

SAPIENZA
UNIVERSITÀ DI ROMA

Learning to Bridge Humanoid Robot Skills

Faculty of Information Engineering, Computer Science, and Statistics
Master's Degree in Artificial Intelligence and Robotics

Daniel Gigliotti
ID number 2014893

Advisor
Prof. Luca Iocchi

Academic Year 2025/2026

Thesis defended on 29 October 2026
in front of a Board of Examiners composed by:

Prof. Alessandro De Luca (chairman)
Prof. Irene Amerini
Prof. Silvia Bonomi
Prof. Luca Iocchi
Prof. Francesco Leotta
Prof. Giuseppe Santucci
Prof. Fabrizio Silvestri
Prof. Marilena Venditelli

Learning to Bridge Humanoid Robot Skills
Master thesis. Sapienza University of Rome

© 2026 Daniel Gigliotti. All rights reserved

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: danielgigliotti99.dg@gmail.com

Abstract

Policies learned for individual humanoid behaviors can now produce agile whole-body motions, but useful tasks often require several behaviors in sequence. Transitioning between independently trained skills, the source skill may leave the robot in a state from which the incoming skill cannot continue. This thesis investigates whether a single intermediate policy, called a *bridge*, can connect such skills while leaving them frozen. Given the robot's recent motion, a short reference sequence from the incoming skill, and a time limit, the bridge must move the robot from the state left by one skill to a state from which the next can proceed.

We develop three bridge architectures in sequence, with each addressing limitations of the previous one: a goal-conditioned reinforcement-learning policy, a conditional variational autoencoder distilled from a tracking teacher, and a diffusion planner paired with a universal tracker. The approaches use human-motion data retargeted to the robot. Quantitative evaluation focuses on the diffusion-based approach and three transitions between frozen skills of the Unitree G1, comparing it with direct switching and pair-specific transition policies. We also demonstrate the approach in two long simulated tasks that chain many skills: soccer with the G1 and Sokoban with the Booster T1.

At the selected settings, the bridge succeeds in all 150 transition trials and keeps the incoming skills' performance near their nominal levels. It is not consistently better than the pair-specific policies, however, and its performance depends on the space and time available for the transition. The results show the potential of a shared bridge under the tested conditions, while leaving open how to choose the target state and transition parameters reliably, and how well the approach transfers to physical robots.

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Motivation . . . 2
1.3 The Problem . . . 3
1.4 Approach and Research Questions . . . 4
1.5 Scope . . . 5
1.6 Outline . . . 5
1.7 Thesis Outcomes . . . 5

2 Background **7**
2.1 Reinforcement Learning . . . 7
2.1.1 Markov Decision Processes . . . 7
2.1.2 Policy Optimization and Actor-Critic Methods . . . 8
2.1.3 Goal-Conditioned Policies . . . 10
2.2 Hierarchical Reinforcement Learning . . . 11
2.2.1 Options . . . 11
2.2.2 Local-Coordinate and Goal-Conditioned Options . . . 12
2.2.3 State Abstraction and ϕ-Relative Options . . . 13
2.2.4 Abstract-Action-Aware Options . . . 15
2.2.5 2-Abstract-Action-Aware Options . . . 16
2.3 Human Motion Data . . . 19
2.3.1 Motion Capture and Retargeting . . . 19
2.3.2 LAFAN1 . . . 20
2.3.3 AMASS and BABEL . . