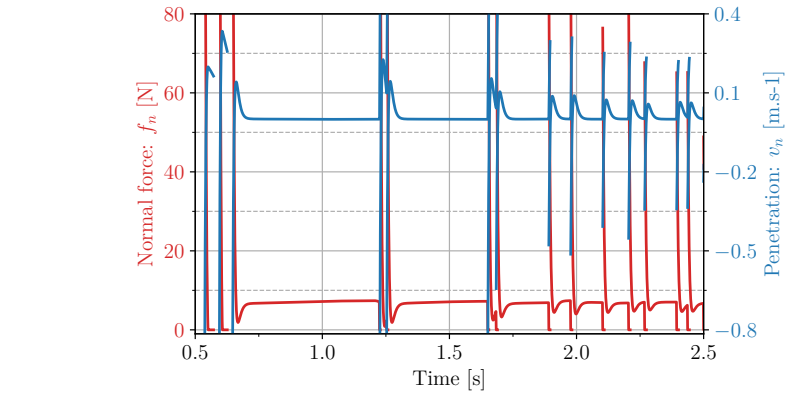
C.3 Mechanical Transmissions and Actuators

C.3.1 Direct Drive

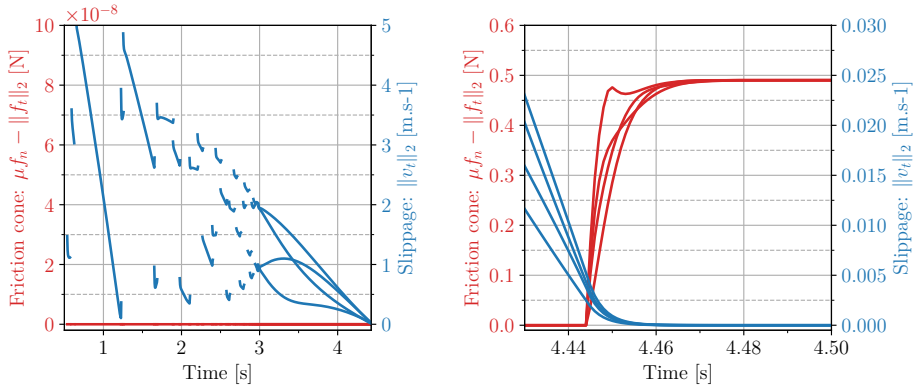
The rigid body dynamics algorithms are unaware of the hardware powering the robot. Yet, taking into account the dynamics of the motors in simulation is important. For instance, the torque and power that a motor can deliver are limited. This limits the efforts that can be applied at the joints by the controller in a way that depends on the current state of the robot. If this aspect is ignored when training control policies, the motor torque may saturate unexpectedly and the robot may fall incidentally.

Physically, the mechanical transmissions are the structural elements that cause the actuators to apply forces to the joints. Each transmission system couples one or more actuators with one or more joints, which are forced to move together as a consequence of this coupling. Most of the time, the motors and joints are connected one by one, which is referred to as *direct drive*. In this particular, the dynamics of the motors and joints are mathematically confused, and thus it is not necessary to model the transmissions explicitly to take into account their effects.

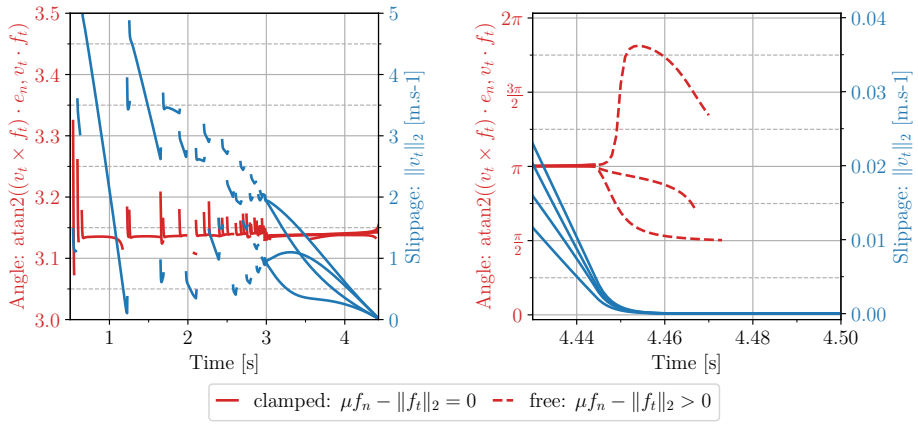
The mechanical parts of transmissions are often not weightless and have their



(a) Normal complementarity



(b) Tangential complementarity



(c) Energy dissipation

Figure C.10: Evaluation of acceleration-level constrained dynamics formulation on a box of size $1\text{m} \times 1\text{m} \times 1\text{m}$ and mass 1kg. There is one candidate contact point at each corner. The box is thrown on flat ground with an initial height of $z = 1.5\text{m}$, linear velocity $v_x = 3.0\text{m}\,\text{s}^{-1}$, and angular velocity $\omega_x = 1.0\text{rad}\,\text{s}^{-1}$, $\omega_z = -5.0\text{rad}\,\text{s}^{-1}$. The regularization is set to $\gamma = 1\text{e}{-3}$. It ensures the tangential forces go back to zero. The explicit Euler scheme with 1ms timestep is used for integration.

Algorithm 6: Contact detection

for $i = 0$ **to** n **do** `// loop over` n `candidate contact points`
  Fetch position $p_P = (x_P, y_P, z_P)$ of the candidate contact point i;
  Evaluate ground profile at P: $z_h, e_n = h(x_P, y_P)$;
  Compute first-order approximate signed depth d: $d = (z_h - z_P)(e_n \cdot e_z)$;
  if $d \geq 0$ **then** `// below ground surface`
    `// enable constraint if below ground surface`
    Activate constraint;
  else if $d \leq -d_{thr}$ **then**
    `// disable constraint if significantly above ground surface`
    Deactivate constraint;
  end
  if *Constraint is active* **then**
    Update target pose (p_O, R_O): $p_O \leftarrow p_P - de_n, R_O \leftarrow R_P$;
    Update local reference frame $\bar{R}$:

$$e_{t_x} = \frac{e_n \times e_x}{\|e_n \times e_x\|}, \qquad e_{t_y} = e_{t_x} \times e_{t_y}, \qquad \bar{R} = \begin{pmatrix} e_{t_x} & e_{t_y} & e_n \end{pmatrix};$$

  end
end

own inertia. Usually, the inertia of the structural elements is very light and can be neglected. However, it may not be the case for the actuators. The importance of taking into account the inertia of the rotor, the so-called armature inertia, can be highlighted in a simple case. Let us consider a single revolute joint driven by a simple rotary motor through a gearbox. Now, assume the rotation axis of the motor is aligned with the joint, the transmission is fixed in the world frame, and the structural elements of the transmission have no inertia. Let q_m be the actuator position and q_j be the joint position. I_{load} denotes the inertia of the load attached to the joint, I_m is the inertia of the motor, and $r > 1$ is the gear ratio. It follows,

$$\dot{q}_m = r\dot{q}_j, \qquad u = ru_m. \tag{C.25}$$

The transmission is in an inertial frame, so the total kinetic energy of the system is,

$$K = \frac{1}{2} I_{load} \dot{q}_j^2 + \frac{1}{2} I_m \dot{q}_m^2 = \frac{1}{2} I_{eq} \dot{q}_j^2, \tag{C.26}$$

where $I_eq = I_{load} + r^2 I_m$ is called *reflected inertia to joint*. This is the equivalent inertia viewed from the joint. Generally, the armature I_m is very small relative to the load I_{load}. However, its impact is quadratic with respect to the gear ratio r. If the latter is large, which is often the case in practice, then it can significantly affect the equivalent inertia and even dominate the load. It clearly shows that the effect of the armature on the kinematic energy, hence on the dynamics, cannot be neglected.

For more complex systems including all free-floating base robots, transmissions are not fixed in the world frame but rather embedded into the system. Taking into account the transmissions is harder if they are not in an inertial frame, as it cannot be modelled as equivalent inertia. Indeed, a Coriolis effect will appear as long as the rotation axis of the motor is not orthogonal to its angular velocity, which depends on the configuration of the robot in general and therefore cannot be assumed. The derivation of the exact dynamical equations including the transmission inertia can be obtained as usual using the Lagrangian formalism presented previously (Sciavicco et al., 1995). To this end, each transmission is considered as a separate entity connected to the robot. It results in a dynamical system whose state includes both the classical joint state and the motor state. Then, the kinematic constraint relating the state of the motors to the state of the joints is enforced through additional bilateral constraints. The advantage of this approach is to model the poly-articulated system and its associated transmissions using the same formalism and without any approximation. However, first, it doubles the dimensionality, which is raising the computation cost for rigid body algorithms involved in the numerical integration. Secondly, it requires solving an optimization problem to compute the acceleration of the system based on Gauss's principle of least constrained, which does not have a closed-form solution in the general case. Solving this problem in particular is several times more expensive than neglecting transmission.

Various approximations can be made to avoid solving this optimization problem explicitly. One solution is to suppose that the coupling between the joint and the transmission is not rigid but flexible. The kinematic constraints associated with transmissions are then replaced in spring forces (Siciliano & Khatib, 2008, Part B|Section 13.1). This model is known as *Serial Elastic Actuator* (SEA). Although effective from a mathematical point of view, it has the same limitations as the phenomenological spring-damper contact model: it results in a stiff differential equation that requires a small integration timestep to be numerically stable, and it adds extra vibration modes that may be unrealistic and impeding control performance. The standard approach is to assume that the kinetic energy of the transmission is due to its own spinning only. It boils down to considering that the transmission system is in an inertial frame by neglecting the coupling effects between the transmission and the link motion. Consequently, there is no additional Coriolis effect due to the transmissions, and it can be modeled as an equivalent inertia as in the simple case. Spong (1987) introduced first this approximation for the modelling of elastic revolute joints. Later, it was extended to any joint with rigid coupling. The equivalent inertia matrix is obtained by adding a block diagonal matrix to the original inertia matrix without transmissions computed with *Composite Rigid Body Algorithm* (CRBA). The block associated with each joint depends on its type. For robots having only linear and revolute joints whose axes are colinear to the ones of the motors, it is a diagonal matrix with non-zero values equal to the rotor inertia at joint-level $r^2 I_m$ for components corresponding to actuated joints. This approximation is implemented in Jiminy as it is cheap to evaluate and constitutes a strict improvement over neglecting the inertia of the transmissions completely.

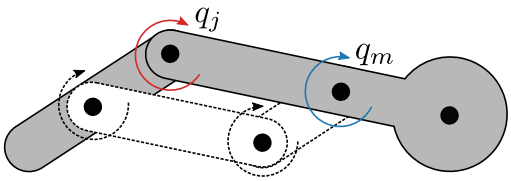


Figure C.11: Pantograph example of closed kinematic chain transmission

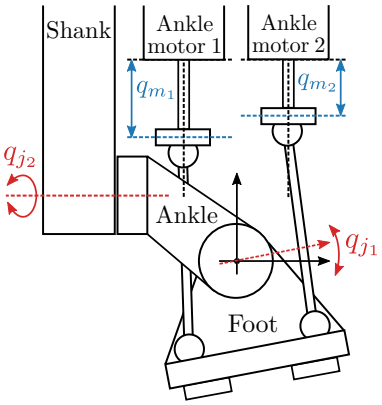
C.3.2 Advanced Transmissions

Complex transmissions are sometimes used in robotics. Differential drives are good examples of non-trivial transmissions that are widespread in wheeled robots. In this section, we present the generalization of direct drive to any n-to-m transmission.

Formally, the motion in motor space $\mathcal{Q}_m$ is related to the motion in joint space $\mathcal{Q}_j$ by some forward transformation $f : \mathcal{Q}_m \mapsto \mathcal{Q}_j$. If the transformation is invertible, the torque and armature inertia can be reflected at the joint level. It follows that the state of the system can be reduced to the one of the mechanical structure and integrated using the classical *Forward Dynamics* (FD) algorithm. The updated state of the motors can be finally computed in closed form. For reasoning is the same as for bilateral contact constraints (cf.). However, the transformation may not be invertible. If the transformation is injective, then the joint state is over-constraint. For instance, the transmission used in graspers to power the two pincers with a single motor falls into this category. A reduced formulation of the system dynamics can still be formulated, but additional holonomic constraints must be enforced to restrict the joint space in accordance. The number of independent constraints corresponds to the rank deficiency of the transformation. The resulting optimization problem can be solved using the approach presented in appendix C.2.1. On the contrary, if the transformation is surjective, then the motor state is undetermined. This is the most challenging scenario to simulate. The only option is to jointly integrate the dynamics of the actuators and motors, coupled together by the forward transformation as a holonomic constraint $f(q_m) = q_j$. This approach is the most generic and can be applied in any other case. However, it significantly increases the computational cost compared to relying on the reduced dynamics whenever possible.

Some closed kinematic chains can be modelled equivalently as the combination of an open kinematic chain with a transmission whose transformation is injective. Leveraging this property would speed up the simulation. The pantograph mechanism is a classical example figure C.11. This kind of mechanical design is interesting as it reduces the inertia of moving parts by placing the actuators closer to the base.

In the case of Atalante, the transmission of the ankle is quite complex. It maps two linear motors to two joints whose axes are orthogonal but not coincident figure C.12. The position of the linear motors is bounded, unlike their rotary counterparts. This reduces the operation space of the joints, and therefore must be accounted for in the simulation. Doing so is straightforward since all it requires is enabling one additional holonomic constraint at the joint level whenever a motor hits a bound.



$$[q_{j_1}, q_{j_2}] = f([q_{m_1}, q_{m_2}])$$

Figure C.12: Transmission of the ankle of Atalante exoskeleton

D Classical Neural Networks Architectures

A bunch of neural network architectures have emerged over the last two decades. Altogether, they cover most applications targeting specific tasks, but none of them is capable of *general intelligence*, i.e. being able to address any task indifferently and simultaneously as a human would do. Hereafter, we present the classical architectures and review some applications for which it achieves state-of-the-art results.

D.1 Multi-layer Perceptron

The *Multi-Layer Perceptron* (MLP) is an generic and unstructured type of *Feedforward Neural Network* (FNN) that has been introduced more than half a century ago. It has been the only architecture found in the literature for about 30 years. It is the most fundamental class of *Artificial Neural Networks* (ANNs) and is still ubiquitous up to this day. MLPs are known to be universal approximators of continuous functions, so they could theoretically tackle a huge diversity of problems, provided that all the necessary information is forwarded as input.

They have the advantage to be simple to implement and relatively cheap to train. Fairly small neural networks with up to three hidden layers having one million parameters each are surprisingly easy to train in practice and cheap enough to be evaluated in embedded hardware afterward. This is large enough for most applications, which makes the MLP the most widely used architecture in the industry. However, they tend to perform poorly for problems with high-dimensional inputs as computer vision tasks, or when it requires memories of events that happened long ago in the past as time series forecasting. In these cases, relying on models with hundreds of millions of parameters is the only way to mitigate the inability of MLPs to leverage efficiently the spatial or temporal correlations in the input data. Such generic architectures tend to be highly inefficient due to their large number of redundant parameters and are notoriously difficult to train. In particular, the results fall short of expectations when applied to end-of-end autonomous driving or handwriting recognition tasks. They are superseded by more conventional methods such as *Support-Vector Machines* (SVM).

MLPs are especially suitable to parametrize control policies in robotics. The inputs are proprioceptive sensor data. They are very compact, assuming that the

feature extraction has been performed upstream for camera data. Thus, it does not suffer from the issue of high-dimensional input. Moreover, the complexity of the agent's behavior comes from the feedback mechanism rather than the policy itself. Indeed, a simple *Proportional-Integral-Derivative controller* (PID) already exhibits complex closed-loop dynamics while it corresponds to a single-layer MLP with the identity as activation function. For these reasons, policy networks are almost exclusively MLPs in *Reinforcement Learning* (RL) when the state is fully observable.

D.2 Residual Neural Network

Recent evidence reveals that network depth is of crucial importance. Indeed, deep networks naturally integrate more low-, mid- and high-level features as the number of layers increases. The limitation of vanishing or exploding gradient has been largely addressed by normalized initialization and intermediate normalization layers (Ioffe & Szegedy, 2015), which enable training networks with tens of layers. The leading results using classical FNNs all exploit depths of sixteen to thirty hidden layers (He et al., 2016). However, a degradation problem has been exposed for deeper networks: with the network depth increasing, accuracy gets saturated and then degrades rapidly. It is not caused by overfitting nor vanishing/exploding gradient but indicates that very deep FNNs are simply hard to train. Specifically, it suggests that solvers have more difficulties in approximating the identity using multiple nonlinear layers. This observation holds for *Convolutional Neural Networks* (CNNs) but also MLPs.

A few years back, residual learning was introduced to completely prevent such degradation. Strictly speaking, it is not a network architecture by itself but rather a modification of the connections between the layers of existing feedforward architectures. It simply consists in adding *shortcut connections* every few layers, also called *skip connections*. The shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers, as depicted in figure D.1. It does not add any extra parameter nor increase computational complexity, and the entire network can still be trained by *Stochastic Gradient-Descent* (SGD) with backpropagation. With the residual learning reformulation, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings, which is much easier to optimize that the original formulation. This seemingly simple trick is powerful enough to keep improving training accuracy monotonously up to a thousand layers (He et al., 2016). It was state-of-the-art for image classification and visual recognition tasks solely due to the extremely deep representations that residual learning enables.

D.3 Encoder-Decoder Architecture

Originally, Kramer (1991) proposed the encoder-decoder architecture by to tackle *unsupervised learning* problems. Unlike supervised learning, unsupervised learning deals with *unlabeled* training samples, which means that the dataset comprises only

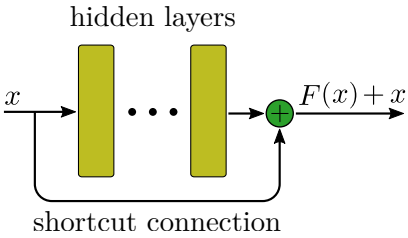


Figure D.1: Residual learning building block

inputs. Generally, the objective is to find patterns in the data, but its scope goes way beyond this. It is useful to identify clusters, but also to compute global coordinate charts (namely atlases) on low-dimensional manifolds that may be embedded in much higher-dimensional euclidean space, or to find geodesics on complex manifolds that may not even have closed-form expressions.

Such topological problems are ubiquitous in the robotics field, especially for trajectory planning problems. A generative neural network was trained in chapter 5 to represent the solutions to trajectory optimization problems as a function of a set of hyperparameters. It is convenient for picking the one trajectory having the desired hyperparameters but does not suit all needs. For example, let us consider a transition from a trajectory having given hyperparameters to another one. How to compute extrapolate the trajectory that is most likely to be stable? This question has been discussed in section 5.5.1. Next, let us assume the human is capable of moving the joints of the robot by his own motion and the robot is supposed to assist him to keep balance or compensate for any asymmetry of the gait. It involves computing the closest state on the manifold of stable trajectories given the current one.

In unsupervised learning, the network is trained to predict an output that matches the input. In principle, a basic FNN with uniform width should be able to do so, but it turns out that this kind of architecture is unable to exhibit a robust and compact representation of the input space. The idea is to force learning such a representation through the network architecture directly. The width of the layers of the encoders is gradually reduced to form a bottleneck. Its output has low dimensionality and is supposed to be a lossless compression of the input. The objective is to achieve the highest compression ratio by shrinking the size of the bottleneck as much as possible while still being able to accurately reconstruct the data. This low-dimension space is called *latent space* but is sometimes referred to by its interpretation, e.g. *context* for machine translation or features for facial recognition. The encoder is inherently robust to corrupted data as it naturally learns to leverage information redundancy from the input to compute the compressed representation. As a result, the encoder should always output the same encoding even after adding noise or partially damaging the input. For images, one may also expect to be able to blur the input, rescale it or even rotate it. Then, the decoder can be used to recover the original input. In general, the decoder has the same structure as the encoder, with layers in reverse order, as illustrated in figure D.2. The encoder and decoder are jointly trained. This

means that the decoder learns to reconstruct the input while the encoder is still refining its representation to reduce compression loss as much as possible. All it requires is minimizing the reconstruction error, as the capability of the decoder is conditioned by the quality of the encoding. In unsupervised learning, the encoder-decoder architecture is referred to as *auto-encoder*. The auto-encoder was initially introduced to perform non-linear *Principal Component Analysis* (PCA), which is a dimensionality reduction method, long before it became popular and generalized to supervised learning problems. Variants of the vanilla auto-encoder exist:

- *Regularized auto-encoders (Alain & Bengio, 2014)*: The objective is to extract features that capture the structure in the input distribution. Such property cannot be assessed directly but can be enforced through regularization. Indeed, one can demonstrate analytically that minimizing a particular form of regularized reconstruction error yields features that locally characterize the manifold of the data-generating density. Thus, this approach has nothing to do with the network architecture itself but rather concerns the training process and can be applied to any auto-encoder architecture. The rationale behind such regularization is that a good low-dimensional representation is supposed to be robust and stable to slight variations of input. Vincent et al. (2008) proposed first the *denoising auto-encoder*. The input is corrupted before being forwarded to the network, and the objective is to recover the true original input. Usually, it is corrupted by the addition of small isotropic Gaussian noise, but any other form of corruption is equally valid. Another approach is to add a penalty term that corresponds to the Frobenius norm of the gradient of the output with respect to the input. This results in a localized space contraction which in turn yields robust features, hence the name *contractive auto-encoder* (Rifai et al., 2011). This penalty explicitly carves a representation that captures the local directions of variation dictated by the data, corresponding to a lower-dimensional manifold, while invariant to orthogonal directions. Contractive auto-encoder tends to surpass denoising auto-encoders but are significantly more costly to evaluate due to the explicit gradient assessment. One can show that denoising auto-encoders with small corruption noise are similar to their contractive counterpart, but with contraction applied on the whole reconstruction function rather than just the encoder. In both cases, they effectively capture the *score*, namely the derivative of the log density with respect to the input, instead of the energy function as sometimes stated.
- *Variational auto-encoders (Kingma & Welling, 2014)*: Variational autoencoders are meant to compress the input information into a constrained multivariate distribution. Thus, the latent space is a mixture of distributions instead of a fixed vector. Unlike vanilla and regularized auto-encoders, the empirical reconstruct error is not minimized directly but rather indirectly in terms of probability. Variational autoencoders belong to the families of variational Bayesian methods. The input data is sampled from a prior distribution, and the auto-encoder is trained to minimize the *Kullback-Leibler* (KL) divergence between

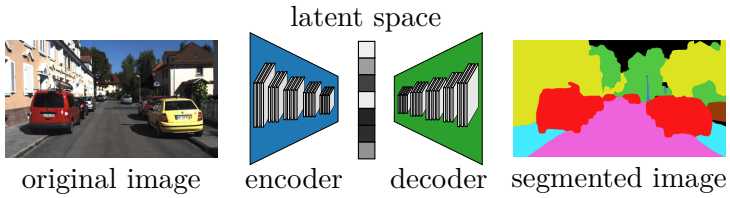


Figure D.2: Illustration of encoder-decoder architecture for image segmentation

the parametric posterior and the true posterior. In practice, a surrogate variational lower bound estimate is optimized instead of the true objective function to make computations tractable. This model scales well to large datasets.

Denoising and contractive auto-encoders are commonly used for learning robust encoding, while variational auto-encoders are more suitable for generation tasks that consist in randomly generating new data looking similar to the training samples from any input. For example, Goodfellow et al. (2014) used variational auto-encoders to generate realistic fake celebrity faces, before being outperformed by *Generative Adversarial Nets* (GAN) on high-resolution images. GAN is not a network architecture but a learning process for encoder-decoder especially designed for this kind of generative task. Instead of training them simultaneously to collaborate, they are trained alternatively to compete against each over. The encoder (called discriminators) must determine if some data is real or fake, while the decoder (called generator) must generate fake outputs that are realistic often to fool the discriminator. Although very powerful, this method is notoriously very difficult to tune to avoid the domination of one of the two networks. The encoder-decoder architecture applies to many problems, including dimensionality reduction, clustering, noise removal, data completion, data visualization, anomaly detection, and feature extraction. Examples of applications are facial recognition, image segmentation, and 3D object reconstructions from 2D pictures. The encoder-decoder architecture is also suitable for problems where the inputs and/or the outputs are variable-length. The encoder is responsible for compressing the input into a fixed-length vector, then the decoder can expend it back independently of the original input. One famous example is machine translation. Finally, pre-trained encoders and/or decoders are often used as part of more complex architectures to provide compact intermediate representations of the input data. This approach can be encountered in classification, deep generative or discriminative models, and control policies for legged robots (Miki et al., 2022).

D.4 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a class of ANN especially tailored to process or generate sequences. They rely on step-by-step sequential evaluation and maintain a hidden state that serves as a memory of the past. At each evaluation step, the hidden state is updated to aggregate new information and forget the previous events that became irrelevant. Different configurations are possible:

- *One-to-many*: A initial state is used to generate a whole sequence by looping recursively on itself. At each generation step, the preceding output is forwarded as input without any extrinsic information. It can be used for music generation.
- *Many-to-one*: A whole input sequence is processed sequentially but only the final output is preserved. It is suitable for sentiment classification.
- *Many-to-many*: A whole input sequence is processed sequentially and the whole output sequence is of interest.

Advanced architectures combine several configurations. In machine translation, state-of-the-art results are achieved by two networks with encoder-decoder architecture. A first RNN of type many-to-one encodes the context, which is then forwarded to another RNN of type one-to-many that decodes the whole translated sentence.

One of the main challenges for RNNs is being able to learn long-term dependencies or correlations. For instance, consider trying to predict the last word in the text "I grew up in France [...] I speak fluent". It is clear from the context that the next word is probably "French", but the gap between the relevant piece of information and the point where it is needed may be very large. Any FNN evaluated sequentially and maintaining a hidden state could be used as RNN. Such architecture is very generic but suffers from the vanishing gradient problem. Indeed, the last output depends on all the previous ones recursively, so that the former can be interpreted as the output of a single virtual neural network with up to thousands of layers sharing parameters. This virtual neural network is said to be the *unfolded representation* of the RNN. Backpropagation of the gradient from the output to the input through the virtual layers induces the same training issues mentioned previously for a single extremely deep neural network. *Long Short-Term Memory* (LSTM) is a network architecture that has been designed by Hochreiter and Schmidhuber (1997) to prevent vanishing gradient. It does so thanks to an additional *cell state* that goes all the way through the network without being altered by any layer directly, so that information can flow backward to the input no matter how long the sequence is. Then, storing and forgetting capabilities are added via a special technique called *gating mechanism*. Gates are a way to selectively let information pass through based on some excitatory signal. This approach is common to any RNN architecture. For LSTMs, the excitatory signals of the gates are the concatenation of the current elements of the sequence with the hidden state. First, the *forget gate* filters out the cell state. Then, a candidate new cell state is computed using a single layer with hyperbolic tangent activation, itself filtered by the *input gate* before summing it up with the previously filtered cell state. Finally, the updated cell state is used to compute the new hidden state using a single layer following by the filtering with the *output gate*. There exists many variants of LSTMs. Its behavior is summarized in figure D.3. The way the excitatory signals of the gates are computed varies from one architecture to another, sometimes gates are coupled together or completely disappear. In the end, they are all very similar in terms of architecture and overall performance (Greff et al., 2017). Yet, *Gated Recurrent Unit* (GRU) stands out more than the others as it merges the cell state and the hidden state as a single signal (Cho et al., 2014). Their performance was

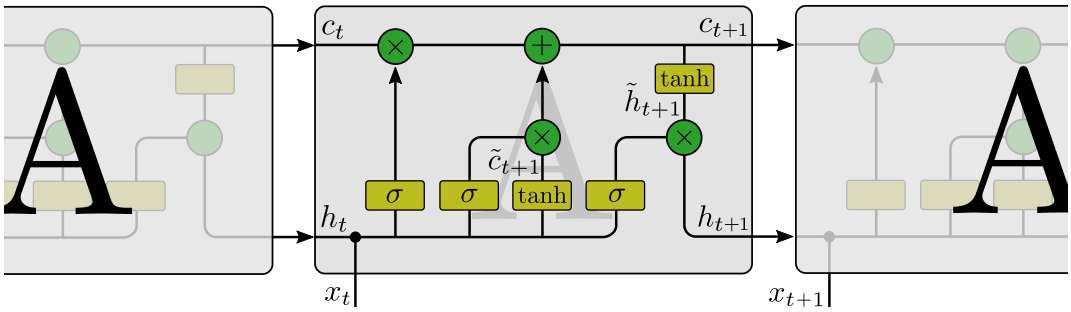


Figure D.3: Description of Long Short-Term Memory network

found to be similar to that of LSTMs while being computationally more tractable.

In few years, they achieved state-of-the-art results for many difficult problems, including handwriting recognition and generation, language modeling and translation, acoustic modeling of speech, speech synthesis, protein secondary structure prediction, analysis of audio and video data. However, they have been surpassed in many domains by non-recurrent architectures. One of the main issues of RNNs is a direct consequence of its own strength, namely sequential evaluation. Not being able to evaluate all input at once is slowing down both the training and evaluation processes as it is not possible to exploit the computational resource to the full extent. Moreover, there is no explicit hierarchy of information, which means that such a structure must arise during training instead of being explicitly enforced by the architecture itself. This is limiting the performance of RNNs on complex multiscale problems for which the hierarchy may be unclear and thereby hard to learn.

D.5 Convolution Neural Network

RNNs can deal with short- and long-term dependencies and can process or generate variable-length sequences. However, this property requires their evaluation to be sequential, therefore costly to evaluate. Next, it is difficult to improve upon the vanilla architecture to push performances further. Finally, fixed-length sequences can approximate reasonably well by unbounded sequences in many problems, since the effect of elements far away in the sequence at the time being tends to vanish. If so, then a CNN can be used in place of a RNN. CNNs have already been introduced in section 3.1.2. It is a special type of FNN with translational invariance property that enables dealing with high-dimensional data that is either spatially or temporally correlated. Groups of layers form filters that extract features at different scales. Each layer has fewer parameters than their equivalent dense representation using MLPs. Combined with a residual architecture, one can stack many of them to extract more features without overfitting the data. CNNs have led to a series of breakthroughs in image classification. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

D.6 Attention-based Network

Historically, CNNs were surpassing RNNs in many problems, notably language understanding tasks. They can exploit long-term dependencies, but doing so requires very deep networks because it goes against their translational invariance property. What truly matters is the *context*, which is defined by the sum of information contained in the input irrespective of the relative positioning of every single element of data. The attention mechanism has been introduced on that basis. It substitutes permutation invariance with translational invariance to remove any preference for data locally. In practice, the network would be equally likely to extract features leveraging short- or long-term dependencies whatever the depth of the network. However, the relative ordering of the data carries some information in many problems, which is lost using such architecture. The original data are usually augmented with a positional encoding to make up this shortfall, e.g. Fourier features (Tancik et al., 2020).

The attention mechanism is commonly referred to as QKV attention as it combines *query*, *key*, and *value* vectors. Each of these vectors belongs to a different low-dimensional embedding encoded by a single fully-connected linear layer. It is called *self-attention* if the same input signal is passed to all these networks, *cross-attention* if the query network is fed by a different input. In general terms, a modality refers to one specific channel of information. Self-attention considers on intra-modality relationships. This mechanism is at the heart of co-reference resolution – the task of finding all expressions that refer to the same entity – based on the past sentence for machine translation. It only concerns a single modality in the latter case, but nothing prevents gathering several modalities at once. It is generally the case when the data are augmented with positional encoding. Cross-attention enables pulling apart those modalities to restrict focus on the inter-modality relationship between them. It is computationally cheaper as the intra-modality relationship is completely ignored. Cross-attention is used to combine efficiently RGB pixel arrays with height maps to enhance image segmentation.

The QKV attention was originally referred to as *Scaled Dot-Product* because it computes the dot product of all queries with all keys and gathers all possible combinations in a matrix called *attention map*. Optionally, the attention map can be masked to enforce causality if relevant. The numbers of queries and key/value pairs are equal for self-attention but different for cross-attention. As one can expect, the number of outputs is always equal to the number of queries. Similarly, the query and key low-dimensional embeddings have to match, but the value embedding may differ. Let $Q \in \mathbb{R}^{n \times e_k}, K \in \mathbb{R}^{m \times e_k}, V \in \mathbb{R}^{m \times e_v}$ be queried, key and values matrices respectively, where E_k, e_v is the size of the query/key and value embeddings, and m, n is the length of the source and target sequences. The operation performed by the QKV attention block is

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{e_k}}\right)V, \tag{D.1}$$

where $\text{Attention}(Q, K, V) \in \mathbb{R}^{n \times e_v}$. This operation has approximate complexity

$\mathcal{O}(mn)$ as it involves two matrix multiplications with matrices of large dimensions.

Transformers (Vaswani et al., 2017) is a type of attention-based network specialized for machine translation. It stacks several self-attention blocks together in an encoder-decoder architecture. Concurrently, Gehring et al. (2017) used a similar attention mechanism to improve the accuracy of CNNs used for machine translation. It turns out the transformer architecture outperforms both state-of-the-art RNNs and attention-based CNNs for machine understanding tasks while being cheaper to evaluate and having fewer parameters. Perceivers (Jaegle et al., 2021) generalizes the Transformer architecture to heterogeneous data that can be possibly anything, e.g. videos, sounds, or texts. Self-attention would in untractable in this case as the computational complexity scaled quadratically with the length of the input sequence (i.e. the number of pixels or audio samples). To circumvent this limitation, cross-attention is used instead. The query embedding is a low-dimensional latent space $n \ll m$, so that the computational complexity is now linear with respect to the input size. Perceivers achieve state-of-the-art results on classification problems without any convolution layer, which is considered a breakthrough.

E Classical Approaches in Reinforcement Learning

E.1 Dynamic Programming

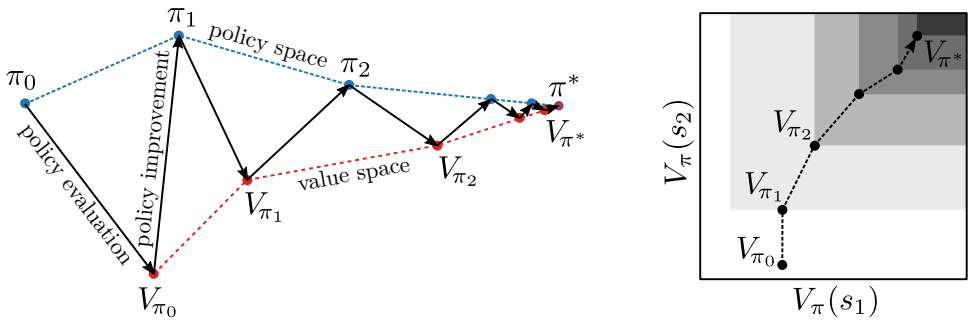
This method can only be applied if the model is perfectly known and the reward is dense enough. Although it is untrue in *Reinforcement Learning* (RL), it is still important to review the method. First, it helps to understand early algorithms that are at the root of state-of-the-art off-policy algorithms. Secondly, it can still be applied after learning an approximate model of the world. Several algorithms are based on this principle, e.g. World Models (Ha & Schmidhuber, 2018) or *Probabilistic Inference for Learning COntrol* (PILCO) (Deisenroth & Rasmussen, 2011). It is an iterative algorithm that alternates between *policy evaluation* and *policy improvement*. This procedure is called *Generalized Policy Iteration* (GPI) and illustrated in figure E.1: the state-value function for the current policy is approximated iteratively to be closer to the true one, while the policy is improved to approach optimality. The way to update the value function is specific to every method and is called the *backup* rule. Let us assume the state and action spaces are finite for simplicity. If not, then quantization may be an option.

- **Policy evaluation**

  It is to compute the state-value V_π for the current policy π. From Bellman equation (3.32), it yields

  $$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a)(R(s,a) + \gamma V_\pi(s')). \tag{E.1}$$

  It is a linear system with $|\mathcal{S}|$ unknowns, namely $V_\pi(s), \forall s \in \mathcal{S}$. It should be invertible and solvable in closed form, but it would be computationally costly. A more efficient method consists in approximating the exact solution iteratively. The value function is initialized arbitrarily for any state $s \in \mathcal{S}$ except the terminal state, if any, that must be given value 0. The approximate value function V_k is updated repeatedly according to equation (E.1). The value function V_π is clearly a fixed point for this backup rule. The sequence can be shown in general to converge as long as V_π exists. The process stops once the norm of the residual improvement is small enough, i.e. $\|V_{k+1} - V_k\|_\infty < \epsilon$.



(a) Generalized Policy Iteration (b) Evolution of the value function

Figure E.1: (a) Generalized Policy Iteration: sequence of policy updates and value computation that converges asymptotically to an optimal solution. (b) Evolution of the value function across the value space for a state space with two states $\mathcal{S} = \{s_1, s_2\}$. Each rectangle corresponds to the region of the value space containing the value function at a given iteration. One iteration is the application of policy improvement and evaluation successively. The value for each state increases monotonically.

- **Policy improvement**

  It generates a better policy by acting greedily according to equation (3.31) after computing the action-value from the state-value using Bellman equation (3.33),

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)(R(s,a) + \gamma V_\pi(s')). \tag{E.2}$$

The proof of convergence and monotonic improvement is very similar to the one that was briefly outlined for purely value-based on-policy methods.

E.2 Monte-Carlo Methods

If the model is unknown, it is not possible to solve analytically the policy evaluation problem, also called *prediction problem*. And even if it was, it may be untractable if the cardinality of the state space is very high. *Monte-Carlo* (MC) methods enabled compute an approximate solution to this problem without involving a model or enumerating all possible states anymore. It simply consists in replacing in computations the expectation over all possible trajectories under a given policy by the empirical mean over a batch of episodes. It requires collecting complete trajectories, so all the episodes must terminate, which is a significant limitation.

Let us assume the state and action spaces are finite and consider a single episode of the length T for simplicity. The empirical value function is

$$\hat{V}_\pi(s) = \frac{\sum_{t=1}^{T} \mathbb{1}_{s_t}(s) G_t}{\sum_{t=1}^{T} \mathbb{1}_{s_t}(s)}, \tag{E.3}$$

This formulation is referred to as the *every-visit* MC method since it averages the future return every time the state has been encountered. Alternatively, one may take into account only the first time it occurs per episode, which is the *first-visit* MC method.

It is straightforward to extend this formulation to the action-value function by counting state-action pairs. Once the action-value function has been estimated, the policy is improved just as in *Dynamic Programming* (DP). The resulting GPI procedure except for the backup rule. This basic approach is never used as is in practice because it has a very high variance, so an extremely large number of samples is needed to approximate reliably the action-value function. The high variance comes from the observed future returns in computations. It can be explained in several ways. First, it involves the integration of the current state over a horizon. This is problematic as the variance of the mapping between the initial and final state tends to grow exponentially with the number of steps, no matter if the world is fully deterministic. This issue is known as *deterministic chaos* (Ott, 2002). At the limit, rounding errors in the initial condition or different instruction sets at the CPU-level can yield diverging outcomes, making long-term prediction impossible. It is systematic for any dynamical system as long as it does not have any attractive fixed points or closed orbits, even for fairly simple models such as the classical double-rod pendulum without friction. Shooting methods for solving *Optimal Control Problems* (OCPs) are facing the exact same issue. Secondly, the policy is at least partially stochastic in practice to promote exploration. The outcome may be different for a single transition step if the transition function is very sensitive or discontinuous. It may result in dramatically different rewards for the same action on average. Typically, it is the case for legged locomotion, where contact is specified by unilateral kinematic constraints, as explained in appendix C.2.2. Working on individual transition steps instead of the complete trajectories would significantly alleviate this issue while relieving the constraint of episode termination. The *Temporal-Difference* (TD) learning enables doing so and has been a milestone in RL.

E.3 Temporal-Difference Learning

TD learning alternates between policy evaluation and policy improvement as in all the previous methods. Similar to MC methods, it is model-free and learns from collected data, but it relies on yet another backup rule to update the value function. The central idea is *bootstrapping*: the computations during policy evaluation leverages the previous estimates rather than exclusively relying on collected data. It enables approximating the return and thereby the value function from incomplete episodes, which is a significant improvement upon MC methods. It is no longer necessary to track the episodes up to termination, and they do not even have to terminate at all.

The return $\hat{G}_t$ can be estimated from the Bellman equation (3.32),

$$\hat{G}_t = r_t + \gamma V(s_{t+1}). \tag{E.4}$$

This term is called *TD target*. It matches the true return in expectation if V is equal to the value function associated with the current policy V_π. As for MC methods, the value function is updated based on an estimate of the return, but the TD target is used in place of the observed return. Moreover, a ratio $\alpha \in [0, 1]$ is introduced to avoid forgetting completely the previous value every time the state is visited. It takes the form of an update rule applied to every collected transition step independently,

$$V(s_t) \leftarrow (1 - \alpha)V(s_t) + \alpha G_t = V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)). \tag{E.5}$$

This update rule can be interpreted as a recursive approximation of the moving average that does not take into account the number of past samples, so it computes a good approximation of the value function if the number of samples is large enough. The hyperparameter α is called the learning rate and must be tuned manually for each specific problem. It can be fixed, scheduled, or adaptive depending on the method. Higher values give more weight to new data by shortening the window of the moving average but increase the variance. Conversely, lower values rely more on the previous estimate but increase the bias. It is straightforward to derive the equivalent formula for the action-value,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)). \tag{E.6}$$

The algorithm that results from applying the GPI procedure using this backup rule is *State-Action-Reward-State-Action* (SARSA). It is on-policy as it estimates the value function associated with the current policy, just like DP and MC methods. The following steps are repeated until convergence:

1. Get into the next state s_{t+1} and receive a reward r_t.
2. Choose the next action a_{t+1} greedily: $a_{t+1} = \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$.
   Update the Q-value function according to equation (E.6).
3. Take previously selected action a_{t+1}.

TD targets can be replaced by the ones corresponding to the Bellman equations for the optimal value functions. According to equation (3.34) it yields,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)). \tag{E.7}$$

As expected, this backup rule is completely independent of the current policy, so it is not necessary to choose the action and take it separately. The resulting off-policy algorithm is Q-learning. The steps are slightly simplified versus SARSA:

1. Get into the next state s_{t+1} and receive a reward r_t.
2. Take the next action a_{t+1} greedily: $a_{t+1} = \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a)$.
   Update the Q-value function according to equation (E.7).

Even though the action taken by the agent and the one used to estimate the optimal action-value appears to be the same in this basic algorithm, there is no relationship

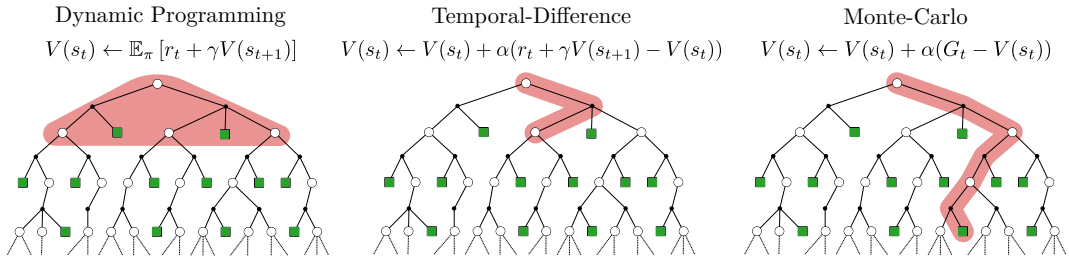


Figure E.2: Comparison of the backup strategies of Dynamic Programming, Temporal-Difference, and Monte-Carlo for the state value function.

between them. The action leading to the maximal Q-value at the next state s_{t+1} is always used to update the estimate of the optimal action-value, but there is no such restriction for the actual action. They may very well be different, which would be the case if the policy is stochastic, e.g. using ϵ-greedy to pick the action.

The computation of the TD target in equation (E.4) relies on one-step look-ahead Bellman equation. It is a low-variance estimate of the return $\hat{G}_t$ compared to MC methods as it avoids the compounding of uncertainties. However, it is biased since the future return beyond the next step is approximated using an estimate of the value function that may be off. On the contrary, MC methods have the advantage to be unbiased since it only involves collected data. Having to choose between low variance or bias is a common dilemma. How much to favor one or the other is problem-specific, so it would be advantageous to have a more fine-grained control than having to choose between two extreme cases. It is straightforward to extend one-step TD learning to take multiple steps. It bridges the gap with MC methods and enables a trade-off between bias and variance (cf. appendix E.7.1). For n steps, it gives

$$\hat{G}_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n}). \tag{E.8}$$

It is preferable to weigh all possible n-step TD targets instead of picking only one of them to get an even better estimate. The weights must decay exponentially with the number of steps n, in consistency with the discount factor for future rewards when computing the return. It is explained by the deterministic chaos theory already mentioned: the confidence about future predictions decreases exponentially over time because of the compounding of uncertainties. Besides, relating a future reward to a specific action becomes increasingly difficult as these events get more distant since all the actions taken in the meantime would affect this exact same reward. This weighted sum of many n-step returns is called λ-return (Sutton & Barto, 2018), $\hat{G}_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{G}_t^{(n)}$, where $\lambda \in [0, 1]$ is the decay factor. The normalization factor $1 - \lambda$ ensures that all weights to sum up to 1.

TD learning that adopts λ-return for computing the TD target is labeled as TD(λ). The decay factor λ is a fixed hyperparameter that must be tuned manually for each problem. A higher decay factor increases the variance but reduces the

bias. Its optimal value depends on how much distance states can be trusted. The naive one-step look-ahead variant is the special case TD(0), while TD(1) corresponds to the every-visit MC method applied on incomplete episodes. Mathematically, it is equivalent to another mechanism called *eligibility traces* (Rummery & Niranjan, 1994). Roughly speaking, the eligibility trace is a record for each episode of the occurrence of recent state-action pairs that vanishes exponentially over time by a factor $\lambda \in [0, 1]$ called *trace decay*. It is then used for computing the TD target by weighting state-action pairs proportionally to how often they have been visited recently. The rationale is to assign credit or blame only the states that are supposed to be responsible for the TD error. The λ-return is more suitable for theoretical analysis while the eligibility trace is convenient for practical implementation. The three different backup strategies are summarized in figure E.2.

E.4 Deep Q-Network

If the state and action spaces are finite, then it is theoretically possible to store in tables the optimal value Q^* for every possible state-action pair. However, the number of pairs grows exponentially with the dimensionality, and it quickly becomes untractable as state and action space grow larger. In many problems, the state space is continuous and this tabular approach is not even applicable.

The naive approach to overcome this limitation is to replace the value function estimate by a function approximation with parameters ϕ. The latter generally takes the form of a neural network. For the action-value in particular, it is referred to as *Q-network* and denoted Q_ϕ. The use of a neural network should be beneficial overall, as it allows for more efficient exploitation of the collected data. Specifically, they leverage similarities through its generalization ability: updating the parameters ϕ to improve the prediction for any given state-action pair would inevitably drag the close-by ones in the same direction. This makes sense at first since similar state-action pairs tend to have similar values under sufficient regularity assumption of the underlying system dynamics. Yet, the whole state-action space is impacted to some extent that is hard to predict. This may also be detrimental from time to time as it causes aliasing and the regularity assumption must not reflect reality.

In principle, it should be possible to mitigate this issue by increasing the expressiveness of the function approximation, but it would be prone to overfitting. Overfitting creates artifacts, especially for states that have been not encountered recently (even if surrounded by dense regions of training samples). Although of minor importance for state-action pairs considered individually, these artifacts induce an *overestimation bias* that has a dramatic effect on the backup rule (Lan et al., 2020). This is due to the TD target involving the maximum over the predicted action-value for a given state, which is very sensitive to outliers. Since the predicted action-value will probably be higher than the true one for one or more of the action, the TD target is likely to be skewed toward an overestimate. For example, even unbiased estimates

$Q_\phi(s,a)$ will vary due to stochasticity. According to Jensen inequality,

$$\mathbb{E}[\max_{a\in\mathcal{A}} Q_\phi(s,a)] \geq \max_{a\in\mathcal{A}} \mathbb{E}[Q_\phi(s,a)] = \max_{a\in\mathcal{A}} Q^*(s,a) \tag{E.9}$$

Consequently, Q-learning suffers from instability and divergence due to the combination off-policy learning, bootstrapping, and nonlinear function approximation concurrently. This issue is known as the *deadly triad* (Sutton & Barto, 2018). *Deep Q Network* (DQN) (Mnih et al., 2015) greatly improves the stability of Q-learning with two mechanisms:

- **Experience Replay**

  All transition steps $e_t = \{t, s_t, a_t, r_t, s_{t+1}\}$ are stored in a *replay buffer* $\mathcal{D} = \{e_1, \dots, e_t\}$. The Q-value is updated with samples from the replay buffer. They are drawn at random, hence they are used many times during the whole training process. It improves sample efficiency, but it also removes correlations in observation sequences and smooths out the effect of the distributional shift consequent to policy improvements.

- **Delayed Target Update**

  In Q-learning, the Q-value is optimized toward targets that depend on the Q-value itself. This direct coupling creates short-term oscillations that make learning unstable. To break it, the Q-value involve in the TD targets is separated from the current estimate. This *target Q-value* $\hat{Q}$ is kept frozen and only updated periodically to match the current estimate Q_ϕ every few transition steps. This way, a delay is introduced between the update of the Q-value and its effect on the TD targets. It significantly weakens their coupling, while still providing a sensible approximation of the Q-value in computations. The current estimate is still used to compute the actions taken by the agent.

The Q-network Q_ϕ cannot be optimized directly by applying successfully the update rule specified by equation (E.7) in Q-learning. It is replaced by the minimization of the residual error between the TD targets and the corresponding predictions of the Q-network with respect to its parameters ϕ. Formally, a batch $\mathcal{B}$ of N transition steps $\{e_i\}_{i=0}^N$ is uniformly sampled from the replay buffer $\mathcal{D}$, i.e. $\mathcal{B} \sim U(\mathcal{D})$. The loss $\mathcal{L}(\phi)$ is the empirical mean of the residual error over the whole batch,

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_{e_t\in\mathcal{B}} \left(r_t + \gamma \max_{a\in\mathcal{A}} \hat{Q}(s_{t+1}, a) - Q_\phi(s_t, a_t) \right)^2. \tag{E.10}$$

Then, a single gradient descent step with learning rate α is applied,

$$\phi \leftarrow \phi - \alpha \nabla \mathcal{L}(\phi). \tag{E.11}$$

Since a function approximation has limited expressiveness, the error cannot be canceled out completely. In this scenario, it is critical to encourage evenly distributed

error over the whole state-action space to reduce overestimation. This is done by minimizing the squared residual error rather than the absolute one. Moreover, the entire batch must be considered at once instead of doing many successive updates as the value for any given state-action pair influences all the others. This was not the case in tabular Q-learning. Note that introducing a non-linear function approximation means that convergence is no longer guaranteed in contrast to previous methods.

Some transition steps in the replay buffer may be completely off, either because such states are never encountered anymore, or because the value associated with the state-action pairs is so low that the agent would no longer take such actions. In both cases, these samples are irrelevant in the update of the estimated optimal value Function. This is not a big deal if the estimate is tabular as the value for each state-action pair is updated independently, but this is potentially harmful if it is a function approximation. Specifically, this may steer the whole approximation in a bad direction regarding the current policy if too many irrelevant samples are considered. For legged locomotion tasks, the robot would not even be able to bear its own weight in the first episodes. It has nothing to do with the actual task, and including those samples for the update of the parameters would just make everything worse once the robot can stand up.

E.5 Policy Gradient Methods

E.5.1 Policy Gradient Theorems

All the previous methods are value-based: they aim to learn the action-value function and then select actions accordingly. For continuous state and action spaces, they are also policy-based, which gives rise to actor-critic algorithms. Both the action-value and the policy are function approximations, and the latter is trained to output the action maximizing the action-value for any state. Vanilla policy gradient methods are only policy-based: the policy is a function approximation π_θ trained to maximize directly the expected return in equation (3.22). At every training iteration, trajectories are collected under the current policy π_θ, then the parameters θ are updated by doing a single gradient ascent step. The core component is the computation of the gradient of the expected return with respect to the parameters θ.

Likelihood Ratio Estimate

As a reminder, the *Markov Decision Process* (MDP) is supposed to be stationary. In addition, the Markov chain resulting from taking actions according to a given policy π is sometimes assumed to be *ergodic*, i.e. irreducible and *aperiodic*. Mathematically, this means that there exists a number N such that any state can be reached from any other state in at most N steps whatever the current time. The *Fundamental Theorem of Markov Chains* states that if a Markov chain is ergodic then it has a unique stationary distribution $\bar{\rho}_\pi$, and the probability to end up in some state s wherever you started is equal to this distribution as the number of steps in between approaches

infinity. Let us denote $\rho_\pi^{(n)}(s \to s')$ the probability to reach the state s' from state s after n steps, $\mathbb{P}(s_n = s'|s_0 = s)$. Then $\lim_{n\to\infty} \rho_\pi^{(n)}(s \to s') = \bar{\rho}_\pi(s'), \forall s, s' \in \mathcal{S}$. The ergodicity property is not essential and never required. The unnormalized *discounted state distribution* ρ_π is the discounted counterpart of the stationary distribution, namely $\rho_\pi(s') = \sum_{k=0}^{\infty} \gamma^k \rho_\pi^{(k)}(s \to s')$. This quantity has no physical meaning but appears in the computation of the gradient.

The *Policy Gradient Theorem* (Sutton et al., 1999) enables computing the gradient of the expected return without knowledge of the generating distribution. This formula is at the heart of all policy gradient methods. The proof is a bit lengthy but insightful, so it is worth carrying it out in detail. Let us present first the *Expected Grad-Log-Prob* (EGLP) lemma and a corollary as the proof relies on it.

Lemma 9 (Expected Grad-Log-Prob)**.** *Let P_θ be probability distribution over a random variable x with parameters θ. Then,*

$$\mathop{\mathbb{E}}_{x\sim P_\theta} [\nabla_\theta \log P_\theta(x)] = 0 \tag{E.12}$$

Corollary 9.1. *Let ψ be a function that depends on the state up to time t and the action up to time $t-1$. Then,*

$$\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} [\psi(s_{:t}, a_{:t-1}) \nabla_\theta \log \pi_\theta(a_t|s_t)] = 0, \quad \forall t \geq 1 \tag{E.13}$$

Proof. First, the *law of iterated expectation* is split the expectation over complete trajectories into two parts. Then it is a direct application of the EGLP lemma.

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\tau\sim\pi_\theta} [\psi(s_{:t}, a_{:t-1}) \nabla_\theta \log \pi_\theta(a_t|s_t)] \\
&\qquad = \mathop{\mathbb{E}}_{s_{:t}, a_{:t-1}\sim\pi_\theta} \left[\mathop{\mathbb{E}}_{s_{t:}, a_{t-1:}\sim\pi_\theta} [\psi(s_{:t}, a_{:t-1}) \nabla_\theta \log \pi_\theta(a_t|s_t) | s_{:t}, a_{:t-1}] \right] \\
&\qquad = \mathop{\mathbb{E}}_{s_{:t}, a_{:t-1}\sim\pi_\theta} \Big[\psi(s_{:t}, a_{:t-1}) \underbrace{\mathop{\mathbb{E}}_{a_t\sim\pi_\theta} [\nabla_\theta \log \pi_\theta(a_t|s_t)]}_{=0 \text{ (EGLP)}} \Big] \\
&\qquad = 0
\end{aligned}$$

■

Theorem 10 (Policy Gradient Theorem)**.** *Let $\mathcal{M} = \{\mathcal{S}, \mathcal{O}, \mathcal{A}, P, R, O, \rho_0, \gamma\}$ be an infinite horizon stationary MDP with discounted reward. The gradient of the return $R(\tau)$ in expectation over the distribution of trajectories τ induced by policy π_θ with parameters θ is given by*

$$\nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} [R(\tau)] = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} \left[\sum_{t=0}^{\infty} R(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \tag{E.14}$$

$$= \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ a\sim\pi_\theta}} [Q_\pi(s,a) \nabla_\theta \log \pi_\theta(a|s)], \tag{E.15}$$

where $Q_\pi(s,a)$ is the action-value function and ρ_π is the unnormalized discounted state distribution under the policy π.

Proof. The expected return $J(\pi)$ is reformulated to replace the return by the value-function that is easier to manipulate,

$$J(\pi) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] = \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ a\sim\pi_\theta}}[R(s,a)] = \mathop{\mathbb{E}}_{s\sim\rho_0}[V_\pi(s)]\,.$$

The first step to differentiate this expression is to compute the gradient of the value function $V_\pi(s)$ with respect to the parameters θ. Equation (3.27) is used to substitute the value function $V_\pi(s)$ for the action-value $Q_\pi(s,a)$,

$$\nabla_\theta V_\pi(s) = \nabla_\theta \mathop{\mathbb{E}}_{a\sim\pi_\theta}[Q_\pi(s,a)]\,.$$

It is possible to move directly the operator ∇_θ inside the inner expectation, but it introduces an additional term because of the dependency on the policy. It yields,

$$\nabla_\theta V_\pi(s) = \underbrace{\int_{a\in\mathcal{A}} Q_\pi(s,a)\nabla_\theta\pi(a|s)\,\mathrm{d}a}_{\phi_\pi(s)} + \mathop{\mathbb{E}}_{a\sim\pi}[\nabla_\theta Q_\pi(s,a)]\,.$$

Bellman equation (3.33) is injected to make the value function appears on both side,

$$\nabla_\theta V_\pi(s) = \phi_\pi(s) + \gamma \mathop{\mathbb{E}}_{\substack{s'\sim P \\ a\sim\pi}}\left[\nabla_\theta V_\pi(s')\right].$$

This recursive representation will be helpful to derive a closed form. The idea is to go through more intermediary states, but it is hard to do so with the current formulation. The trick is to apply the definition of the expectation for a continuous domain and to realize that $\rho_\pi^{(1)}(s\to s') = \mathbb{P}(s_{t+1}=s'|s_t=s) = \mathop{\mathbb{E}}_{a\sim\pi}[P(s'|s,a)]$:

$$\begin{aligned}
\nabla_\theta V_\pi(s) &= \phi_\pi(s) + \gamma\int_{s'\in\mathcal{S}} \nabla_\theta V_\pi(s')\int_{a\in\mathcal{A}} \pi(a|s)P(s'|s,a)\,\mathrm{d}a\,\mathrm{d}s' \\
&= \phi_\pi(s) + \gamma\int_{s'\in\mathcal{S}} \nabla_\theta V_\pi(s')\mathop{\mathbb{E}}_{a\sim\pi}\left[P(s'|s,a)\right]\mathrm{d}s' \\
&= \phi_\pi(s) + \gamma\int_{s'\in\mathcal{S}} \rho_\pi^{(1)}(s\to s')\nabla_\theta V_\pi(s')\,\mathrm{d}s'
\end{aligned}$$

This formula can be unrolled to transition from the state s to any state after any number of steps infinitely by summing up the visitation probabilities using the identities $\int_{s'\in\mathcal{S}} \rho_\pi^{(n)}(s\to s')\rho_\pi^{(1)}(s'\to s'') = \rho_\pi^{(n+1)}(s\to s'')$ and $\rho_\pi^{(0)}(s\to s') = \mathbb{1}_{\{s\}}(s')$:

$$\begin{aligned}
\nabla_\theta V_\pi(s) &= \phi_\pi(s) + \gamma\int_{s'\in\mathcal{S}} \rho_\pi^{(1)}(s\to s')\left\{\phi_\pi(s') + \gamma\int_{s''\in\mathcal{S}} \rho_\pi^{(1)}(s'\to s'')\nabla_\theta V_\pi(s'')\,\mathrm{d}s''\right\}\mathrm{d}s' \\
&= \phi_\pi(s) + \gamma\int_{s'\in\mathcal{S}} \rho_\pi^{(1)}(s\to s')\phi_\pi(s')\,\mathrm{d}s' + \gamma^2\int_{s''\in\mathcal{S}} \rho_\pi^{(2)}(s\to s'')\nabla_\theta V_\pi(s'')\,\mathrm{d}s''
\end{aligned}$$

$$\cdots = \sum_{k=0}^{\infty} \gamma^k \int_{s' \in \mathcal{S}} \rho_\pi^{(k)}(s \to s') \phi_\pi(s') \, \mathrm{d}s' = \int_{s' \in \mathcal{S}} \phi_\pi(s') \underbrace{\sum_{k=0}^{\infty} \gamma^k \rho_\pi^{(k)}(s \to s') \, \mathrm{d}s'}_{\eta_\pi(s'|s)}$$

$\eta_\pi(s'|s)$ can be interpreted as the expected discounted visitation of state s' during a single episode starting in state s. The gradient of the objective function is given by:

$$\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [R(\tau)] &= \mathop{\mathbb{E}}_{s \sim \rho_0} [\nabla_\theta V_\pi(s)] \\
&= \int_{s \in \mathcal{S}} \int_{s' \in \mathcal{S}} \phi_\pi(s') \eta_\pi(s'|s) \rho_0(s) \, \mathrm{d}s' \, \mathrm{d}s \\
&= \int_{s' \in \mathcal{S}} \phi_\pi(s') \underbrace{\int_{s \in \mathcal{S}} \eta_\pi(s'|s) \rho_0(s) \, \mathrm{d}s}_{\rho_\pi(s')} \mathrm{d}s' \\
&= \mathop{\mathbb{E}}_{s \sim \rho_\pi} [\phi_\pi(s)]
\end{aligned}$$

$\rho_\pi(s')$ denotes unnormalized discounted state distribution. It is impossible to evaluate $\phi_\pi(s)$ directly, but it can be rewritten as the expectation over the policy itself by noticing $\nabla_\theta \pi_\theta(a|s) = \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)$:

$$\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [R(\tau)] &= \mathop{\mathbb{E}}_{s \sim \rho_\pi} \left[\int_{a \in \mathcal{A}} Q_\pi(s, a) \nabla_\theta \pi(a|s) \, \mathrm{d}a \right] \\
&= \mathop{\mathbb{E}}_{s \sim \rho_\pi} \left[\int_{a \in \mathcal{A}} Q_\pi(s, a) \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) \, \mathrm{d}a \right] \\
&= \mathop{\mathbb{E}}_{s \sim \rho_\pi} \left[\mathop{\mathbb{E}}_{a \sim \pi_\theta} [Q_\pi(s, a) \nabla_\theta \log \pi_\theta(a|s)] \right] \\
&= \sum_{t=0}^{\infty} \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [\gamma^t Q_\pi(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] \qquad \text{(E.16)}
\end{aligned}$$

The sum operator can be moved inside the expectation at will. Moreover, the reward at time t is a deterministic function of the state r_t and action a_t, $r_t = R(s_t, a_t)$. So, it is sufficient to replace the action-value $Q_\pi(s, a)$ by its definition, apply the *law of iterated expectation*, then the EGLP corollary to end the proof:

$$\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [R(\tau)] &= \sum_{t=0}^{\infty} \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\gamma^t \mathop{\mathbb{E}}_{\tau_{t:} \sim \pi_\theta} [R_t(\tau) | s_t, a_t] \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \\
&= \sum_{t=0}^{\infty} \mathop{\mathbb{E}}_{\tau_{:t} \sim \pi_\theta} \left[\mathop{\mathbb{E}}_{\tau_{t:} \sim \pi_\theta} \left[\mathop{\mathbb{E}}_{\tau_{t:} \sim \pi_\theta} \left[\gamma^t R_t(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t) | s_t, a_t \right] \middle| \tau_{:t} \right] \right] \\
&= \sum_{t=0}^{\infty} \mathop{\mathbb{E}}_{\tau_{:t} \sim \pi_\theta} \left[\mathop{\mathbb{E}}_{\tau_{t:} \sim \pi_\theta} \left[\gamma^t R_t(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t) | \tau_{:t} \right] \right] \\
&= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} R(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]
\end{aligned}$$

■

The effect of following the gradient can be better understood after rewriting equation (E.14) once again. The probability of a trajectory is given by equation (3.23), so its log-probability is simply $\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta) = \sum_{t=0}^{\infty} \log \pi_\theta(a_t|s_t)$. It yields,

$$\nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)]\,. \tag{E.17}$$

The log is a strictly monotonic function and can be ignored in the interpretation of this equation, but it is worth mentioning that the term $\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)$ is called *score* in statistics. $\mathbb{P}(\tau|\pi_\theta)$ is the *likelihood* of the trajectory τ given the parameters θ. It measures how likely it is to sample the trajectory τ under the policy π_θ. Assuming a batch of trajectories is available, every gradient ascent step would change their respective likelihood on purpose by updating the parameters θ. Whether the likelihood will increase or decrease is proportional to the observed return $R(\tau)$: the higher the return the more likely to sample the corresponding trajectory in the future. The direct consequence will be the shift of the distribution of trajectories and the discounted state distribution. This seems all very intuitive: making what is working happen more often and avoiding what is not.

MC sampling is used to estimate the policy gradient since the actual distribution is unknown. Let $\mathcal{D}$ be a set of N chunks of trajectory τ with length T. Then,

$$\hat{\nabla}_\theta J(\theta) = \frac{1}{N}\sum_{\tau\in\mathcal{D}}\sum_{t=0}^{\infty} \hat{R}_\tau \nabla_\theta \log \pi_\theta(a_t|s_t), \tag{E.18}$$

where $\hat{R}_\tau$ is the total return for the trajectory τ.

A numerical loss must be constructed from this estimator to interoperate with automatic differentiation frameworks. In theory, only the policy should be differentiated. This is not an issue when the loss is computed using the actual total return. Indeed, the transition steps (s_t, a_t, r_t) are black-box outputs of the environment for which it is impossible to track back their relationship with the policy anyway. However, the total return is commonly replaced with the estimated advantage function, which involves a function approximation of the state-value function. If so, the state-value function and the policy are updated alternatively at every training iteration. While optimizing the policy, the state-value function must be considered independent of the training parameters during backpropagation. This includes any parameter in common with the policy itself. This point is important because the policy and the value networks may share the same feature encoding, as mentioned in section 3.2.2.

The log-likelihood $\log \pi_\theta$ must be known analytically. Let us consider only the most common case where the policy distribution is multivariate diagonal Gaussian with parametric mean and constant standard deviation. Then the log-likelihood is,

$$\log \pi_\theta(a|s) = -\sum_{i=1}^{k}\left(\frac{1}{2}\frac{(\mu_\theta(s_t)_i - a_i)^2}{\sigma_i^2} + \log\sigma_i + \log 2\pi\right), \tag{E.19}$$

where k is the dimension of the action space $\mathcal{A}$.

Reparametrization Estimate

Let us suppose that the action-value function is perfectly known analytically and its gradient can be evaluated at will. This additional information can be leveraged to estimate the policy gradient without relying on the log-likelihood of the policy. It is a direct application of the reparametrization trick (cf. section 3.2.2).

Theorem 11. *Let $\mathcal{M} = \{\mathcal{S}, \mathcal{O}, \mathcal{A}, P, R, O, \rho_0, \gamma\}$ be an infinite horizon stationary MDP with discounted reward. The gradient of the action-value function $Q_\pi(s,a)$ in expectation over the unnormalized discounted state distribution ρ_π induced by policy π_θ with parameters θ is given by*

$$\nabla_\theta \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ a\sim\pi}} [Q_\pi(s,a)] = \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ z\sim\xi}} [\nabla_\theta f_\theta(s,z) \nabla_a Q_\pi(s,a) \mid a = f_\theta(s,z)] \,. \tag{E.20}$$

Proof. This proof follows the same mathematical reasoning as for the theorem 10, so intermediary computations are omitted when possible to avoid redundancies. Let us recall first the expression of the gradient:

$$\nabla_\theta V_\pi(s) = \nabla_\theta \mathop{\mathbb{E}}_{a\sim\pi_\theta} [Q_\pi(s,a)] = \mathop{\mathbb{E}}_{z\sim\xi} [\nabla_\theta Q_\pi(s, f_\theta(s,z))]$$

Taking the derivative of the inner term directly is tricky since both the action and the action-value function depend on the parameters, but it is easy once expended:

$$\begin{aligned}
&\nabla_\theta Q_\pi(s, f_\theta(s,z)) \\
&\quad = \nabla_\theta \mathop{\mathbb{E}}_{s'\sim P} \big[R(s, f_\theta(s,z), s') + \gamma V_\pi(s')\big] \\
&\quad = \int_{s'\in\mathcal{S}} \Big\{ (R(s, f_\theta(s,z), s') + \gamma V_\pi(s')) \nabla_\theta P(s'|s, f_\theta(s,z)) \\
&\qquad\quad + \nabla_\theta (R(s, f_\theta(s,z), s') + \gamma V_\pi(s')) P(s'|s, f_\theta(s,z)) \Big\} \, \mathrm{d}s' \\
&\quad = \int_{s'\in\mathcal{S}} \Big\{ (R(s, f_\theta(s,z), s') + \gamma V_\pi(s')) \nabla_\theta f_\theta(s,z) \nabla_a P(s'|s,a) \\
&\qquad + (\nabla_\theta f_\theta(s,z) \nabla_a R(s,a,s') + \gamma \nabla_\theta V_\pi(s')) P(s'|s, f_\theta(s,z)) \Big\} \, \mathrm{d}s' \\
&\quad = \int_{s'\in\mathcal{S}} \nabla_\theta f_\theta(s,z) \Big\{ (R(s, f_\theta(s,z), s') + \gamma V_\pi(s')) \nabla_a P(s'|s,a) \\
&\qquad + \nabla_a R(s,a,s') P(s'|s, f_\theta(s,z)) \Big\} \, \mathrm{d}s' + \gamma \int_{s'\in\mathcal{S}} \nabla_\theta V_\pi(s') P(s'|s, f_\theta(s,z)) \, \mathrm{d}s' \\
&\quad = \nabla_\theta f_\theta(s,z) \int_{s'\in\mathcal{S}} \nabla_a \Big\{ (R(s,a,s') + \gamma V_\pi(s')) P(s'|s,a) \Big\} \, \mathrm{d}s' + \gamma \mathop{\mathbb{E}}_{s'\sim P} \big[\nabla_\theta V_\pi(s')\big] \\
&\quad = \nabla_\theta f_\theta(s,z) \nabla_a Q_\pi(s,a)|_{f_\theta(s,z)} + \gamma \mathop{\mathbb{E}}_{s'\sim P} \big[\nabla_\theta V_\pi(s')\big]
\end{aligned}$$

The gradient of the value function is obtained by injecting and unrolling this formula:

$$\nabla_\theta V_\pi(s) = \mathop{\mathbb{E}}_{z\sim\xi} [\nabla_\theta f_\theta(s,z) \nabla_a Q_\pi(s,a) | a = f_\theta(s,z)] + \gamma \mathop{\mathbb{E}}_{\substack{s'\sim P \\ z\sim\xi}} \big[\nabla_\theta V_\pi(s')\big]$$

$$= \underbrace{\mathop{\mathbb{E}}_{z\sim\xi}[\nabla_\theta f_\theta(s,z)\nabla_a Q_\pi(s,a)|a=f_\theta(s,z)]}_{\phi_\pi(s)} + \gamma \int_{s'\in\mathcal{S}} \rho_\pi^{(1)}(s\to s')\nabla_\theta V_\pi(s')\,\mathrm{d}s'$$

$$\cdots = \int_{s'\in\mathcal{S}} \phi_\pi(s')\eta_\pi(s'|s)\,\mathrm{d}s'$$

It just remains to take the expectation over the initial state distribution ρ_0:

$$\begin{aligned}\nabla_\theta J(\pi) &= \mathop{\mathbb{E}}_{s\sim\rho_0}[\nabla_\theta V_\pi(s)] \\ &= \mathop{\mathbb{E}}_{s\sim\rho_\pi}[\phi_\pi(s)] \\ &= \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ z\sim\xi}}[\nabla_\theta f_\theta(s,z)\nabla_a Q_\pi(s,a) \mid a=f_\theta(s,z)]\end{aligned}$$

■

E.5.2 REINFORCE

Introduced by Williams (1992), REINFORCE ("REward Increment × Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility") is the most basic policy gradient algorithm (see algorithm 7). It relies on the empirical future return for complete episodes just like the MC methods, i.e. $G_t = \sum_{k=0}^{T-t}\gamma^k r_{t+k}$. It has the benefit to be simple and does not require estimating a value function. However, it suffers from the same high variance issues as MC methods. The root cause is that all information is lost between iterations, and hence any estimate is going to be extremely sensitive to the training batch at every iteration. REINFORCE algorithm is generally not able to find reasonably good solutions for real-world applications.

Algorithm 7: REINFORCE (Williams, 1992)

Input: Step size: α, Initial parameters of the policy: θ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect a set $\mathcal{D}_k$ of N trajectories τ of length T on policy $\pi_k = \pi(\theta_k)$;
  Compute future return $\hat{G}_t = \sum_{k=0}^{T-t}\gamma^k r_{t+k}$;
  Estimate policy gradient $\hat{g}_k$ as

$$\hat{g}_k = \frac{1}{N}\sum_{\tau\in\mathcal{D}_k}\sum_{t=0}^{T}\hat{G}_t\nabla_\theta \log\pi_\theta(a_t|s_t)|_{\theta_k};$$

  Update the policy by gradient ascent $\theta_{k+1} = \theta_k + \alpha\hat{g}_k$;
end

One way to mitigate this issue is defining a *baseline*. Mathematically, it is a quantity subtracted from the action-value in the gradient estimate to reduce its variance without introducing bias. There are infinitely many possibilities. Williams suggest

using a moving average of the empirical future return written as a recursive filter,

$$\bar{G}_{k+1} = \alpha G_t + (\alpha - 1)\bar{G}_k. \tag{E.21}$$

They also mentioned training a function approximation of the state-value function, which is still today the most common baseline in state-of-the-art on-policy learning algorithms. Indeed, one can show that it is the optimal state-dependent baseline. Nonetheless, there may exist a better baseline without such a restriction, and finding the most effective one is still an active research topic (cf. appendix E.7.1).

Sutton et al. (1999) avoid defining a baseline entirely by replacing the empirical future return in the gradient with a function approximation of the action value under the current policy. At every training iteration, the Q-Network is first updated as many times as necessary to converge. Then, the parameters of the policy are updated once via the gradient descent following REINFORCE's update rule. This allows for aggregating new information and benefits from the generalization ability of the network, which dramatically reduces the variance of the gradient estimate. Any of the methods previously introduced can be used to train the Q-Network. To deviate the least from REINFORCE, Sutton et al. proposed originally to minimize the mean squared error between the current prediction and the empirical future return. Although intuitive, this approach is flawed. First, the training procedure converges to a local minimum. As such, the function approximation will be more prone to get stuck in a local minimum if the step size is large, but it will forget already cumulated knowledge at a slower rate. Formally, a small step size increases the bias but decreases the variance and vice versa. Secondly, the function approximation has limited expressiveness, which also introduces bias unless some impracticable condition is met. Any function approximation satisfying these conditions is said to be *compatible*, and the corresponding update rule is a *true gradient*.

It is possible to reduce the variance without adding much bias by rather minimizing the residual TD error and performing only one update step per training sample. This actor-critic algorithm has the same benefits as TD learning, including learning from incomplete episodes. It can either be on-policy like SARSA or off-policy like Q-learning. Yet, the on-policy variant was never really used because its drawbacks outweigh its advantages. Notably, a gradient estimate based on a function approximation of the action-value would most certainly be biased. On the contrary, gradient estimates involving the advantage function based on a function approximation of the state-value function are guaranteed to be unbiased while having even lower variance in practice. The lone consequence of a poor approximation of the state-value function is a larger variance than could have been optimally achieved. This on-policy algorithm is simply called Stochastic Actor-Critic (SAC) or *Advantage Actor-Critic* (A2C) (Degris et al., 2012a). *Asynchronous A2C* (A3C) (Mnih et al., 2016) extend it to multiple agents learning in parallel but sharing the same parameters to get the most out of the available CPU resources.

E.5.3 Vanilla Off-Policy Actor-Critic Learning

Degris et al. (2012b) present the vanilla *Off-Policy Actor-Critic* (OffPAC) algorithm for stochastic policies with normal distribution, which is a very important milestone. As with any off-policy methods, the distribution of training samples is not consistent with the target policy being trained. This is not an issue in DQN because it learns the optimal action-value function instead of the current one, but it presents its own limitations. First, it does not scale well to continuous action spaces: finding the best action for every single visited state is a non-linear non-convex optimization problem on its own. Secondly, it suffers from an overestimation bias without advanced techniques to prevent it. In theory, gradient ascent could be used to find it since action-value is a function approximation and thus analytically differentiable. Yet, this approach is impracticable because it results in a non-linear non-convex optimization problem that would be very costly to solve. Another option is to compute an approximation solution using *Cross-Entropy Method* (CEM). It is a simple derivative-free optimization algorithm that samples a batch of N values at each iteration, fits a Gaussian distribution to the best $M < N$ of these samples, and then samples the next batch from that Gaussian. This method is relatively cheap and scalable but inaccurate. The resulting algorithm from combining DQN and CEM is called QT-Opt (Kalashnikov et al., 2018). It has been successfully applied for vision-based manipulation using an online RL framework with many robots in parallel.

Degris et al. propose to completely avoid computing the best action by learning the current action-value instead of the optimal one. None of the mechanisms of DQN are used, so there is no replay buffer nor delayed update. A dedicated behavior policy is used to collect samples and is theoretically unrelated to the target policy and action-value that are learned. Let $\beta(a|s)$ denote a stochastic *behavior policy*. The objective function is defined as the expectation of the return over the discounted state distribution ρ_β induced by the behavior policy,

$$J(\theta) = \mathop{\mathbb{E}}_{s\sim\rho_\beta}\left[V_\pi(s)\right] = \mathop{\mathbb{E}}_{\substack{s\sim\rho_\beta \\ a\sim\pi_\theta}}\left[Q_\pi(s,a)\right]. \tag{E.22}$$

Any state distribution can be used without adding much bias, even though in theory it should be the initial state distribution ρ_0 as mentioned before. The analytical gradient of the objective function is easy to compute since the state distribution is independent of the target policy. It gives,

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{s\sim\rho_\beta}\left[\int_{a\in\mathcal{A}} \left\{Q_\pi(s,a)\nabla_\theta\pi_\theta(a|s) + \pi_\theta(a|s)\nabla_\theta Q_\pi(s,a)\right\}\right]. \tag{E.23}$$

The inner term in the integrand $\pi_\theta(a|s)\nabla_\theta Q_\pi(s,a)$ is neglected because it is too difficult to estimate in an incremental off-policy setting. In spite of this, Degris et al. (2012b) have proven that monotonic improvement of the policy is guaranteed, and it converges to a true local minimum of the objective $J(\theta)$. At this point, the actions must still be taken according to the target policy. This is an issue because we would like to be able to collect samples without relying on the target policy at all. This

is where *Importance Sampling* (IS) comes into play. It is a technique for estimating quantities associated with a particular distribution, while only samples drawn from a different distribution are available. The gradient estimate is reformulated as follows,

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \mathop{\mathbb{E}}_{s\sim\rho_\beta}\left[\int_{a\in\mathcal{A}} Q_\pi(s,a)\beta(a|s)\frac{\pi_\theta(a|s)}{\beta(a|s)}\frac{\nabla_\theta\pi_\theta(a|s)}{\pi_\theta(a|s)}\right] \\ &= \mathop{\mathbb{E}}_{\substack{s\sim\rho_\beta \\ a\sim\beta}}\left[Q_\pi(s,a)\frac{\pi_\theta(a|s)}{\beta(a|s)}\nabla_\theta\log\pi_\theta(a|s)\right], \end{aligned} \tag{E.24}$$

In practice, the behavior policy is a uniform distribution over all possible actions, and the parameters θ are updated using a single gradient ascent step as always.

Degris et al. present an action-value function estimate that combines a function approximation of the state-value function with eligibility trace. The eligibility trace assigns a lower weight to future transition steps as time progresses to rely instead on the predicted state-value. This significantly reduces the variance but introduces bias because the function approximation is always off to some extent. The decay rate λ is a hyperparameter that must be fine-tuned manually for each problem to adjust this trade-off optimally. The resulting action-value estimate is based on the discounted residual TD errors. It is very similar to the *Generalized Advantage Estimator* (GAE) presented in appendix E.7.1 but in the off-policy setting. OffPAC performs much better than the off-policy Q-learning methods that define the target policy as the best action according to a function approximation of the optimal action-value. It is true at least for algorithms using greedy or softmax policies for exploration. However, the vanilla IS estimator does not scale well with the dimensionality of the action space, and the weights rapidly degenerate (cf. appendix E.7.1). It dramatically increases the variance of the gradient estimate and makes learning unstable, so it is never used and deterministic methods are preferred (cf. appendix E.6).

E.6 Deterministic Policy Gradient

Until now, the policy was considered stochastic, but this is not a prerequisite. Mathematically, a deterministic policy $\mu_\theta(s)$ is equivalent to a stochastic one with binary indicator distribution $\pi(a|s) = \mathbb{1}_{\{\mu_\theta(s)\}}(a)$, so that expectation over actions is a single value. This distribution is not differentiable, which implies that the policy gradient estimate based on the likelihood ratio is not applicable. Nevertheless, a gradient estimate based on the reparametrization trick can be derived,

$$\nabla_\theta J(\pi) = \mathop{\mathbb{E}}_{s\sim\rho_\pi}\left[\nabla_\theta\mu_\theta(s)\nabla_a Q_\pi(s,a) \mid a=\mu_\theta(s)\right]. \tag{E.25}$$

The resulting family of policy learning algorithm is called *Deterministic Policy Gradient* (DPG) by Silver et al. (2014). Although on-policy learning with a deterministic policy is theoretically possible, it is not viable. The main issue is that the policy cannot promote exploration. This would lead to sub-optimal solutions unless there

is sufficient noise in the environment itself to ensure adequate exploration, which is highly likely in practice. Thus, its primary purpose is didactic.

It seems more reasonable to train a deterministic target policy while still relying on a stochastic behavior policy for the sake of exploration. The stochastic off-policy learning algorithm OffPAC served as the foundation for the deterministic off-policy methods. The minimal modification consists in replacing the stochastic target policy with a deterministic one. Let β denote the behavior policy. The objective in equation (E.22) can be simplified,

$$J(\theta) = \mathop{\mathbb{E}}_{s\sim\rho_\beta}\left[Q_\pi(s, \mu_\theta(s))\right]. \tag{E.26}$$

The behavior policy is supposed to be independent of the target policy μ_θ that is being trained. This avoids any coupling between the sampling distribution and the one induced by the target policy. The gradient estimate is obtained by direct application of the chain rule:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathop{\mathbb{E}}_{s\sim\rho_\beta}\left[\nabla_\theta Q_\pi(s, \mu_\theta(s))\right] \\ &= \mathop{\mathbb{E}}_{s\sim\rho_\beta}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q_\pi(s, a) + \mu_\theta(s) \nabla_\theta Q_\pi(s, a) \mid a = \mu_\theta(s)\right]\end{aligned}$$

Unlike OffPAC, this gradient estimate does not involve IS despite the mismatch between the target and behavior policy. This is a major advantage since the IS is known to dramatically increase the variance, making learning unstable. Following the same rationale as Degris et al. (2012b), the second inner term $\mu_\theta(s)\nabla_\theta Q_\pi(s, a)$ is dropped. It can be viewed as a greedy local improvement of the target policy with the sole objective of taking the best action according to the current estimate of the action-value function in all visited states, while completely ignoring the impact this will have on the closed-loop dynamics, much like *Behavior Cloning* (BC).

The critic is a linear function approximation that estimates the action-value from features $\phi(s, a) = \nabla_\theta \mu_\theta(s)(a - \mu_\theta(s))$. The linearity property guarantees some of the compatibility conditions to preserve the true gradient when replacing the actual action-value function with this approximation. Off-policy Q-learning may diverge when using linear function approximation. Similar to the OffPAC algorithm, Silver et al. use gradient temporal-difference learning to update the parameters if the critic. This ensures that the critic converges as long as it is updated at a much faster rate than the actor to decouple them. A linear function approximator is not powerful enough for predicting action-values globally since the action-value diverges for large actions. Silver et al. mitigate this issue by injecting a linear function approximation of the state-value function as a baseline, i.e. $Q_w(s, a) = w^T\phi(s, a) + v^T\phi(s)$ where the features $\phi(s)$ are generated by tile-coding the state-space. A natural interpretation is that $V_v(s) = v^T\phi(s)$ estimates the value of state s, while the first term estimates the local advantage $A_w(s, a) = w^T\phi(s, a)$ of taking action a over action $\mu_\theta(s)$ in state s.

At every iteration, the critic first estimate the current action-value function, and then the actor is updated in the direction prescribed by the critic until convergence to a consensus. This training procedure resembles *Alternating Direction Method of*

Multipliers (ADMM) where two mutually dependent are solved separately until convergence. Therefore, it is reasonable to expect this algorithm converges to the globally optimal policy, but it cannot be guaranteed for non-linear function approximations.

In practice, the behavior policy is simply to target policy with a normalized noise with fixed variance added to it. DPG has been compared to OffPAC for the same behavior policy. DPG was performing slightly better and was more stable.

E.7 Deep Deterministic Policy Gradient

DQN is still competitive nowadays for discrete action space problems. It has the advantage to be sample efficient thanks to the replay buffer, but it does not generalize well to continuous action spaces. DPG does not have such a limitation, but learning is unstable because of the coupling between the update of the action-value and the policy. It is natural to combine both approaches to circumvent those limitations. The resulting off-policy actor-critic algorithm is called *Deep DPG* (DDPG) (Lillicrap et al., 2016). As for updating the parameters of the Q-network, a single gradient ascent step is done at every training iteration, using the very same batch of transition steps. Let $\mathcal{B}$ be a set of N transition steps $\{e_i\}_{i=0}^{N}$. It follows,

$$\theta \leftarrow \theta + \alpha \frac{1}{N} \sum_{e_t \in \mathcal{B}} \nabla_\theta \mu_\theta(s) \nabla_a Q_\pi(s,a)|_{a=\mu_\theta(s)}. \tag{E.27}$$

The policy π_θ is updated toward the best action based on an approximation of the action-value Q_ϕ that is itself changing to better estimate the optimal one Q^*. This coupling is stable because it is not reciprocal. Nevertheless, a target policy $\hat{pi}$ must be introduced in the computation of TD targets to avoid learning instabilities. Following the delayed update mechanism, it is a frozen copy of the policy π_θ that is updated periodically, jointly with the target Q-network. The original loss function of DQN is modified as follows,

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_{e_t \in \mathcal{B}} \Big(r_t + \gamma \hat{Q}(s_t, \hat{\pi}(s_t)) - Q_\phi(s_t, a_t) \Big)^2. \tag{E.28}$$

ϵ-greedy is no longer used to sample the actual action performed by the agent. Instead, a small gaussian noise is systematically added to every action taken by the action. The standard deviation is a hyperparameter. It is usually scheduled to be large at first and monotonically reduce as training progresses. This helps to fine-tune the behavior of the agent in complex problems while escaping bad local minima.

E.7.1 Variance Reduction

Monte-Carlo Sampling from Incomplete Episodes

The gradient estimate in equation (E.18) is unbias but has several flaws. First, it requires sampling from complete episodes. Secondly, every trajectory is considered as

a whole, ignoring the effect of individual actions. It corresponds to the MC method presented in appendix E.2, which is known to be unreliable due to very large variance.

The formulation in Equation (E.15) enables sampling from incomplete episodes to compute the policy gradient, contrary to equation (E.14). Let us consider a batch $\mathcal{B}$ of N transition steps $e_t = \{t, s_t, a_t, r_t, s_{t+1}\}$. According to equation (E.16), the gradient of the objective function $J(\theta)$ can be estimated using MC sampling as follows,

$$\hat{\nabla}_\theta J(\theta) = \frac{1}{N} \sum_{e_t \in \mathcal{B}} \gamma^t Q_\pi(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{E.29}$$

This gradient estimator is theoretically unbiased. This issue has already been discussed in appendix E.4. However, the action-value $Q_\pi(s_t, a_t)$ itself is unknown and must be estimated beforehand. While this could be done in a non-parametric fashion based on the collected data alone, the resulting variance would be prohibitive. Instead, it is a function approximation jointly trained with the policy. This function approximation is never going to match the true action-value function in practice, which introduces bias in the gradient estimator. Off-line methods are facing the very same issue (cf. appendix E.4).

Thomas (2014) pointed out that most if not all state-of-the-art policy gradient methods are dropping the term γ^t in equation (E.29). This modification is essential for these algorithms to have any practical value as it dramatically improves their sample efficiency. Otherwise, the term γ^t would decay to zero rapidly, causing the true gradient to ignore data collected after a short burn-in period. The modified formula goes against the policy gradient theorem except in the averaging scenario ($\gamma = 1$). In all other cases, it is not even the gradient of any function because its mixed partials are not symmetric, which is impossible according to *Clairaut's theorem*. Interestingly, the discount factor is ignored in the state distribution but not in the action-value function. It means that these biased algorithms are actually optimizing something closer to the expectation of the average reward, with the discount serving to reduce variance, much like TD(λ) (Thomas, 2014).

Recently, Nota and Thomas (2020) have derived a very generic identity that highlights the mistake in the existing formulation. Let us define $\tilde{\rho}_\pi = \sum_{k=0}^{\infty} \omega_k \mathbb{P}(s_t = s)$, where the sequence $\{w_k\}_{k=0}^{\infty}$ are free parameters. Then,

$$\mathop{\mathbb{E}}_{s \in \tilde{\rho}_\pi} [\nabla_\theta V_\pi(s)] = \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} Q(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{k=0}^{t} \omega_k \gamma^{t-k} \right] \tag{E.30}$$

The true gradient is obtained for $w_0 = 1, w_i = 0 \ \forall i \geq 1$. Likewise, the gradient estimate after dropping the term γ^t corresponds to $w_0 = 1, w_i = 1 - \gamma \ \forall i \geq 1$. As expected, both matches in the averaging scenario but that is all. The factor γ^t is ignored most of the time, which is a mistake introducing bias to an otherwise unbiased estimator. More precisely, it means that it does not optimize the return with respect to the initial state distribution ρ_0 but some weighted stationary distribution.

This discrepancy is not a big deal in most cases, but it may lead to complete failure in finding an even reasonable policy in pathological scenarios (Nota & Thomas,

2020). Nevertheless, it is generally profitable for continuing tasks. In this context, setting a time limit to the episodes would guarantee sufficient sample diversity as long as the start time has no particular meaning. The initial state should be indistinguishable from any other to prevent side effects. This implies that the initial state distribution should match the stationary one under the optimal policy. The latter is unknown in advance, so the former is going to be a cheap approximation based on prior assumptions. Thus, the state distribution involved in the modified gradient estimate is actually closer to the stationary one than the initial one in practice.

Generalized Advantage Estimator

It is impossible to compute exactly the policy gradient using MC sampling if the state space is continuous because it requires collecting an infinite amount of data. Instead, it is approximated from a batch of transition steps at every training iteration. Replacing the true expectation with the empirical mean results in different estimators, all having their own bias and variance. Limiting the bias is important to converge to the optimal policy while reducing the variance improves sample efficiency. How to obtain an unbiased estimator with the lowest possible variance is an open question.

The most straightforward solution to reduce the variance is to replace the return $R(\tau)$ in equation (E.14) by the future rewards following action a_t, i.e. $\gamma^t R_t(\tau)$. Intuitively, past events do not depend on the current state and action, so it is possible to get rid of them in the expectation. They have zero mean but non-zero variance, so they would just increase the variance of the estimate of the policy gradient without any advantage. Beyond this, the control variates method is the most widely used variance reduction technique in policy gradient. Suppose the expectation $\mu = \mathbb{E}_{\tau\sim\mathcal{T}}[g(s,a)]$ must be estimated via MC sampling of trajectories $\tau = \{(s_t, a_t)\}_{t=0}^{T}$ drawn from some distribution $\mathcal{T}$. A *control variate* is an arbitrary function $f(s,a)$ with known expectation. It is assumed without lose of generality that $\mathbb{E}_{\tau\sim\mathcal{T}}[f(s,a)] = 0$. With f, an alternative unbiased estimator of μ can be defined,

$$\hat{\mu} = \hat{\mathop{\mathbb{E}}_{\tau\in\mathcal{D}}}[g(s,a) - f(s,a)], \tag{E.31}$$

where $\hat{\mathbb{E}}$ denotes the empirical mean, and $\mathcal{D}$ is a finite set of trajectories. The variance of this estimator is proportional to $\mathrm{var}(g - f)$ instead of $\mathrm{var}(g)$ for the vanilla MC estimator. The variance can be significantly reduced by taking f to be similar to g, thus resulting in a more reliable estimator.

When applied to policy gradient, the control variate is referred to as the *baseline* and denoted ψ. Historically, Weaver and Tao (2001) suggested learning the optimal constant baseline. From equation (E.17), it yields,

$$\psi = \frac{\mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)^2]}{\mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)^2]}. \tag{E.32}$$

This approach has been largely disregarded as the marginal reduction of variance it brings is not worth the additional complexity. From the EGLP corollary, any function

ψ that depends on the current state s_t would be valid baseline, i.e.

$$\mathop{\mathbb{E}}_{a\sim\pi}[\psi(s)\nabla_a \log \pi(a|s)] = 0. \tag{E.33}$$

The optimal state-dependent baseline can be learned by minimizing the variance of the gradient estimator explicitly. However, it involves training yet another function approximation since the state-value function is needed to a compute the action-value function from incomplete episodes regardless. This can be avoided by using the state-value function itself as a baseline. It is a reasonably good approximation of optimal. One can demonstrate that the variance of the estimator is approximately proportional to $\mathbb{E}_{\tau\sim\pi}[(\gamma^t R_t(\tau) - b(s_t))^2]$, and hence minimizing it is a Least Squares problem. It is well known that the optimal solution is given by $\mathbb{E}_{\tau\sim\pi}[\gamma^t R_t(\tau)|s_t]$, i.e. the discounted value function $\gamma^t V_\pi(s_t)$. It follows that the action-value $Q_\pi(s_t, a_t)$ can be replaced by the advantage function $A_\pi(s_t, a_t)$ in the gradient estimate.

It turns out that the discounted residual TD error $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is an unbias estimator of the advantage for the true value function.

$$\mathop{\mathbb{E}}_{s_{t+1}\sim\rho_\pi}[\delta_t|s_t] = \mathop{\mathbb{E}}_{s_{t+1}\sim\rho_\pi}[r_t + \gamma V(s_{t+1})|s_t] - V(s_t) = Q(s_{t+1}) - V(s_t) = A(s_t)$$

The GAE generalizes this formulation (Schulman et al., 2016). This allows for reducing the variance even further, at the cost of adding more bias. The trade-off between both is determined by the additional parameter λ. It is very similar to the return estimator in TD(λ) algorithm. First, an n-step advantage function estimator $\hat{A}_t^{(n)}$ is defined as the difference between the n-step return $\hat{G}_t^{(n)}$ in equation (E.8) and a baseline $V_\pi(s_t)$. The contribution is to reformulate the advantage as a telescoping sum of residual TD errors,

$$\hat{A}_t^{(n)} = \hat{G}_t^{(n)} - V_\pi(s_t) = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V_\pi(s_{t+n}) - V_\pi(s_t) = \sum_{i=0}^{n-1} \gamma^i \delta_{t+i}. \tag{E.34}$$

Then, following the same principle that TD(λ), all possible n-step estimators are summed up with exponential weight decay factor λ to get a better estimate, namely

$$\hat{A}_t^{\text{GAE}} = (1-\lambda)\sum_{k=1}^{\infty} \lambda^{k-1} \hat{A}_t^{(k)} = \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k}. \tag{E.35}$$

This advantage estimator is state-of-the-art. In practice, it is used systematically.

Stein Control Variate

More recently, Liu et al. (2018) have proposed a broader class of functionals based on Stein-Hudson identity. Let $p(x)$ be a smooth density supported on $\mathcal{X} \subseteq \mathbb{R}^d$, and $\psi(x) = [\psi_1(x), \ldots, \psi_d(x)]^T$ a smooth vector function. Stein's identity states that for sufficiently regular $\psi(x)$,

$$\mathop{\mathbb{E}}_{x\sim p}[\mathcal{A}_p\psi(x)] = 0, \quad \mathcal{A}_p\psi(x) = \nabla_x \log p(x)\, \psi(x)^T + \nabla_x \psi(x). \tag{E.36}$$

where $\mathcal{A}_p$ is called the Stein operator. This identity can be easily checked using integration by parts, assuming mild zero boundary conditions on $\psi(x)$, either $p(x)\psi(x) = 0, \forall x \in \partial\mathcal{X}$ if $\mathcal{X}$ is compact, or $\lim_{\|x\|\to\infty} p(x)\psi(x) = 0$ when $\mathcal{X} = \mathbb{R}^d$. The following theorem enables applying this identity as a control variate. It is not straightforward because the derivatives with respect to a and θ must be converted.

Theorem 12 (Stein Control Variate)**.** *Let π_θ be a stochastic policy that is reparameterizable, i.e. $a = f_\theta(s, z)$ where $z \sim \xi$ and ξ is independent of θ. Then, for all real-valued function $\psi(s, a)$ s.t. $\pi(a|s)\psi(s, a)$ decays exponentially fast to zero,*

$$\mathop{\mathbb{E}}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\, \psi(s,a)^T \mid s\right] = \mathop{\mathbb{E}}_{z\sim\xi}\left[\nabla_\theta f_\theta(s,z) \nabla_a \psi(s,a) \mid s, a = f_\theta(s,z)\right]. \tag{E.37}$$

Proof. First, the Stein's identity is applied on $\mathbb{P}(a|s, z)$ s.t. $x = (a, s, z)$,

$$\mathop{\mathbb{E}}_{\mathbb{P}(a|s,z)}\left[\nabla_{(a,s,z)} \log \mathbb{P}(a|s,z)\, \psi(a,s,z)^T + \nabla_{(a,s,z)}\psi(a,s,z)\right] = 0$$

The functional ψ is independent of z, and only the derivative with respect to a is of interest. Besides, $\nabla_a \log \mathbb{P}(a|s, z) = \nabla_a(\log \mathbb{P}(a, z|s) - \log \mathbb{P}(z|s)) = \nabla_a \log \mathbb{P}(a, z|s)$.

$$\mathop{\mathbb{E}}_{\mathbb{P}(a|s,z)}\left[\nabla_a \log \mathbb{P}(a,z|s)\, \psi(a,s)^T + \nabla_a \psi(a,s)\right] = 0$$

The policy is given by $\pi(a|s) = \mathbb{E}_{z\sim\xi}[\mathbb{1}_{\{a\}}(f_\theta(s, z))]$. The weak derivative of the integrand is the same as $\lim_{h\to 0} \exp(-\|a - f_\theta(s, z)\|_2^2/h^2)/h^2$. It follows from the Lebesgue-dominated convergence and monotone convergence theorems,

$$\pi_\theta(a|s) = \lim_{h\to 0} \mathop{\mathbb{E}}_{z\sim\xi}\left[\exp(-\|a - f_\theta(s,z)\|_2^2/h^2)/h^2 \mid s\right].$$

Therefore, $\mathbb{P}(a, z|s)$ can be replaced by the Gaussian $\exp(-\|a - f_\theta(s, z)\|_2^2/h^2)/h^2$ in computations, taking the limit afterward. It is easy to show that,

$$\nabla_\theta \log \mathbb{P}(a,z|s) = -\nabla_\theta\, f_\theta(s,z) \nabla_a \log \mathbb{P}(a,z|s).$$

Multiplying both sides with $\psi(s, a)$ and taking the conditional expectation:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathbb{P}(a,z|s)}\left[\nabla_\theta \log \mathbb{P}(a,z|s)\, \psi(s,a)^T\right] &= \mathop{\mathbb{E}}_{\mathbb{P}(a,z|s)}\left[-\nabla_\theta f_\theta(s,z) \nabla_a \log \mathbb{P}(a,z|s)\, \psi(s,a)^T\right] \\
&= \mathop{\mathbb{E}}_{z\sim\xi}[\nabla_\theta f_\theta(s,z) \mathop{\mathbb{E}}_{\mathbb{P}(a|s,z)}\left[-\nabla_a \log \mathbb{P}(a,z|s)\, \psi(s,a)^T\right]|s] \\
&= \mathop{\mathbb{E}}_{z\sim\xi}[\nabla_\theta f_\theta(s,z) \mathop{\mathbb{E}}_{\mathbb{P}(a|s,z)}\left[\nabla_a \psi(s,a)\right] \mid s] \\
&= \mathop{\mathbb{E}}_{z\sim\xi}\left[\nabla_\theta f_\theta(s,z) \nabla_a \psi(s,a) \mid s, a = f_\theta(s,z)\right]
\end{aligned}$$

where the third equality comes from Stein's identity. On the other hand:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\mathbb{P}(a,z|s)}\left[\nabla_\theta \log \mathbb{P}(a,z|s)\,\psi(s,a)^T\right] \\
&\quad= \mathop{\mathbb{E}}_{\mathbb{P}(a,z|s)}\left[\nabla_\theta \log(\pi_\theta(a|s)\mathbb{P}(z|s,a))\,\psi(s,a)^T\right] \\
&\quad= \mathop{\mathbb{E}}_{\mathbb{P}(a,z|s)}\left[\nabla_\theta \log \pi_\theta(a|s)\,\psi(s,a)^T\right] + \mathop{\mathbb{E}}_{a\sim\pi_\theta}\Big[\underbrace{\mathop{\mathbb{E}}_{\mathbb{P}(z|s,a)}\left[\nabla_\theta \log \mathbb{P}(z|s,a)\right]}_{=0 \text{ (EGLP)}}\,\psi(s,a)^T \mid s\Big] \\
&\quad= \mathop{\mathbb{E}}_{a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\,\psi(s,a)^T \mid s\right]
\end{aligned}$$

■

The most generic formulation of policy gradient $\nabla_\theta J(\theta)$ with control variate $\psi(s,a)$ is derived from theorems 10 and 12,

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ z\sim\xi}}[\nabla_\theta \log \pi_\theta(a|s)\,(Q_\pi(s,a) - \psi(s,a))^T + \nabla_\theta f_\theta(s,z)\nabla_a \psi(s,a) \mid a = f_\theta(s,z)], \tag{E.38}$$

where any fixed choice of $\psi(s,a)$ does not introduce bias. Let us consider a batch $\mathcal{B}$ of N transition steps $e_t = \{t, s_t, z_t, a_t, r_t, s_{t+1}\}$ s.t. $a_t = f_\theta(s_t, z_t)$, an estimator of the gradient is

$$\hat{\nabla}_\theta J(\theta) = \frac{1}{N}\sum_{e_t\in\mathcal{B}}\Big\{\nabla_\theta \log \pi_\theta(a_t|s_t)\,(Q_\pi(s_t,a_t) - \psi(s_t,a_t))^T + \nabla_\theta f_\theta(s_t,z_t)\nabla_a \psi(s_t,a_t)\Big\}. \tag{E.39}$$

Equation (E.38) bridges the gap between on- and off-policy methods. If ψ is action-independent, then the last term in the integrand equals zero and the estimator corresponds to the formulation in equation (E.17). It is known as *likelihood-ratio gradient* (LR) and is used in all on-policy algorithms based on REINFORCE or A2C. Conversely, if ψ is the action-value function, then the first term vanishes. The resulting estimator is referred to as *ReParametrization gradient* (RP),

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\substack{s\sim\rho_\pi \\ z\sim\xi}}\left[\nabla_\theta f_\theta(s,z)\nabla_a Q_\pi(s,a) \mid a = f_\theta(s,z)\right].$$

It is well known from the variational inference literature that LR has much higher variance than RP (Liu et al., 2018; Tang & Abbeel, 2010). The former is based on the principle of modifying the likelihood of an action based on its outcome. The capability of doing so is directly related to the amount of information brought by collected samples, namely the entropy of the policy. If the latter is almost deterministic, then the variance of the estimator would be infinite. On the contrary, the latter is rather trying to modify the policy to predict the best action no matter the actual outcome. As such, the variance of this gradient estimator is not affected by the diversity of action. If the policy is deterministic, then it corresponds to DPG algorithm presented in the next section. The two formulations are compared in figure E.3. In any case, stochasticity is still useful to mitigate overfitting of the action-value and smoothing

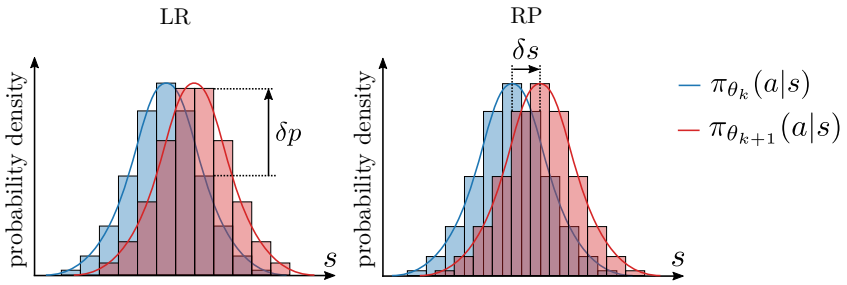


Figure E.3: Comparison between LR and RP gradients. LR keeps the positions of the bars fixed, while RP keeps the probability mass fixed.

out artifacts that would lead to over-estimation otherwise. It is especially critical for off-policy algorithms to rely entirely on the action-value estimate to select the best action. In particular, *Twin-Delayed DDPG* (TD3) addresses this issue in the context of deterministic policy. In practice, the policy is close to being deterministic to keep optimizing locally around the best performing strategy, so RP gradient can be expected to have to lower variance than LR in the context of RL. Lately, Parmas and Sugiyama (2021) have carried out an in-depth analysis of the relationship between both. They provide a physical interpretation based on fluid dynamics. LR and RP are alternative methods of keeping track of the movement of probability mass, and the two are connected via the divergence theorem. It also proves that there is no better MC gradient estimator outside the search space characterized by equation (E.38).

There are two main approaches for constructing a baseline: fit the action-value function or minimize the variance of the gradient estimator explicitly. The action-value function is natural because it is just the generalization of the usual state-value function baseline. It is a good choice as it cancels the LR term in expectation if the approximation is unbiased, which has the largest variance between both terms. However, it is not the best choice because the RP term has also its own variance and bias. The optimal trade-off depends on the stochasticity of the transition steps, which varies from one application to another and is likely to change over training iterations. It seems preferable to find the baseline minimizing the variance at the current point. Note that $\mathrm{Var}(\hat{\nabla}_\theta J(\theta)) = \mathbb{E}[(\hat{\nabla}_\theta J(\theta))^2] - \mathbb{E}[\hat{\nabla}_\theta J(\theta)]^2$. Since $\mathbb{E}[\hat{\nabla}_\theta J(\theta)] = \nabla_\theta J(\theta)$ does not depend on ψ, it is sufficient to minimize the first term, that is

$$\min_\omega \sum_{t=0}^{T} \|(Q_\pi(s_t, a_t) - \psi_\omega(s_t, a_t))\nabla_\theta \log \pi_\theta(a_t|s_t) + \nabla_\theta f_\theta(s_t, z_t)\nabla_a \psi_\omega(s_t, a_t)\|_2^2 , \tag{E.40}$$

where ψ is a function approximation with parameter ω. It involves the gradient of f_θ with respect to the parameters θ, which is costly to evaluate by design for some deep learning frameworks but not all. Liu et al. (2018) suggested to specialize the computations for Gaussian policies and to optimize $\mathrm{var}(\hat{\nabla}_\mu J(\theta)) + \mathrm{var}(\hat{\nabla}_\Sigma J(\theta))$ as a proxy. It gets rid of the derivative with respect to θ, but it lacks mathematical grounding. It works reasonably well but not significantly better than fitting the

action-value instead. In any case, it is more efficient to write the baseline as $\psi(s,a) = V_\pi(s) + \phi(s,a)$ where V_π is a function approximation. It is equivalent to replacing the action-value function by the advantage in equation (E.38). It is profitable because the function to learn would be centered around zero.

It is convenient to use the action-value function as a baseline because it can be estimated off-policy, even if the rest of the algorithm is on-policy. In which case, it consists in minimizing the Bellman error for the best action, just like DQN. It is not true for all baseline $\psi(s,a)$. For this reason, it may be beneficial to learn the action-value instead of a more generic control variate because it may be harder to learn an accurate estimate of it. The resulting formulation of the policy gradient differs from DDPG in that it corrects the bias induced by estimating the action-value with a function approximation. Q-Prop (Gu et al., 2017) is the earliest method that does exactly this. It constructs a state-action dependent control variate using Taylor expansion. The resulting gradient estimator is identified with a special case of the above formulation, where the control variate depends on the action linearly, i.e. $\psi(s,a) = \hat{V}_\pi(s) + (a - \mu_\pi(s))^T \nabla_a \hat{Q}_\pi(s,a)|_{\mu_\pi(s)}$ where both $\hat{V}_\pi, \hat{Q}_\pi$ are estimates of the state- and action-value functions under target policy π, respectively. They are actually function approximations but their parameter dependency is ignored in the computation of the gradient. It is a better baseline than the state-value alone as the Stein control variate is capable of decreasing the variance even more.

E.8 Zero-Order Optimization

Zeroth-Order (ZO) optimization goes by many names depending on its field of application: *black-box optimization*, derivative-free optimization, pattern search, or direct search to list a few. It is arguably the most generic category of optimization methods that can be imagined. All it requires is being able to evaluate at will the scalar objective function to be maximized given a set of parameters. No analytical gradient is provided, so it can be applied to objective functions that are not continuous or differentiable. No assumption is made about the structure of the problem.

Let us consider a multi-variate scalar stochastic function $f : \mathbb{R}^n \to \mathbb{R}$. A naive approach to estimate its gradient is the forward finite difference. The input $x \in \mathbb{R}^n$ is perturbed by a small amount $\sigma > 0$ along every single component e_i individually,

$$[\hat{\nabla}_x J(x)]_i = \frac{J(x + \sigma e_i) - J(x)}{\sigma}. \tag{E.41}$$

It requires $n + 1$ function evaluations to estimate the gradient in any direction. The accuracy of this estimate depends on the number of parameters n, the intrinsic noise $|J(x) - \mathbb{E}[J(x)]| \leq \epsilon, \forall x \in \mathbb{R}^n$, and the global Lipchitz constant $\|\nabla_x \mathbb{E}[J(x)]\| \leq L, \forall x \in \mathbb{R}^n$ (Berahas et al., 2022). Specifically,

$$\|\hat{\nabla} J(x) - \nabla_x \mathbb{E}[J(x)]\|_2 \leq \frac{\sqrt{n} L \sigma}{2} + \frac{2\sqrt{n}\epsilon}{\sigma}. \tag{E.42}$$

It appears that the perturbation σ must be small but not infinitesimal, unless the function f is perfectly deterministic. It is easy to show that the optimal value is,

$$\sigma = 2\sqrt{\frac{\epsilon}{L}}. \tag{E.43}$$

Alternatively, the central finite difference could be used to estimate the gradient. It is slightly more accurate and stable but doubles the number of evaluations.

Gradient estimates based on finite differences are accurate but impracticable for many problems. Regarding policy learning, each function evaluation requires running a complete episode while a basic policy network has at least 10000 parameters. A more reasonable approach would be to sample random direction ξ in the parameter space and to pick the one that better improves the performance on average:

$$\hat{\nabla} J(x) = \mathop{\mathbb{E}}_{\xi \sim N(0,I)} \left[\frac{J(x+\sigma\xi) - J(x)}{\sigma} \right] = \frac{1}{\sigma} \mathop{\mathbb{E}}_{\xi \sim N(0,I)} [J(x+\sigma\xi)\xi] \tag{E.44}$$

It is sometimes referred to as the one-point gradient estimate as the function is evaluated once per direction. It can be demonstrated that this estimate corresponds to the gradient of the average fitness for perturbed parameters x, $\tilde{J}(x) = \mathbb{E}_{z \sim N(x,\sigma)}[J(z)]$.

Proof. The proof relies on the reparametrization trick, the log-derivative trick, and the expression of the log-likelihood of an isotropic Gaussian:

$$\begin{aligned} \nabla_x \mathop{\mathbb{E}}_{z \sim N(x,\sigma)} [J(z)] &= \int_z J(z) \nabla_x p(z|x,\sigma) \, \mathrm{d}z = \mathop{\mathbb{E}}_{z \sim N(x,\sigma)} [J(z) \nabla_x \log p(z)] \\ &= \mathop{\mathbb{E}}_{z \sim N(x,\sigma)} \left[J(z) \frac{z-x}{\sigma^2} \right] = \frac{1}{\sigma} \mathop{\mathbb{E}}_{\xi \sim N(0,I)} [J(x+\sigma\xi)\xi] \end{aligned}$$

■

The averaging smooths out the original function f. Its effect is comparable to a moving average or a low-pass filter. It is not surprising that smoothing appears naturally. Indeed, such regularization of the Lipschitz constant is a precondition for the gradient to be properly defined if the function f is not differentiable but bounded. This approach is generally called *directional Gaussian smoothing*.

Although stochastic estimates are cheap to evaluate, their accuracy is worse than finite difference methods because the sampling directions are not orthogonal. Many other ZO gradient estimators are found in the literature, but none of them is really superior (Berahas et al., 2022). The pivotal idea to make ZO optimization viable is to leverage previous estimates to bootstrap new ones instead of throwing them away. It makes sense because the parameters are only slightly updated at every iteration, which means that the gradient should have barely changed. Moreover, it ensures the gradient keeps steering in roughly the direction for a while before eventually changing, which prevents getting stuck in poor local minima. Overall, it dramatically improves the sample efficiency by reducing the variance of the gradient estimate but

introduces bias. Such optimization algorithms tend to find better solutions and have stable convergence. One of the most effective ZO optimization algorithms is ZO-Adam (Chen et al., 2019). It is a modified version of *ADAptive Moment estimation* (ADAM) that estimates the gradient according to equation (E.44). The sampling distribution is not adapted, but the first and second-order momentums are used to adjust the update rate and add inertia. The effect is expected to be similar to adjusting the covariance matrix instead.

E.9 Evolutionary Strategies

Evolutionary Strategies (ES) (Beyer & Schwefel, 2002) belongs to the big family of ZO optimization. It is a specific type of *Evolutionary Algorithms* (EA), just like generic algorithms. They mostly differ from each other by their encoding of candidate solutions. For OCP in particular, the underlying agent-environment interaction loop is completely disregarded, and the MDP formalism so ubiquitous in RL is not even involved. The individual transition steps are not accessible, nor are the complete trajectories. Within the scope of the RL, ES are similar to the vanilla policy gradient method in several aspects. The policy is a function approximation with parameters θ. Those parameters are directly optimized by gradient descent, following a stochastic approximation of the gradient. However, the policy gradient theorem does not apply since trajectories are not available. In this context, the return for a complete episode is referred to as the *fitness score*.

EA try to reproduce the Darwin evolution process. In the wild, only the fittest can survive in competition for limited resources. It starts with a population of random agents, corresponding to different sets of parameters. There are four steps per iteration: evaluation, selection, recombination, and mutation. First, all the agents interact with the environment in parallel. Secondly, the candidate policies associated with low relative fitness scores are discarded. Next, a new generation is then created by recombining the parameters of high-fitness survivors. Finally, some agents endure random mutations. This process is repeated until one of the agents solves the task.

EA are known to explore the search space extensively. No backpropagation must be performed, it is invariant to delayed or long-term rewards, and it is massively parallelizable with very little data communication. However, it has poor sample efficiency compare to RL methods because it cannot leverage the temporal structure of the problem. In the vanilla Evolutionary Strategy, the parameters are sampled according to an isotropic gaussian distribution with a fixed standard deviation. The efficiency can be further enhanced using a full covariance matrix instead, which would be automatically adapted. It reduces the variance of the gradient estimator, converges faster, and increases the fitness of the final solution. This algorithm is called *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) (Hansen & Ostermeier, 1996). The samples are spread out initially, with a large variance in the direction of interest. Then, it becomes more concentrated as the confidence in finding a good solution increases. It is closely related to ZO-Adam (cf. appendix E.8).

Alphabetical Index

Actor-critic, 87
Admittance control, 45
Artifical neural network
    Activation function, 54
    Backpropagation, 55
    Batch processing, 56
    Chain rule, 56
    Convolution kernel, 62
    Dead unit, 61
    Discrete convolution, 62
    Fully-connected, 62
    Neuron, 54
    Polling, 64
    Structured network, 62
    Transpose convolution, 63
    Universal approximator, 58
    Vanishing gradient, 60
Attention-based network
    Attention map, 266
    Cross-attention, 266
    Key, 266
    Query, 266
    Self-attention, 266
    Value, 266
Automatic differentiation, 56

Capturability, 34
Capture Point, 39
Centroidal Dynamics, 9
Classification, 49
Collocation method
    Cardinal node, 223
    Collocation condition, 223
    Collocation point, *see* Interior node
    Interior node, 223
Contact wrench, 36
Convex hull, 250
Curriculum learning, 71

Encoder-decoder architecture
    Auto-encoder, 262
    Latent space, 261
Expressive power, 50

Feedback linearization, 218
    Decoupling matrix, 219
Function approximation, 50

Gauss's principle of least constraint, 238

Hadamard product, 226
Hybrid system model
    Guard surface, *see* Switching surface
    Phase, 217
    Relabeling, 216
    Reset map, 212
    Switching surface, 212
Hybrid zero dynamics
    Hybrid invariant, 220
    Partial zero dynamics, 220
    Zero dynamics, 220

Kinematic constraint
    Drift, 238
    Friction pyramid, 248
    Painleve's paradox, 242
    Scleronomic, 25
    Second-order cone, 242

Meta-optimization, 61

Neural architecture search, 61

Poly-articulated system, 15
- Base, 17
- Body, 15
- Chain rule, 23
- Delassus matrix, 239
- End-Effector, 17
- Freeflyer, 17
- Inertia matrix, 27
- Joint, 15
- Kinematic sub-chain, 16
- Kinematic tree, 17
- Link, 15
- Mass matrix, *see* Inertia matrix

Recurrent neural network
- Cell state, 264
- Forget gate, 264
- Gating mechanism, 264
- Input gate, 264
- Output gate, 264

Regression, 49

Reinforcement learning
- Action-value, 75
- Advantage, 75
- Control policy, 67
- Curiosity, 83
- Dense reward, 69
- Distributional shift, 69
- Domain randomization, 72
- Exploration-exploitation dilemma, 80
- Fully observable, 71
- Imitation learning, 69
- Importance sampling, 285
- Off-policy, 88
- Offline *Reinforcement Learning* (RL), 68
- On-policy, 88
- Online RL, 68
- Partially observable, 73
- Policy distillation, 69
- Privileged information, 69
- Q-value, *see* Action-value
- Reality gap, 69
- Replay buffer, 275
- Return, 65
- Reward engineering, 69
- Sim-to-real transfer, 68
- Sparse reward, 70
- Value function, 74

Residual neural network
- Shortcut connection, 260
- Skip connection, *see* Shortcut connection

Spatial vector algebra
- Apparent derivative, 21
- Body Jacobian, 20
- Cartesian frame, 19
- Classical acceleration, 22
- Euler angles, 22
- Generalized coordinates, 17
- Pose, 22
- Spatial force, 20
- Spatial motion, 19
- Twist, 19
- Versor, 22
- Wrench, 20

Supervised learning
- Approximation error, 51
- Complexity penalty, 52
- Epproximation error, 52
- Estimation error, 51
- Explanatory variable, 49
- Feature, *see* Explanatory variable
- Generalization ability, 49
- Label, 49
- Overfitting, 52
- Prediction error, 51
- Training set, 49
- Underfitting, 52

Teacher-Student, 69

Tikhonov regularization, 54

Unsupervised learning, 260

Virtual constraints, 217

Bibliography

Acary, V., F. Cadoux, C. Lemaréchal, and J. Malick (2011). "A formulation of the linear discrete Coulomb friction problem via convex optimization". In: *ZAMM Zeitschrift fur Angewandte Mathematik und Mechanik* 91.2, pp. 155–175 (see page 246).

Achiam, J., D. Held, A. Tamar, and P. Abbeel (2017). "Constrained policy optimization". In: *34th International Conference on Machine Learning, ICML 2017*. Vol. 1, pp. 30–47 (see pages 93, 117, 172).

Achiam, J. and S. Sastry (2017). "Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning". In: *Deep Reinforcement Learning Workshop, NIPS 2016*, pp. 1–13 (see page 85).

Ahn, J., S. J. Jorgensen, S. H. Bang, and L. Sentis (2021). "Versatile Locomotion Planning and Control for Humanoid Robots". In: *Frontiers in Robotics and AI* 8.August, pp. 1–17 (see page 9).

Alain, G. and Y. Bengio (2014). "What Regularized Auto-Encoders Learn from the Data-Generating Distribution". In: *Journal of Machine Learning Research* 15.110, pp. 3743–3773 (see page 262).

Alessio, A. and A. Bemporad (2009). "A survey on explicit model predictive control". In: *Lecture Notes in Control and Information Sciences*. Vol. 384, pp. 345–369 (see page 101).

Amid, E., R. Anil, and M. Warmuth (2022). "LocoProp: Enhancing BackProp via Local Loss Optimization". In: *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. Ed. by G. Camps-Valls, F. J. R. Ruiz, and I. Valera. Vol. 151. Proceedings of Machine Learning Research. PMLR, pp. 9626–9642. URL: https://proceedings.mlr.press/v151/amid22a.html (see page 58).

Anderson, B. D. O. and J. B. Moore (2012). *Optimal Filtering*. Courier Corporation, p. 368 (see page 112).

Anderson, B. D. O. and J. B. Moore (2007). *Optimal Control: Linear Quadratic Methods*. Courier Corporation, p. 360 (see pages 47, 221).

Andreani, R., E. G. Birgin, J. M. Martínez, and M. L. Schuverdt (2008). "On Augmented Lagrangian Methods with General Lower-Level Constraints". In: *SIAM Journal on Optimization* 18.4, pp. 1286–1309 (see pages 136, 140).

Andrychowicz, M. et al. (2017). "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems* 2017-Decem.Nips, pp. 5049–5059 (see page 79).

Anil, C., J. Lucas, and R. Grosse (2019). "Sorting out Lipschitz function approximation". In: *36th International Conference on Machine Learning, ICML 2019* 2019-June, pp. 432–452 (see pages 121, 140).

Anitescu, M. and F. A. Potra (1997). "Formulating Dynamic Multi-Rigid-Body Contact Problems with Friction as Solvable Linear Complementarity Problems". In: *Nonlinear Dynamics* 14.3, pp. 231–247 (see pages 241, 246).

Apgar, T., P. Clary, K. Green, A. Fern, and J. Hurst (2018). "Fast Online Trajectory Optimization for the Bipedal Robot Cassie". In: *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation (see pages 9, 44, 47, 100, 110).

Apra, C., M. Serra, H. Robert, and A. Carpentier (2022). "Early rehabilitation using gait exoskeletons is possible in the neurosurgical setting, even in patients with cognitive impairment". In: *Neurochirurgie*, pp. 12–14 (see page 4).

Arechavaleta, G., E. López-Damian, and J. L. Morales (2009). "On the use of iterative LCP solvers for dry frictional contacts in grasping". In: *2009 International Conference on Advanced Robotics, ICAR 2009* (see page 246).

Arndt, K., M. Hazara, A. Ghadirzadeh, and V. Kyrki (2020). "Meta Reinforcement Learning for Sim-to-real Domain Adaptation". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2725–2731 (see page 107).

Arora, S. and P. Doshi (2021). "A survey of inverse reinforcement learning: Challenges, methods and progress". In: *Artificial Intelligence* 297, pp. 1–48 (see pages 108, 117).

Åström, K. J. and T. Hägglund (1995). *PID controllers: theory, design, and tuning*. Vol. 2. ISA: The Instrumentation, Systems, and Automation Society (see page 42).

Atkeson, C. and B. Stephens (2008). "Random Sampling of States in Dynamic Programming". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.4, pp. 924–929 (see page 104).

Atkeson, C. G. and C. Liu (2013). "Trajectory-Based Dynamic Programming". In: *Cognitive Systems Monographs*, pp. 1–15 (see page 103).

Attia, A. and S. Dayan (2018). *Global overview of Imitation Learning*. URL: https://arxiv.org/abs/1801.06503 (see page 102).

Badia, A. P. et al. (2020). "Never Give Up: Learning Directed Exploration Strategies". In: *International Conference on Learning Representations* (see pages 85, 206).

Bain, M. and C. Sammut (1999). "A Framework for Behavioural Cloning". In: *Machine Intelligence* 15, pp. 103–129 (see pages 69, 103).

Behrman, A. L., M. G. Bowden, and P. M. Nair (2006). "Neuroplasticity after spinal cord injury and training: An emerging paradigm shift in rehabilitation and walking recovery". In: *Physical Therapy* 86.10, pp. 1406–1425 (see page 2).

Bellman, R. (1957). "Markovian decision processes". In: *Indiana University Mathematics Journal* 6.4, pp. 679–684 (see page 70).

Bellman, R., I. L. Glicksberg, and O. Gross (1956). "On the "bang-bang" control problem". In: *Quarterly of Applied Mathematics* 14.1, pp. 11–18 (see page 68).

Bemporad, A., M. Morari, V. Dua, and E. N. Pistikopoulos (2000). "Explicit solution of model predictive control via multiparametric quadratic programming". In: *Proceedings of the American Control Conference*. Vol. 2. June, pp. 872–876 (see page 101).

Ben-David, S., N. Eiron, and P. M. Long (2003). "On the difficulty of approximately maximizing agreements". In: *Journal of Computer and System Sciences* 66.3, pp. 496–514 (see page 51).

Bengio, Y., J. Louradour, R. Collobert, and J. Weston (2009). "Curriculum learning". In: *ACM International Conference Proceeding Series*. Vol. 382. New York, New York, USA: ACM Press, pp. 1–8 (see page 78).

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166 (see page 60).

Bentley, J. L. (1975). "Multidimensional Binary Search Trees Used for Associative Searching". In: *Communications of the ACM* 18.9, pp. 509–517 (see page 101).

Berahas, A. S., L. Cao, K. Choromanski, and K. Scheinberg (2022). "A Theoretical and Empirical Comparison of Gradient Approximations in Derivative-Free Optimization". In: *Foundations of Computational Mathematics* 22.2, pp. 507–560 (see pages 294, 295).

Berenson, D., S. S. Srinivasa, D. Ferguson, and J. J. Kuffner (2009). "Manipulation planning on constraint manifolds". In: *IEEE International Conference on Robotics and Automation*, pp. 625–632 (see page 101).

Bertsekas, D. P. (1976). "Multiplier methods: A survey". In: *Automatica* (see pages 136, 227, 229, 231).

– (1982). *Constrained Optimization and Lagrange Multiplier Methods*. Ed. by A. Press. Elsevier, p. 416 (see pages 136, 227–229).

Beyer, H.-G. and H.-P. Schwefel (2002). "Evolution strategies: a comprehensive introduction". In: *Natural Computing* 1.1, pp. 3–52 (see page 296).

Beygelzimer, A., V. Dani, T. Hayes, J. Langford, and B. Zadrozny (2005). "Error limiting reductions between classification tasks". In: *ICML 2005 - Proceedings of the 22nd International Conference on Machine Learning*, pp. 49–56 (see page 103).

Bezdek, J. and R. Hathaway (2003). "Convergence of alternating optimization". In: *Neural, Parallel and Scientific Computations* 11.4, pp. 351–368 (see page 137).

Björck, Å. and C. Bowie (1971). "An Iterative Algorithm for Computing the Best Estimate of an Orthogonal Matrix". In: *SIAM Journal on Numerical Analysis* 8.2, pp. 358–364 (see page 121).

Blanco-Claraco, J. L. (2021). "A tutorial on SE(3) transformation parameterizations and on-manifold optimization". In: 3. URL: http://arxiv.org/abs/2103.15980 (see page 179).

Boer, T. de (2012). "Foot placement in robotic bipedal locomotion". PhD thesis. Delft University of Technology (see pages 9, 33, 39, 41, 42, 45, 110).

Bojarski, M. et al. (2016). *End to End Learning for Self-Driving Cars*. URL: http://arxiv.org/abs/1604.07316 (see page 69).

Borg, I. and P. J. F. Groenen (2005). *Modern Multidimensional Scaling: Theory and Applications*. Springer Series in Statistics. New York, NY: Springer New York, p. 614 (see page 181).

Bousmalis, K. et al. (2018). "Using simulation and domain adaptation to improve efficiency of deep robotic grasping". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4243–4250 (see page 106).

Boyd, S. (2010). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends® in Machine Learning* 3.1, pp. 1–122 (see pages 109, 137).

Bretl, T., G. Arechavaleta, A. Akce, and J.-P. Laumond (2010). "Comments on "an optimality principle governing human walking"". In: *IEEE Transactions on Robotics* 26.6, pp. 1105–1106 (see page 31).

Bridle, J. S. (1990). "Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition". In: *Neurocomputing* C, pp. 227–236 (see page 61).

Budhiraja, R., J. Carpentier, and N. Mansard (2019a). "Dynamics consensus between centroidal and whole-body models for locomotion of legged robots". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2019-May, pp. 6727–6733 (see page 48).

Budhiraja, R., J. Carpentier, C. Mastalli, and N. Mansard (2019b). "Differential Dynamic Programming for Multi-Phase Rigid Contact Dynamics". In: *IEEE-RAS International Conference on Humanoid Robots*. Vol. 2018-Novem. IEEE, pp. 53–58 (see pages 9, 211).

Burda, Y., H. Edwards, A. Storkey, and O. Klimov (2019). "Exploration by random network distillation". In: *7th International Conference on Learning Representations (ICLR 2019)*, pp. 1–17 (see page 84).

Calabrò, R. S. et al. (2016). "Robotic gait rehabilitation and substitution devices in neurological disorders: where are we now?" In: *Neurological Sciences* 37.4, pp. 503–514 (see pages 1, 2).

Campo, P. J. and M. Morari (1987). "Robust Model Predictive Control". In: *IEEE American control conference*, pp. 1021–1026 (see page 112).

Caron, S. (2020). "Biped Stabilization by Linear Feedback of the Variable-Height Inverted Pendulum Model". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 9782–9788 (see pages 9, 40, 45).

Caron, S., A. Escande, L. Lanari, and B. Mallein (2020). "Capturability-Based Pattern Generation for Walking With Variable Height". In: *IEEE Transactions on Robotics* 36.2, pp. 517–536 (see page 34).

Caron, S. and A. Kheddar (2017). "Dynamic walking over rough terrains by nonlinear predictive control of the floating-base inverted pendulum". In: *IEEE International Conference on Intelligent Robots and Systems* 2017-Septe, pp. 5017–5024 (see pages 38, 47, 48, 100, 112).

Caron, S., A. Kheddar, and O. Tempier (2019). "Stair climbing stabilization of the HRP-4 humanoid robot using whole-body admittance control". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2019-May. IEEE, pp. 277–283 (see page 45).

Caron, S., Q. C. Pham, and Y. Nakamura (2017). "ZMP Support Areas for Multicontact Mobility under Frictional Constraints". In: *IEEE Transactions on Robotics* 33.1, pp. 67–80 (see pages 34–36, 39).

Caron, S., Q. C. Pham, and Y. Nakamura (2015). "Leveraging cone double description for multi-contact stability of humanoids with applications to statics and dynamics". In: *Robotics: Science and Systems*. Vol. 11. 8. Robotics: Science and Systems Foundation (see pages 36, 113).

Carpentier, J., R. Budhiraja, and N. Mansard (2021). "Proximal and Sparse Resolution of Constrained Dynamic Equations". In: *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation (see page 239).

Carpentier, J. and N. Mansard (2018). "Analytical Derivatives of Rigid Body Dynamics Algorithms". In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania (see pages 15, 17, 100).

Carpentier, J., G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard (2019). "The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". In: *Proceedings*

of the 2019 IEEE/SICE International Symposium on System Integration, SII 2019, pp. 614–619 (see pages 13, 166).

Carpentier, J., S. Tonneau, M. Naveau, O. Stasse, and N. Mansard (2016). "A versatile and efficient pattern generator for generalized legged locomotion". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2016-June, pp. 3555–3561 (see page 100).

Castañeda, F., M. Wulfman, A. Agrawal, T. Westenbroek, S. Sastry, C. Tomlin, and K. Sreenath (2020). "Improving Input-Output Linearizing Controllers for Bipedal Robots via Reinforcement Learning". In: *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. Ed. by A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger. Vol. 120. Proceedings of Machine Learning Research. PMLR, pp. 990–999 (see page 121).

Castillo, G. A., B. Weng, W. Zhang, and A. Hereid (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)*. IEEE, pp. 5136–5143 (see pages 11, 34, 114).

Censor, Y. (1977). "Pareto Optimality in Multiobjective Problems". In: *Applied Mathematics and Optimization*, pp. 41–59 (see page 67).

Charalambous, C. P. (2014). "The major determinants in normal and pathological gait". In: *Classic Papers in Orthopaedics*. Ed. by P. A. Banaszkiewicz and D. F. Kader. Vol. 10. 15. London: Springer London, pp. 403–405 (see page 30).

Chen, X., S. Liu, K. Xu, X. Li, X. Lin, M. Hong, and D. Cox (2019). "ZO-AdaMM: Zeroth-order adaptive momentum method for black-box optimization". In: *Advances in Neural Information Processing Systems* 32.NeurIPS, pp. 1–30 (see page 296).

Chevallereau, C., D. Djoudi, and J. W. Grizzle (2008). "Stable bipedal walking with foot rotation through direct regulation of the zero moment point". In: *IEEE Transactions on Robotics* 24.2, pp. 390–401 (see page 45).

Chisari, E., A. Liniger, A. Rupenyan, L. Van Gool, and J. Lygeros (2021). "Learning from Simulation, Racing in Reality". In: pp. 8046–8052 (see page 124).

Cho, K., B. van Merriënboer, D. Bahdanau, and Y. Bengio (2014). "On the properties of neural machine translation: Encoder–decoder approaches". In: *Proceedings of SSST 2014 - 8th Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111 (see page 264).

Chou, P. W., D. Maturana, and S. Scherer (2017). "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution". In: *34th International Conference on Machine Learning, ICML 2017* 2, pp. 1386–1396 (see page 82).

Chow, Y., O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh (2019). "Lyapunov-based Safe Policy Optimization for Continuous Control". In: *ICML Workshop RL4RealLife* (see pages 117, 120).

Cisse, M., P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier (2017). "Parseval networks: Improving robustness to adversarial examples". In: *34th International Conference on Machine Learning, ICML 2017* 2, pp. 1423–1432 (see page 121).

Cobbe, K. W., J. Hilton, O. Klimov, and J. Schulman (2021). "Phasic Policy Gradient". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 2020–2027 (see page 96).

Cohen, J., E. Rosenfeld, and J. Z. Kolter (2019). "Certified adversarial robustness via randomized smoothing". In: *36th International Conference on Machine Learning, ICML 2019*, pp. 2323–2356 (see page 120).

Collins, S. H., P. G. Adamczyk, and A. D. Kuo (2009). "Dynamic arm swinging in human walking". In: *Proceedings of the Royal Society B: Biological Sciences* 276.1673, pp. 3679–3688 (see page 30).

Cominetti, R. and J. P. Dussault (1994). "Stable exponential-penalty algorithm with superlinear convergence". In: *Journal of Optimization Theory and Applications* 83.2, pp. 285–309 (see page 172).

Conesa, L., Ú. Costa, E. Morales, D. J. Edwards, M. Cortes, D. Leán, M. Bernabeu, and J. Medina (2012). "An observational report of intensive robotic and manual gait training in sub-acute stroke". In: *Journal of NeuroEngineering and Rehabilitation* 9.1, p. 13 (see page 1).

Cooman, B. D., J. Suykens, and A. Ortseifen (2021). "Improving temporal smoothness of deterministic reinforcement learning policies with continuous actions". In: *Benelux Conference on Artificial Intelligence and Belgian Dutch Conference on Machine Learning (BNAIC/BENELEARN 2021)*, pp. 1–24 (see pages 124, 161).

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to Algorithms*. 3rd. MIT Press (see page 17).

Cottle, R. W., J.-S. Pang, and R. E. Stone (2009). *The Linear Complementarity Problem*. Boston, MA: Society for Industrial and Applied Mathematics, pp. 1873–1878 (see page 246).

Da, X. and J. Grizzle (2019). "Combining trajectory optimization, supervised machine learning, and model structure for mitigating the curse of dimensionality in the control of bipedal robots". In: *The International Journal of Robotics Research* 38.9, pp. 1063–1097 (see page 108).

Dai, H., A. Valenzuela, and R. Tedrake (2014). "Whole-body Motion Planning with Simple Dynamics and Full Kinematics". In: *IEEE-RAS International Conference on Humanoid Robots* (see pages 47, 100).

Dalal, G., K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa (2018). *Safe Exploration in Continuous Action Spaces*. URL: http://arxiv.org/abs/1801.08757 (see pages 117, 163).

Dalibard, S., A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J. P. Laumond (2013). "Dynamic walking and whole-body motion planning for humanoid robots: An integrated approach". In: *International Journal of Robotics Research* 32.9-10, pp. 1089–1103 (see page 9).

Dantec, E. et al. (2021). "Whole Body Model Predictive Control with a Memory of Motion: Experiments on a Torque-Controlled Talos". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 8202–8208 (see pages 100, 102, 132).

Daumé III, H., J. Langford, and D. Marcu (2009). "Search-based structured prediction". In: *Machine Learning* (see pages 104, 108).

Degris, T., P. M. Pilarski, and R. S. Sutton (2012a). "Model-Free reinforcement learning with continuous action in practice". In: *Proceedings of the American Control Conference*, pp. 2177–2182 (see page 283).

Degris, T., M. White, and R. S. Sutton (2012b). "Off-policy actor-critic". In: *Proceedings of the 29th International Conference on Machine Learning, ICML 2012* 1, pp. 457–464 (see pages 284–286).

Deisenroth, M. P. and C. E. Rasmussen (2011). "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *Proceedings of the International Conference on Machine Learning*, pp. 465–472 (see pages 86, 269).

Dette, H. and N. Henze (1989). "The limit distribution of the largest nearest-neighbour link in the unit d -cube". In: *Journal of Applied Probability* 26.1, pp. 67–80 (see page 130).

Dewey, D. (2014). "Reinforcement Learning and the Reward Engineering Principle". In: *2014 AAAI Spring Symposium Series*, pp. 13–16 (see page 69).

Di Pillo, G. and L. Grippo (1989). "Exact Penalty Functions in Constrained Optimization". In: *SIAM Journal on Control and Optimization* 27.6, pp. 1333–1360 (see page 136).

Diehl, M., H. G. Bock, and J. P. Schlöder (2005). "A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control". In: *SIAM Journal on Control and Optimization* 43.5, pp. 1714–1736 (see page 99).

Dirkse, S. P. and M. C. Ferris (1995). "The path solver: A nonmonotone stabilization scheme for mixed complementarity problems". In: *Optimization Methods and Software* 5.2, pp. 123–156 (see page 246).

Dong, C., C. C. Loy, K. He, and X. Tang (2015). "Image super-resolution using deep convolutional networks". In: *IEEE transactions on pattern analysis and machine intelligence* 38.2, pp. 295–307 (see page 146).

Drucker, H. and Y. L. Cun (1992). *Improving Generalization Performance Using Double Backpropagation* (see page 123).

Duburcq, A. (2019). *Jiminy: a fast and portable Python/C++ simulator of poly-articulated systems with OpenAI Gym interface for reinforcement learning*. URL: https://github.com/duburcqa/jiminy (see pages 166, 240).

Dumoulin, V. and F. Visin (2016). *A guide to convolution arithmetic for deep learning*. URL: http://arxiv.org/abs/1603.07285 (see pages 62, 63).

Dwyer, R. A. (1991). "Higher-dimensional voronoi diagrams in linear expected time". In: *Discrete & Computational Geometry* 6.1, pp. 343–367 (see page 130).

Eiben, A. and J. Smith (2015). *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–294 (see page 82).

Englsberger, J., C. Ott, and A. Albu-Schäffer (2015). "Three-Dimensional Bipedal Walking Control Based on Divergent Component of Motion". In: *IEEE Transactions on Robotics* 31.2, pp. 355–368 (see page 45).

Englsberger, J., C. Ott, M. A. Roa, A. Albu-Schäffer, and G. Hirzinger (2011). "Bipedal walking control based on capture point dynamics". In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 4420–4427 (see pages 45, 111).

Enzenhöfer, A., S. Andrews, M. Teichmann, and J. Kövecses (2018). "Comparison of mixed linear complementarity problem solvers for multibody simulations with contact". In: *VRIPHYS 2018 - 14th Workshop on Virtual Reality Interactions and Physical Simulations, VRIPHYS 2018*, pp. 11–20 (see page 247).

Erez, T., K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov (2013). "An integrated system for real-time model predictive control of humanoid robots". In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, pp. 292–299 (see pages 99, 100).

Everett, M., B. Lutjens, and J. P. How (2021). "Certifiable Robustness to Adversarial State Uncertainty in Deep Reinforcement Learning". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–18 (see page 120).

Fang, J., Y. Sun, Q. Zhang, K. Peng, Y. Li, W. Liu, and X. Wang (2021). "FNA++: Fast Network Adaptation via Parameter Remapping and Architecture Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.9, pp. 2990–3004 (see page 61).

Fawzi, A. et al. (2022). "Discovering faster matrix multiplication algorithms with reinforcement learning". In: *Nature* 610.7930, pp. 47–53 (see page 65).

Featherstone, R. (2008). *Rigid Body Dynamics Algorithms*. Ed. by Springer. Boston, MA: Springer US, p. 276 (see pages 15, 17, 28).

Feigin, V. L. et al. (2014). "Global and regional burden of stroke during 1990-2010: Findings from the Global Burden of Disease Study 2010". In: *The Lancet* 383.9913, pp. 245–255 (see page 1).

Feller, C. and C. Ebenbauer (2017). "Relaxed Logarithmic Barrier Function Based Model Predictive Control of Linear Systems". In: *IEEE Transactions on Automatic Control* 62.3, pp. 1223–1238 (see page 172).

Feng, S., X. Xinjilefu, C. G. Atkeson, and J. Kim (2016). "Robust dynamic walking using online foot step optimization". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5373–5378 (see page 111).

Ferigo, D., R. Camoriano, P. M. Viceconte, D. Calandriello, S. Traversaro, L. Rosasco, and D. Pucci (2021). "On the emergence of whole-body strategies from humanoid robot push-recovery learning". In: *IEEE Robotics and Automation Letters* 6.4, pp. 8561–8568 (see pages 11, 115, 193, 195, 198).

Finet, S. (2018). "Contribution to the Control of Biped Robots". PhD thesis. Université Paris sciences et lettres (see pages 29, 43, 211, 218).

Flores, P., M. MacHado, E. Seabra, and M. Tavares Da Silva (2011). "A parametric study on the baumgarte stabilization method for forward dynamics of constrained multibody systems". In: *Journal of Computational and Nonlinear Dynamics* 6.1, pp. 1–9 (see page 238).

Fujimoto, S., H. van Hoof, and D. Meger (2018). "Addressing Function Approximation Error in Actor-Critic Methods". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 1587–1596 (see page 88).

Fujita, Y. and S. I. Maeda (2018). "Clipped Action Policy Gradient". In: *35th International Conference on Machine Learning, ICML 2018* 4, pp. 2602–2614 (see page 82).

Galashov, A., J. Merel, and N. Heess (2022). "Data augmentation for efficient learning from parametric experts". In: URL: http://arxiv.org/abs/2205.11448 (see page 144).

Gehring, J., M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin (2017). "Convolutional sequence to sequence learning". In: *34th International Conference on Machine Learning, ICML 2017*. Vol. 3, pp. 2029–2042 (see page 267).

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger. Vol. 27. Curran Associates, Inc. (see page 263).

Goodfellow, I. J., J. Shlens, and C. Szegedy (2015). "Explaining and Harnessing Adversarial Examples". In: *3rd International Conference on Learning Representations, ICLR 2015*. Ed. by Y. Bengio and Y. LeCun (see pages 64, 119).

Goodman, R. N., J. C. Rietschel, A. Roy, B. C. Jung, J. Diaz, R. F. Macko, and L. W. Forrester (2014). "Increased reward in ankle robotics training enhances motor control and cortical efficiency in stroke". In: *Journal of Rehabilitation Research and Development* 51.2, pp. 213–227 (see page 2).

Goswami, A. and P. Vadakkepat (2019). *Humanoid robotics: a reference*. Springer (see pages 3, 15, 29, 30, 33, 41, 42, 45, 110, 211).

Greff, K., R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber (2017). "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10, pp. 2222–2232 (see page 264).

Grimmer, M., R. Riener, C. J. Walsh, and A. Seyfarth (2019). "Mobility related physical and functional losses due to aging and disease - A motivation for lower limb exoskeletons". In: *Journal of NeuroEngineering and Rehabilitation* 16.1, p. 2 (see page 2).

Grüne, L. and J. Pannek (2011). *Nonlinear Model Predictive Control*. Communications and Control Engineering. London: Springer London (see page 46).

Gu, S., T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine (2017). "Q-Prop: Sample-efficient policy gradient with an off-policy critic". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–13 (see pages 92, 294).

Gurriet, T. et al. (2018). "Towards Restoring Locomotion for Paraplegics: Realizing Dynamically Stable Walking on Exoskeletons". In: *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, pp. 2804–2811 (see pages 10, 44, 109, 222).

Gustafson, J. L. and I. Yonemoto (2017). "Beating Floating Point at its Own Game: Posit Arithmetic". In: *Supercomputing Frontiers and Innovations* 4.2, pp. 71–86 (see page 64).

Györfi, L., N. Henze, and H. Walk (2019). "The limit distribution of the maximum probability nearest-neighbour ball". In: *Journal of Applied Probability* 56.2, pp. 574–589 (see page 130).

Ha, D. and J. Schmidhuber (2018). "Recurrent World Models Facilitate Policy Evolution". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Montreal, Canada: Curran Associates Inc., pp. 2455–2467 (see pages 12, 86, 269).

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *35th*

International Conference on Machine Learning, ICML 2018 5, pp. 2976–2989 (see page 81).

Hachiya, H., T. Akiyama, M. Sugiayma, and J. Peters (2009). "Adaptive importance sampling for value function approximation in off-policy reinforcement learning". In: *Neural Networks* 22.10, pp. 1399–1410 (see page 91).

Haider, T., F. S. Roza, D. Eilers, K. Roscher, and S. Günnemann (2021). "Domain shifts in reinforcement learning: Identifying disturbances in environments". In: *CEUR Workshop Proceedings*. Vol. 2916 (see page 106).

Han, S. P. and O. L. Mangasarian (1979). "Exact penalty functions in nonlinear programming". In: *Mathematical Programming* 17.1, pp. 251–269 (see page 136).

Hanin, B. (2019). "Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations". In: *Mathematics* 7.10, p. 992 (see page 59).

Hansen, N. and A. Ostermeier (1996). "Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation". In: *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE, pp. 312–317 (see page 296).

Hargraves, C. and S. Paris (1987). "Direct trajectory optimization using nonlinear programming and collocation". In: *Journal of Guidance, Control, and Dynamics* 10.4, pp. 338–342 (see page 222).

Hartley, R., M. Ghaffari, R. M. Eustice, and J. W. Grizzle (2020). "Contact-aided invariant extended Kalman filtering for robot state estimation". In: *International Journal of Robotics Research* 39.4, pp. 402–430 (see page 112).

Hays, R. T. and M. J. Singer (1989). *Simulation Fidelity in Training System Design: Bridging the gap between reality and training*. Ed. by R. T. Hays and M. J. Singer. Recent Research in Psychology. New York, NY: Springer New York (see page 106).

He, J. B., Q. G. Wang, and T. H. Lee (2000). "PI/PID controller tuning via LQR approach". In: *Chemical Engineering Science* 55.13, pp. 2429–2439 (see page 47).

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (see page 260).

Heess, N. et al. (2017). *Emergence of Locomotion Behaviours in Rich Environments*. URL: https://arxiv.org/abs/1707.02286 (see page 113).

Heirung, T. A. N., J. A. Paulson, J. O'Leary, and A. Mesbah (2018). "Stochastic model predictive control — how does it work?" In: *Computers and Chemical Engineering* 114, pp. 158–170 (see page 112).

Herdt, A. (2012). "Model predictive control of a humanoid robot". PhD thesis. École nationale supérieure des mines de Paris (see page 99).

Herdt, A., N. Perrin, and P. B. Wieber (2010). "Walking without thinking about it". In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 190–195 (see pages 47, 100).

Hereid, A. and A. D. Ames (2017). "FROST: Fast robot optimization and simulation toolkit". In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 2017-Septe. IEEE, pp. 719–726 (see pages 9, 109).

Hereid, A., E. A. Cousineau, C. M. Hubicki, and A. D. Ames (2016a). "3D dynamic walking with underactuated humanoid robots: A direct collocation framework for optimizing hybrid zero dynamics". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2016-June, pp. 1447–1454 (see pages 222, 225).

Hereid, A., C. M. Hubicki, E. A. Cousineau, and A. D. Ames (2018). "Dynamic Humanoid Locomotion: A Scalable Formulation for HZD Gait Optimization". In: *IEEE Transactions on Robotics* 34.2, pp. 370–387 (see pages 9, 30, 211, 218).

Hereid, A., C. M. Hubicki, E. A. Cousineau, J. W. Hurst, and A. D. Ames (2015). "Hybrid zero dynamics based multiple shooting optimization with applications to robotic walking". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2015-June. June, pp. 5734–5740 (see page 222).

Hereid, A., S. Kolathaya, and A. D. Ames (2016b). "Online optimal gait generation for bipedal walking robots using legendre pseudospectral optimization". In: *2016 IEEE 55th Conference on Decision and Control, CDC 2016*. IEEE, pp. 6173–6179 (see page 9).

Herzog, A., N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti (2016). "Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid". In: *Autonomous Robots* 40.3, pp. 473–491 (see page 10).

Hirai, K., M. Hirose, Y. Haikawa, and T. Takenaka (1988). "The development of Honda humanoid robot". In: *IEEE International Conference on Robotics and Automation*. Vol. 2. IEEE, pp. 1321–1326 (see page 44).

Hirukawa, H., S. Hattori, K. Harada, S. Kajita, K. Kaneko, F. Kanehiro, K. Fujiwara, and M. Morisawa (2006). "A universal stability criterion of the foot contact of legged robots - Adios ZMP". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2006. May. IEEE, pp. 1976–1983 (see pages 36, 39).

Hobbs, B. and P. Artemiadis (2020). "A Review of Robot-Assisted Lower-Limb Stroke Therapy: Unexplored Paths and Future Directions in Gait Rehabilitation". In: *Frontiers in Neurorobotics* 14.April (see page 2).

Hochreiter, S. and J. Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780 (see page 264).

Hong, M. and Z.-Q. Luo (2017). "On the linear convergence of the alternating direction method of multipliers". In: *Mathematical Programming* 162.1-2, pp. 165–199 (see page 140).

Hong, M., Z.-Q. Luo, and M. Razaviyayn (2015). "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 3836–3840 (see page 140).

Hornik, K. (1991). "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2, pp. 251–257 (see page 59).

Hornik, K., M. Stinchcombe, and H. White (1989). "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5, pp. 359–366 (see page 59).

Hurmuzlu, Y. and D. B. Marghitu (1994). "Rigid Body Collisions of Planar Kinematic Chains With Multiple Contact Points". In: *The International Journal of Robotics Research* 13.1, pp. 82–92 (see page 215).

Hussein, A., M. M. Gaber, E. Elyan, and C. Jayne (2017). "Imitation learning: A survey of learning methods". In: *ACM Computing Surveys* 50.2, pp. 1–35 (see pages 69, 102).

Hutter, M. et al. (2016). "ANYmal - A highly mobile and dynamic quadrupedal robot". In: *IEEE International Conference on Intelligent Robots and Systems* 2016-Novem, pp. 38–44 (see page 5).

Huynh, D. Q. (2009). "Metrics for 3D rotations: Comparison and analysis". In: *Journal of Mathematical Imaging and Vision* 35.2, pp. 155–164 (see page 178).

Huynh, V., G. Burger, Q. V. Dang, R. Pelgé, G. Boéris, J. W. Grizzle, A. D. Ames, and M. Masselin (2021). "Versatile Dynamic Motion Generation Framework: Demonstration With a Crutch-Less Exoskeleton on Real-Life Obstacles at the Cybathlon 2020 With a Complete Paraplegic Person". In: *Frontiers in Robotics and AI* 8.September, pp. 1–15 (see pages 9, 109, 110, 126, 211).

Hwangbo, J., J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter (2019). "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (see pages 114, 207).

Hyon, S.-H., R. Osu, and Y. Otaka (2009). "Integration of multi-level postural balancing on humanoid robots". In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1549–1556 (see page 111).

Ioffe, S. and C. Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *32nd International Conference on Machine Learning, ICML 2015* 1, pp. 448–456 (see page 260).

Jacobson, D. harris and D. Q. Mayne (1970). *Differential Dynamic Programming*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company (see page 222).

Jaegle, A., F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira (2021). "Perceiver: General Perception with Iterative Attention". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 4651–4664 (see page 267).

Jakobi, N., P. Husbands, and I. Harvey (1995). "Noise and the reality gap: The use of simulation in evolutionary robotics". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 929, pp. 704–720 (see page 69).

Jin, M. and J. Lavaei (2020). "Stability-Certified Reinforcement Learning: A Control-Theoretic Perspective". In: *IEEE Access* 8, pp. 229086–229100 (see pages 121, 123, 161).

Kahn, G., T. Zhang, S. Levine, and P. Abbeel (2017). "PLATO: Policy learning using adaptive trajectory optimization". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3342–3349 (see page 105).

Kajita, S., F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa (2003). "Biped walking pattern generation by using preview control of zero-moment point". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2. IEEE, pp. 1620–1626 (see pages 9, 46, 47, 111).

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: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 1. IEEE, pp. 239–240 (see pages 40, 111).

Kakade, S. (2002). "A natural policy gradient". In: *Advances in Neural Information Processing Systems* (see page 57).

Kakade, S. M. and J. Langford (2002). "Approximately Optimal Approximate Reinforcement Learning". In: *Proceedings of the International Conference of Machine Learning (ICML)*, pp. 267–274 (see pages 92–94).

Kalashnikov, D. et al. (2018). "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *Proceedings of The 2nd Conference on Robot Learning*. Vol. 87. Proceedings of Machine Learning Research. PMLR, pp. 651–673 (see page 284).

Kalman, R. E. (1960). "Contributions to the theory of optimal control". In: *Boletín de la Sociedad Matemática Mexicana* 5.2, pp. 102–119 (see page 221).

Kaown, D. and J. Liu (2009). "A fast geometric algorithm for finding the minimum distance between two convex hulls". In: *Proceedings of the IEEE Conference on Decision and Control* September, pp. 1189–1194 (see page 192).

Kazdadi, S. E., J. Carpentier, and J. Ponce (2021). "Equality Constrained Differential Dynamic Programming". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 8053–8059 (see pages 109, 222).

Kelly, M. (2017). "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation". In: *SIAM Review* 59.4, pp. 849–904 (see page 222).

Kelly, M. P. (2015). "Transcription Methods for Trajectory Optimization: a beginners tutorial". In: pp. 1–14. URL: http://arxiv.org/abs/1707.00284 (see pages 221, 222).

Khadka, S. and K. Tumer (2018). "Evolution-guided policy gradient in reinforcement learning". In: *Advances in Neural Information Processing Systems* 2018-Decem, pp. 1188–1200 (see page 92).

Khatib, O. (1987). "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation". In: *IEEE Journal on Robotics and Automation* 3.1, pp. 43–53 (see page 239).

Kidger, P. and T. Lyons (2020). "Universal Approximation with Deep Narrow Networks". In: *Proceedings of Thirty Third Conference on Learning Theory*. Ed. by J. Abernethy and S. Agarwal. Vol. 125. Proceedings of Machine Learning Research. PMLR, pp. 2306–2327 (see page 59).

Kim, D., S. J. Jorgensen, J. Lee, J. Ahn, J. Luo, and L. Sentis (2020). "Dynamic locomotion for passive-ankle biped robots and humanoids using whole-body locomotion control". In: *International Journal of Robotics Research* 39.8, pp. 936–956 (see page 10).

Kingma, D. P. and J. L. Ba (2015). "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15 (see page 58).

Kingma, D. P. and M. Welling (2014). "Auto-encoding variational bayes". In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pp. 1–14 (see pages 77, 262).

Kleim, J. A. and T. A. Jones (2008). "Principles of experience-dependent neural plasticity: Implications for rehabilitation after brain damage". In: *Journal of Speech, Language, and Hearing Research* 51.1 (see page 2).

Koenemann, J. et al. (2015). "Whole-body model-predictive control applied to the HRP-2 humanoid". In: *IEEE International Conference on Intelligent Robots and Systems*. Vol. 2015-Decem. IEEE, pp. 3346–3351 (see pages 99, 100).

Konda, V. R. and J. N. Tsitsiklis (2000). "Actor-critic algorithms". In: *Advances in Neural Information Processing Systems*, pp. 1008–1014 (see page 87).

Kramer, M. A. (1991). "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE Journal* 37.2, pp. 233–243 (see page 260).

Krause, M., J. Englsberger, P. B. Wieber, and C. Ott (2012). "Stabilization of the Capture Point dynamics for bipedal walking based on model predictive control". In: *IFAC Proceedings Volumes* 45.22, pp. 165–171 (see page 111).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2017). "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6, pp. 84–90 (see page 62).

Kudruss, M., M. Naveau, O. Stasse, N. Mansard, C. Kirches, P. Soueres, and K. Mombaur (2015). "Optimal control for whole-body motion generation using center-of-mass dynamics for predefined multi-contact configurations". In: *IEEE-RAS International Conference on Humanoid Robots*. Vol. 2015-Decem. IEEE, pp. 684–689 (see page 108).

Kuo, A. D. (2007). "The six determinants of gait and the inverted pendulum analogy: A dynamic walking perspective". In: *Human Movement Science* 26.4, pp. 617–656 (see page 30).

Kurakin, A., I. Goodfellow, and S. Bengio (2017). "Adversarial examples in the physical world". In: *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings* (see page 124).

Kwakkel, G., B. J. Kollen, and H. I. Krebs (2008). "Effects of robot-assisted therapy on upper limb recovery after stroke: A systematic review". In: *Neurorehabilitation and Neural Repair* 22.2, pp. 111–121 (see page 1).

Lacoursière, C. (2003). "Splitting methods for dry frictional contact problems in rigid multibody systems: Preliminary performance results". In: *Conference Proceedings from SIGRAD2003, November*, pp. 20–21 (see page 247).

Laderman, J., V. Pan, and X. H. Sha (1992). "On practical algorithms for accelerated matrix multiplication". In: *Linear Algebra and Its Applications* 162-164.C, pp. 557–588 (see page 65).

Lam, M. Y. et al. (2015). "Perceptions of technology and its use for therapeutic application for individuals with hemiparesis: Findings from adult and pediatric focus groups". In: *JMIR Rehabilitation and Assistive Technologies* 2.1, e1 (see page 2).

Lan, Q., Y. Pan, A. Fyshe, and M. White (2020). "Maxmin Q-learning: Controlling the Estimation Bias of Q-learning". In: *International Conference on Learning Representations*, pp. 1–20 (see pages 98, 274).

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (see page 62).

Lee, J. et al. (2018). "DART: Dynamic Animation and Robotics Toolkit". In: *The Journal of Open Source Software* 3.22, p. 500 (see pages 12, 106).

Lee, S., M. Park, K. Lee, and J. Lee (2019). "Scalable muscle-actuated human simulation and control". In: *ACM Transactions on Graphics* 38.4 (see pages 188, 206).

Lehman, J. and K. O. Stanley (2011). "Abandoning objectives: Evolution through the search for novelty alone". In: *Evolutionary Computation* 19.2, pp. 189–222 (see pages 83, 84, 206).

Lembono, T. S., C. Mastalli, P. Fernbach, N. Mansard, and S. Calinon (2020). "Learning How to Walk: Warm-starting Optimal Control Solver with Memory of Motion". In: *IEEE International Conference on Robotics and Automation*. March, pp. 1357–1363 (see page 101).

Levine, S. and V. Koltun (2013). "Guided Policy Search". In: *Proceedings of the 30th International Conference on Machine Learning* (see pages 89, 103–105, 109).

Li, J., R. Zhao, J. T. Huang, and Y. Gong (2014). "Learning small-size DNN with output-distribution-based criteria". In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* September, pp. 1910–1914 (see page 69).

Li, Z., C. Zhou, Q. Zhu, and R. Xiong (2017). "Humanoid Balancing Behavior Featured by Underactuated Foot Motion". In: *IEEE Transactions on Robotics (T-RO)* 33.2, pp. 298–312 (see page 111).

Li, Z., X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath (2021). "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2811–2817 (see pages 11, 34, 114, 116, 118, 198).

Liang, E. et al. (2018). "RLlib: Abstractions for Distributed Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 3053–3062 (see page 198).

Lillicrap, T., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2016). *Continuous control with deep reinforcement learning* (see pages 88, 113, 161, 287).

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (1960). "A new approach to linear filtering and prediction problems". In: *Journal of Fluids Engineering, Transactions of the ASME* 82.1, pp. 35–45 (see page 112).

Liu, H., Y. Feng, Y. Mao, D. Zhou, J. Peng, and Q. Liu (2018). "Action-dependent Control Variates for Policy Optimization via Stein Identity". In: *International Conference on Learning Representations* (see pages 92, 206, 290, 292, 293).

Liu, Y., J. Ding, and X. Liu (2020a). "A Constrained Reinforcement Learning Based Approach for Network Slicing". In: *Proceedings - International Conference on Network Protocols, ICNP* 2020-Octob (see page 118).

– (2020b). "IPO: Interior-point policy optimization under constraints". In: *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, pp. 4940–4947 (see pages 118, 146, 170).

Liu, Y., A. Halev, and X. Liu (2021a). "Policy Learning with Constraints in Model-free Reinforcement Learning: A Survey". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, {IJCAI-21}*. Ed. by Z.-H. Zhou. International Joint Conferences on Artificial Intelligence Organization, pp. 4508–4515 (see page 118).

Liu, Y., Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan (2021b). "A Survey on Evolutionary Neural Architecture Search". In: *IEEE Transactions on Neural Networks and Learning Systems* 1, pp. 1–21 (see page 61).

Lohmeier, S., T. Buschmann, and H. Ulbrich (2009). "System design and control of anthropomorphic walking Robot LOLA". In: *IEEE/ASME Transactions on Mechatronics* 14.6, pp. 658–666 (see page 10).

Ma, L.-K., Z. Yang, X. Tong, B. Guo, and K. Yin (2021). "Learning and Exploring Motor Skills with Spacetime Bounds". In: *Computer Graphics Forum* 40, pp. 251–263. URL: http://arxiv.org/abs/2103.16807 (see pages 118, 160, 168, 173, 185).

Madry, A., A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu (2018). "Towards deep learning models resistant to adversarial attacks". In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–28 (see page 124).

Magnusson, S., P. C. Weeraddana, M. G. Rabbat, and C. Fischione (2016). "On the convergence of alternating direction lagrangian methods for nonconvex structured optimization problems". In: *IEEE Transactions on Control of Network Systems* (see pages 136, 140, 142).

Mahmood, A. R., H. Van Hasselt, and R. S. Sutton (2014). "Weighted importance sampling for off-policy learning with linear function approximation". In: *Advances in Neural Information Processing Systems* 4.January, pp. 3014–3022 (see page 91).

Mahony, R., T. Hamel, and J.-M. Pflimlin (2008). "Nonlinear Complementary Filters on the Special Orthogonal Group". In: *IEEE Transactions on Automatic Control (TACON)* 53.5, pp. 1203–1218 (see page 165).

Mandlekar, A., Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese (2017). "Adversarially Robust Policy Learning: Active construction of physically-plausible perturbations". In: *IEEE International Conference on Intelligent Robots and Systems* 2017-Septe, pp. 3932–3939 (see pages 121, 122).

Mansard, N., A. Delprete, M. Geisert, S. Tonneau, and O. Stasse (2018). "Using a Memory of Motion to Efficiently Warm-Start a Nonlinear Predictive Controller".

In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2986–2993 (see page 101).

Marhefka, D. W. and D. E. Orin (1996). "Simulation of contact using a nonlinear damping model". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 2. April. IEEE, pp. 1662–1668 (see page 245).

Markley, F. L., Y. Cheng, J. L. Crassidis, and Y. Oshman (2007). "Averaging quaternions". In: *Advances in the Astronautical Sciences* 127 PART 2, pp. 1675–1682 (see page 178).

Martens, J. and R. Grosse (2015). "Optimizing neural networks with Kronecker-factored approximate curvature". In: *32nd International Conference on Machine Learning, ICML 2015* 3, pp. 2398–2407 (see pages 57, 206).

Masiero, S., A. Celia, G. Rosati, and M. Armani (2007). "Robotic-Assisted Rehabilitation of the Upper Limb After Acute Stroke". In: *Archives of Physical Medicine and Rehabilitation* 88.2, pp. 142–149 (see page 1).

Mastalli, C. et al. (2020). "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2536–2542 (see page 211).

Matheson, J. A. L. (1959). *Hyperstatic Structures: An Introduction to the Theory of Statically Indeterminate Structures*. Academic Press (see page 16).

Matthias, P. et al. (2018). "Parameter Space Noise for Exploration". In: *International Conference on Learning Representations (ICLR)* (see page 82).

Melo, D. C., M. R. Maximo, and A. M. Da Cunha (2020). "Push Recovery Strategies through Deep Reinforcement Learning". In: *2020 Latin American Robotics Symposium, 2020 Brazilian Symposium on Robotics and 2020 Workshop on Robotics in Education, LARS-SBR-WRE 2020*, pp. 1–6 (see pages 11, 198).

Mesesan, G., J. Englsberger, and C. Ott (2021). "Online DCM Trajectory Adaptation for Push and Stumble Recovery during Humanoid Locomotion". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 12780–12786 (see pages 41, 46, 111).

Micikevicius, P. et al. (2022). "FP8 Formats for Deep Learning". In: URL: http://arxiv.org/abs/2209.05433 (see page 64).

Miki, T., J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter (2022). "Learning robust perceptive locomotion for quadrupedal robots in the wild". In: *Science Robotics* 7.62 (see pages 69, 107, 114, 116, 263).

Miyato, T., T. Kataoka, M. Koyama, and Y. Yoshida (2018). "Spectral normalization for generative adversarial networks". In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (see page 121).

Mnih, V., A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning". In: *33rd International Conference on Machine Learning, ICML 2016* 4, pp. 2850–2869 (see pages 88, 283).

Mnih, V. et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533 (see page 275).

Mordatch, I. and E. Todorov (2014). "Combining the benefits of function approximation and trajectory optimization". In: *Robotics: Science and Systems*. Vol. 4 (see page 109).

Moreau, J. J. (1988). "Unilateral Contact and Dry Friction in Finite Freedom Dynamics". In: *Nonsmooth Mechanics and Applications*. Vienna: Springer Vienna, pp. 1–82 (see page 240).

Moro, F. L. and L. Sentis (2018). "Whole-Body Control of Humanoid Robots". In: *Humanoid Robotics: A Reference*. Ed. by A. Goswami and P. Vadakkepat. Dordrecht: Springer Netherlands, pp. 1–23 (see page 10).

Mysore, S., B. Mabsout, R. Mancuso, and K. Saenko (2021). "Regularizing Action Policies for Smooth Control with Reinforcement Learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1810–1816 (see pages 11, 119, 123, 124, 161, 194).

Nair, A., B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel (2018). "Overcoming Exploration in Reinforcement Learning with Demonstrations". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 6292–6299 (see page 83).

Nair, V. and G. E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Madison, WI, USA: Omnipress, pp. 807–814 (see page 60).

Narvekar, S., B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone (2020). "Curriculum learning for reinforcement learning domains: A framework and survey". In: *Journal of Machine Learning Research* 21, pp. 1–50 (see page 78).

Nota, C. and P. S. Thomas (2020). "Is the policy gradient a gradient?" In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 2020-May, pp. 939–947 (see pages 169, 288).

Odena, A., V. Dumoulin, and C. Olah (2016). "Deconvolution and Checkerboard Artifacts". In: *Distill* 1.10 (see page 146).

Ott, E. (2002). *Chaos in Dynamical Systems*. Cambridge University Press (see page 271).

Oudeyer, P.-Y., F. Kaplan, and V. V. Hafner (2007). "Intrinsic Motivation Systems for Autonomous Mental Development". In: *IEEE Transactions on Evolutionary Computation* 11.2, pp. 265–286 (see pages 84, 85).

Paine, N., J. S. Mehling, J. Holley, N. A. Radford, G. Johnson, C. L. Fok, and L. Sentis (2015). "Actuator control for the NASA-JSC valkyrie humanoid robot: A decoupled dynamics approach for torque control of series elastic robots". In: *Journal of Field Robotics* 32.3, pp. 378–396 (see page 30).

Pardo, F., A. Tavakoli, V. Levdik, and P. Kormushev (2018). "Time limits in reinforcement learning". In: *35th International Conference on Machine Learning, ICML 2018* 9, pp. 6443–6452 (see page 76).

Park, S., H. Ryu, S. Lee, S. Lee, and J. Lee (2019). "Learning predict-and-simulate policies from unorganized human motion data". In: *ACM Transactions on Graphics* 38.6 (see pages 118, 185).

Parmas, P. and M. Sugiyama (2021). "A unified view of likelihood ratio and reparameterization gradients". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 4078–4086 (see pages 92, 293).

Pathak, D., P. Agrawal, A. A. Efros, and T. Darrell (2017). "Curiosity-Driven Exploration by Self-Supervised Prediction". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. JMLR.org, pp. 2778–2787 (see pages 85, 113).

Pattanaik, A., Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary (2018). "Robust Deep Reinforcement Learning with adversarial attacks". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 3, pp. 2040–2042 (see page 122).

Pearson, K. (1905). "The Problem of the Random Walk". In: *Nature* 72.1865, pp. 294–294 (see page 70).

Peng, X. B., P. Abbeel, S. Levine, and M. van de Panne (2018). "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills". In: *ACM Transactions on Graphics (TOG)* 37, pp. 1–14 (see pages 113, 116).

Peng, X. B., E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine (2020). "Learning Agile Robotic Locomotion Skills by Imitating Animals". In: *Robotics: Science and Systems (RSS)*. URL: http://arxiv.org/abs/2004.00784 (see pages 107, 116, 119).

Pignat, E. and S. Calinon (2019). "Bayesian Gaussian Mixture Model for Robotic Policy Imitation". In: *IEEE Robotics and Automation Letters* 4.4, pp. 4452–4458 (see page 77).

Pistikopoulos, E. N., M. C. Georgiadis, and V. Dua (2007). *Multi-Parametric Programming*. Weinheim, Germany: Wiley (see pages 101, 133).

Postol, N., J. Grissell, C. McHugh, A. Bivard, N. J. Spratt, and J. Marquez (2021). "Effects of therapy with a free-standing robotic exoskeleton on motor function and other health indicators in people with severe mobility impairment due to chronic stroke: A quasi-controlled study". In: *Journal of Rehabilitation and Assistive Technologies Engineering* 8, p. 13 (see pages 1, 2).

Pratt, G. A. and M. M. Williamson (1995). "Series elastic actuators". In: *IEEE International Conference on Intelligent Robots and Systems* 1, pp. 399–406 (see page 5).

Pratt, J., J. Carff, S. Drakunov, and A. Goswami (2006). "Capture point: A step toward humanoid push recovery". In: *Proceedings of the 2006 6th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS*. IEEE, pp. 200–207 (see page 39).

Quarteroni, A., R. Sacco, and F. Saleri (2007). "Iterative Methods for Solving Linear Systems". In: *Numerical Mathematics*. New York, NY: Springer New York, pp. 123–181 (see page 248).

Racanière, S. et al. (2017). "Imagination-augmented agents for deep reinforcement learning". In: *Advances in Neural Information Processing Systems*. Vol. 2017-Decem. Nips, pp. 5691–5702 (see page 86).

Raffin, A., J. Kober, and F. Stulp (2022). "Smooth Exploration for Robotic Reinforcement Learning". In: *Proceedings of the 5th Conference on Robot Learning*. Ed. by A. Faust, D. Hsu, and G. Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, pp. 1634–1644 (see page 161).

Raissi, M., P. Perdikaris, and G. Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378.October, pp. 686–707 (see page 12).

Real, E., S. Moore, A. Selle, S. Saxena, Y. Leon, S. Jie, T. Quoc, and V. L. Alexey (2017). "Large-Scale Evolution". In: *Pmlr* (see page 61).

Rifai, S., P. Vincent, X. Muller, X. Glorot, and Y. Bengio (2011). "Contractive auto-encoders: explicit invariance during feature extraction". In: *Proceedings of The 28th International Conference on Machine Learning (ICML-11)* (see page 262).

Rodriguez, D. and S. Behnke (2021). "DeepWalk: Omnidirectional Bipedal Gait by Deep Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3033–3039 (see page 114).

Ross, S., G. Gordon, and D. Bagnell (2011). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson, and M. Dudík. Vol. 15. Proceedings of Machine Learning

Research. Fort Lauderdale, FL, USA: PMLR, pp. 627–635 (see pages 103, 104, 108).

Ross, S. and J. A. Bagnell (2010). "Efficient reductions for imitation learning". In: *Journal of Machine Learning Research*. Vol. 9, pp. 661–668 (see pages 69, 103).

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. URL: http://arxiv.org/abs/1609.04747 (see page 57).

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536 (see page 56).

Rummery, G. A. and M. Niranjan (1994). *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. September. Cambridge, England: Cambridge University Engineering Department (see pages 87, 274).

Ruppert, D. and M. P. Wand (1994). "Multivariate Locally Weighted Least Squares Regression". In: *The Annals of Statistics* 22.3 (see page 50).

Rusu, A. A. et al. (2016). "Policy distillation". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–13 (see page 69).

Salimans, T. and R. Chen (2018). *Learning Montezuma's Revenge from a Single Demonstration*. URL: http://arxiv.org/abs/1812.03381 (see page 83).

Sardain, P. and G. Bessonnet (2004). "Forces acting on a biped robot. Center of pressure - Zero moment point". In: *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*. 34.5, pp. 630–637 (see pages 35, 36, 38).

Schmidhuber, J. (1991). "Curious model-building control systems". In: *1991 IEEE International Joint Conference on Neural Networks*. IEEE, pp. 1458–1463 (see page 83).

Schrittwieser, J. et al. (2020). "Mastering Atari, Go, chess and shogi by planning with a learned model". In: *Nature* 588.7839, pp. 604–609 (see page 86).

Schulman, J., S. Levine, P. Abbeel, M. Jordan, and P. Moritz (2015). "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1889–1897 (see pages 88, 93–95).

Schulman, J., P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel (2016). "High-dimensional continuous control using generalized advantage estimation". In: *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–14 (see pages 94, 290).

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). *Proximal Policy Optimization Algorithms*. URL: http://arxiv.org/abs/1707.06347 (see pages 88, 97).

Sciavicco, L., B. Siciliano, and L. Villani (1995). "Lagrange and Newton-Euler dynamic modeling of a gear-driven robot manipulator with inclusion of motor inertia effects". In: *Advanced Robotics* 10.3, pp. 317–334 (see page 255).

Seo, Y., L. Chen, J. Shin, H. Lee, P. Abbeel, and K. Lee (2021). "State Entropy Maximization with Random Encoders for Efficient Exploration". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 9443–9454 (see page 84).

Seok, S., A. Wang, M. Y. Chuah, D. J. Hyun, J. Lee, D. M. Otten, J. H. Lang, and S. Kim (2015). "Design Principles for Energy-Efficient Legged Locomotion and Implementation on the MIT Cheetah Robot". In: *IEEE/ASME Transactions on Mechatronics* 20.3, pp. 1117–1129 (see page 5).

Servin, M., D. Wang, C. Lacoursière, and K. Bodin (2014). "Examining the smooth and nonsmooth discrete element approaches to granular matter". In: *International Journal for Numerical Methods in Engineering* 97.12, pp. 878–902 (see page 247).

Shen, Q., Y. Li, H. Jiang, Z. Wang, and T. Zhao (2020). "Deep Reinforcement Learning with Robust and Smooth Policy". In: *International Conference on Machine Learning (ICML)*. Vol. 119. Proceedings of Machine Learning Research, pp. 8707–8718 (see pages 124, 161, 194).

Shi, D., W. W. Zhang, W. W. Zhang, and X. Ding (2019). "A Review on Lower Limb Rehabilitation Exoskeleton Robots". In: *Chinese Journal of Mechanical Engineering* 32.1 (see pages 1, 2).

Siciliano, B. and O. Khatib (2008). *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer (see pages 21, 42, 255).

Siekmann, J., Y. Godse, A. Fern, and J. Hurst (2021a). "Sim-to-Real Learning of All Common Bipedal Gaits via Periodic Reward Composition". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2021-May, pp. 9943–9949 (see pages 163, 182, 187).

Siekmann, J., K. Green, J. Warila, A. Fern, and J. W. Hurst (2021b). "Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning". In: *Robotics: Science and Systems (RSS)* (see pages 107, 114, 163).

Silcowitz, M., S. Niebe, and K. Erleben (2010). "Projected Gauss-Seidel subspace minimization method for interactive rigid body dynamics: Improving Animation quality using a Projected Gauss-Seidel subspace minimization method". In: *GRAPP 2010 - Proceedings of the International Conference on Computer Graphics Theory and Applications*. Vol. 36. 4. SciTePress - Science, pp. 38–45 (see page 248).

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller (2014). "Deterministic policy gradient algorithms". In: *31st International Conference on Machine Learning, ICML 2014*. Vol. 1, pp. 605–619 (see pages 91, 285, 286).

Spong, M. W. (1987). "Modeling and control of elastic joint robots". In: *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME* 109.4, pp. 310–319 (see page 255).

Stadie, B. C., S. Levine, and P. Abbeel (2015). "Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models". In: *NIPS Workshop on Deep Reinforcement Learning*, pp. 1–11 (see page 85).

Stanley, K. O. and R. Miikkulainen (2002). "Evolving neural networks through augmenting topologies". In: *Evolutionary Computation* 10.2, pp. 99–127 (see page 61).

Stasse, O. et al. (2017). "TALOS: A new humanoid research platform targeted for industrial applications". In: *IEEE-RAS International Conference on Humanoid Robots*. IEEE, pp. 689–695 (see page 30).

Stephens, B. J. and C. G. Atkeson (2010). "Dynamic Balance Force Control for compliant humanoid robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1248–1255 (see page 111).

Stewart, D. E. and J. C. Trinkle (1996). "An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction". In: *International Journal for Numerical Methods in Engineering* 39.15, pp. 2673–2691 (see pages 242, 243, 246).

Still, G. (2018). "Lectures on Parametric Optimization: An Introduction". In: *Optimization Online* (see page 140).

Strassen, V. (1969). "Gaussian elimination is not optimal". In: *Numerische Mathematik* 13.4, pp. 354–356 (see page 65).

Stroke Foundation (2020). "The economic impact of stroke in Australia: Estimating the magnitude and impacts of stroke in Australia". In: *Melbourne: National Stroke Foundation* (see page 1).

Sutton, R. S., D. McAllester, S. Singh, and Y. Mansour (1999). "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press (see pages 277, 283).

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT press (see pages 65, 72, 81, 87, 273, 275).

Tachibana, H., K. Uenoyama, and S. Aihara (2018). "Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention". In: *2018 43th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 4784–4788 (see page 146).

Tan, M. and Q. V. Le (2019). "EfficientNet: Rethinking model scaling for convolutional neural networks". In: *36th International Conference on Machine Learning, ICML 2019* 2019-June, pp. 10691–10700 (see page 62).

Tancik, M. et al. (2020). "Fourier features let networks learn high frequency functions in low dimensional domains". In: *Advances in Neural Information Processing Systems* 2020-Decem, pp. 1–24 (see page 266).

Tang, J. and P. Abbeel (2010). "On a Connection between Importance Sampling and the Likelihood Ratio Policy Gradient". In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc. (see pages 92, 292).

Tanielian, U., M. Sangnier, and G. Biau (2021). "Approximating Lipschitz continuous functions with GroupSort neural networks". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by A. Banerjee and K. Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 442–450 (see page 60).

Tassa, Y., T. Erez, and E. Todorov (2012). "Synthesis and stabilization of complex behaviors through online trajectory optimization". In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 4906–4913 (see page 99).

Tassa, Y., N. Mansard, and E. Todorov (2014). "Control-limited differential dynamic programming". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 53. 2. IEEE, pp. 1168–1175 (see page 222).

Tatla, S. K., K. Sauve, N. Virji-Babul, L. Holsti, C. Butler, and H. F. Van Der Loos (2013). "Evidence for outcomes of motivational rehabilitation interventions for children and adolescents with cerebral palsy: An american academy for cerebral palsy and developmental medicine systematic review". In: *Developmental Medicine and Child Neurology* 55.7, pp. 593–601 (see page 2).

Tesau, C. and G. Tesau (1995). "Temporal Difference Learning and TD-Gammon". In: *Communications of the ACM* 38.3, pp. 58–68 (see page 66).

Thomas, P. (2014). "Bias in Natural Actor-Critic Algorithms". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by E. P. Xing and T. Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, pp. 441–448 (see pages 170, 288).

Tieleman, T. and G. Hinton (2012). "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2, pp. 26–31 (see page 58).

Tikhonov, A. N., A. V. Goncharsky, V. V. Stepanov, and A. G. Yagola (1995). *Numerical Methods for the Solution of Ill-Posed Problems*. Dordrecht: Springer Netherlands (see page 54).

Tobin, J., R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel (2017). "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IEEE International Conference on Intelligent Robots and Systems* 2017-Septe, pp. 23–30 (see page 106).

Todorov, E. (2010). "Implicit nonlinear complementarity: A new approach to contact dynamics". In: *Proceedings - IEEE International Conference on Robotics and Automation* 5, pp. 2322–2329 (see pages 82, 245).

– (2014). "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 6054–6061 (see page 245).

Todorov, E., T. Erez, and Y. Tassa (2012). "MuJoCo: A physics engine for model-based control". In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 5026–5033 (see pages 12, 106, 242, 246).

Tokic, M. (2010). "Adaptive ϵ-greedy exploration in reinforcement learning based on value differences". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6359 LNAI. Springer, pp. 203–210 (see page 80).

Tunyasuvunakool, S. et al. (2020). "dm-control : Software and tasks for continuous control". In: *Software Impacts* 6.July, p. 100022 (see page 82).

Uchendu, I. et al. (2022). "Jump-Start Reinforcement Learning". In: URL: http://arxiv.org/abs/2204.02372 (see page 115).

Uhlenbeck, G. E. and L. S. Ornstein (1930). "On the Theory of the Brownian Motion". In: *Physical Review* 36.5, pp. 823–841 (see page 82).

Van Havermaet, S., Y. Khaluf, and P. Simoens (2021). "No more hand-tuning rewards: Masked constrained policy optimization for safe reinforcement learning". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS* 3, pp. 1332–1340 (see page 117).

Van Otterlo, M. and M. Wiering (2012). "Reinforcement learning and markov decision processes". In: *Adaptation, Learning, and Optimization*. Vol. 12. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 3–42 (see page 71).

Vapnik, V. and A. Vashist (2009). "A new learning paradigm: Learning using privileged information". In: *Neural Networks* 22.5-6, pp. 544–557 (see pages 69, 107).

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems* 2017-Decem.Nips, pp. 5999–6009 (see page 267).

Vecerik, M. et al. (2017). *Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards*. URL: http://arxiv.org/abs/1707.08817 (see page 83).

Veerbeek, J. M., A. C. Langbroek-Amersfoort, E. E. Van Wegen, C. G. Meskers, and G. Kwakkel (2017). "Effects of Robot-Assisted Therapy for the Upper Limb after Stroke". In: *Neurorehabilitation and Neural Repair* 31.2, pp. 107–121 (see page 1).

Vigne, M. (2021). "Estimation and control of the deformations of an exoskeleton using inertial sensors". PhD thesis. Université Paris sciences et lettres (see page 7).

Vigne, M., A. E. Khoury, F. Di Meglio, and N. Petit (2020a). "Improving low-level control of the exoskeleton atalante in single support by compensating joint flexibility". In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 3437–3443 (see pages 7, 112, 235).

Vigne, M., A. E. Khoury, M. Masselin, F. Di Meglio, and N. Petit (2019). "Estimation of Multiple Flexibilities of an Articulated System Using Inertial Measurements". In: *Proceedings of the IEEE Conference on Decision and Control* 2018-Decem.Cdc, pp. 6779–6785 (see pages 235, 236).

Vigne, M., A. E. Khoury, F. D. Meglio, and N. Petit (2020b). "State Estimation for a Legged Robot with Multiple Flexibilities Using IMUs: A Kinematic Approach". In: *IEEE Robotics and Automation Letters* 5.1, pp. 195–202 (see pages 6, 37, 236).

Vigne, M., A. E. Khoury, M. Pétriaux, F. D. Meglio, and N. Petit (2022). "MOVIE: A Velocity-Aided IMU Attitude Estimator for Observing and Controlling Multiple Deformations on Legged Robots". In: *IEEE Robotics and Automation Letters* 7.2, pp. 3969–3976 (see page 165).

Villa, N. A. and P. B. Wieber (2017). "Model predictive control of biped walking with bounded uncertainties". In: *IEEE-RAS International Conference on Humanoid Robots*, pp. 836–841 (see page 112).

Vincent, P., H. Larochelle, Y. Bengio, and P. A. Manzagol (2008). "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*. Vol. 311. New York, New York, USA: ACM Press, pp. 1096–1103 (see page 262).

Vukobratović, M. and J. Stepanenko (1972). "On the stability of anthropomorphic systems". In: *Mathematical Biosciences* 15.1-2, pp. 1–37 (see page 34).

Wandercraft (2021). *About us*. URL: https://www.wandercraft.eu/about-us/ (visited on 11/15/2021) (see pages 3, 4, 8, 12, 13).

Watkins, C. J. C. H. (1989). "Learning from delayed rewards". PhD thesis. King's College, Cambridge United Kingdom (see pages 80, 87).

Weaver, L. and N. Tao (2001). "The Optimal Reward Baseline for Gradient-Based Reinforcement Learning". In: *Proceedings of the 17th conference on Uncertainty in artificial intelligence (UAI)*. Ed. by J. S. Breese and D. Koller. Morgan Kaufmann, pp. 538–545 (see page 289).

Westervelt, E. R., J. W. Grizzle, and D. E. Koditschek (2003). "Hybrid zero dynamics of planar biped walkers". In: *IEEE Transactions on Automatic Control* 48.1, pp. 42–56 (see page 211).

Wieber, P. B. and C. Chevallereau (2006). "Online adaptation of reference trajectories for the control of walking systems". In: *Robotics and Autonomous Systems* 54.7, pp. 559–566 (see page 108).

Wieber, P.-b. (2006). "Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations". In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pp. 137–142 (see pages 47, 99, 111).

Williams, R. J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3, pp. 229–256 (see pages 87, 282).

Wittmann, R., A. C. Hildebrandt, D. Wahrmann, D. Rixen, and T. Buschmann (2015). "Real-time nonlinear model predictive footstep optimization for biped robots". In: *IEEE-RAS International Conference on Humanoid Robots* 2015-Decem, pp. 711–717 (see page 111).

Wojna, Z. et al. (2019). "The Devil is in the Decoder: Classification, Regression and GANs". In: *International Journal of Computer Vision (IJCV)* (see page 146).

Won, J., D. Gopinath, and J. Hodgins (2020). "A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters - Facebook Research". In: 39.4 (see pages 160, 168, 182, 205).

Wu, F., L. Li, Z. Huang, Y. Vorobeychik, D. Zhao, and B. Li (2022). "CROP: Certifying Robust Policies for Reinforcement Learning through Functional Smoothing". In: *International Conference on Learning Representations* (see pages 108, 120).

Xie, Z., P. Clary, J. Dao, P. Morais, J. Hurst, and M. van de Panne (2019). "Learning Locomotion Skills for Cassie: Iterative Design and Sim-to-Real". In: *3rd Conference on Robot Learning (CoRL)*. Vol. 100. Proceedings of Machine Learning Research. PMLR, pp. 317–329 (see pages 89, 114).

Yang, C., K. Yuan, S. Heng, T. Komura, and Z. Li (2020). "Learning Natural Locomotion Behaviors for Humanoid Robots Using Human Bias". In: *IEEE Robotics and Automation Letters (RA-L)* 5.2, pp. 2610–2617 (see page 115).

Yang, C., K. Yuan, W. Merkt, T. Komura, S. Vijayakumar, and Z. Li (2018). "Learning Whole-Body Motor Skills for Humanoids". In: *IEEE/RAS International Conference on Humanoid Robotics (Humanoids)*, pp. 270–276 (see pages 115, 193, 200, 201).

Yu, T., G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma (2020). "MOPO: Model-based offline policy optimization". In: *Advances in Neural Infor-*

mation Processing Systems. Vol. 33. Curran Associates, Inc., pp. 14129–14142 (see page 69).

Yu, W., J. Tan, C. K. Liu, and G. Turk (2017). "Preparing for the unknown: Learning a universal policy with online system identification". In: *Robotics: Science and Systems* 13 (see pages 106, 206).

Yuan, K., C. McGreavy, C. Yang, W. Wolfslag, and Z. Li (2020). "Decoding Motor Skills of Artificial Intelligence and Human Policies: A Study on Humanoid and Human Balance Control". In: *IEEE Robotics and Automation Magazine (RA-M)* 27.2, pp. 87–101 (see page 110).

Zhang, H., H. Chen, C. Xiao, B. Li, M. Liu, D. Boning, and C. J. Hsieh (2020). "Robust deep reinforcement learning against adversarial perturbations on state observations". In: *Advances in Neural Information Processing Systems* 2020-Decem.NeurIPS, pp. 1–38 (see pages 123, 124, 161).

Zhang, T. T. C. K., S. Tu, N. M. Boffi, J.-J. E. Slotine, and N. Matni (2022). "Adversarially Robust Stability Certificates can be Sample-Efficient". In: *4th Annual Learning for Dynamics & Control Conference* (see page 120).

Zhao, W., J. P. Queralta, and T. Westerlund (2020). "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey". In: *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020*, pp. 737–744 (see pages 69, 106).

Zhou, D. X. (2020). "Universality of deep convolutional neural networks". In: *Applied and Computational Harmonic Analysis* 48.2, pp. 787–794 (see page 59).

Zhou, X. Y. (1990). "Maximum principle, dynamic programming, and their connection in deterministic control". In: *Journal of Optimization Theory and Applications* 65.2, pp. 363–373 (see page 221).

Zhu, X. (2021). *Awesome Deep Reinforcement Learning*. URL: https://github.com/tigerneil/awesome-deep-rl (see page 89).

Ziegler, J. G. and N. B. Nichols (1993). "Optimum settings for automatic controllers". In: *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME* 115.2B, pp. 220–222 (see page 43).

RÉSUMÉ

Cette thèse contribue à améliorer la planification de trajectoires et le contrôle des robots bipèdes. Le but concret est de permettre aux paraplégiques de remarcher de façon autonome avec l'exosquelette de membres inférieurs Atalante. Notre approche combine les méthodes issues l'apprentissage automatique et de la robotique traditionnelle. Nous mettons d'abord de côté le contrôle. L'objectif est de permettre la planification de trajectoires en ligne tout en garantissant un fonctionnement sûr. C'est une étape cruciale vers la navigation en milieu incertain et la prise en compte des préférences utilisateur. Nous entraînons ensuite un contrôleur par renforcement afin de généraliser un ensemble prédéfini de mouvements élémentaires. Nous ne cherchons pas la meilleure performance, mais plutôt la transférabilité et la sécurité. Nous proposons une formulation qui apparente à l'apprentissage par imitation mais laisse suffisamment de marge de manœuvre pour affronter des événements inattendus.

MOTS CLÉS

Robots à jambes, exosquelettes de membres inférieurs, locomotion, apprentissage automatique, apprentissage par renforcement, optimisation, contrôle, planification de trajectoires

ABSTRACT

This thesis contributes to improving the motion planning and control of biped robots. Our concrete goal is restoring natural locomotion for paraplegic people in their daily lives using the medical lower-limb exoskeleton Atalante, notably walking safely and autonomously without crutches. The core idea is to combine traditional robotics and state-of-the-art machine learning. We put aside closed-loop control to focus on planning at first. The objective is to enable online trajectory planning while ensuring safe operation. This is a milestone toward realizing versatile navigation in an unstructured environment and accommodating the user preferences. Second, we train a policy using reinforcement learning to generalize a predefined set of primitive motions. We do not seek the best possible performance, but rather transferability and safety. We propose a formulation closely related to imitation learning while giving enough leeway to deal with unexpected events.

KEYWORDS

Legged robots, lower-limb exoskeletons, locomotion, machine learning, reinforcement learning, optimization, control, trajectory planning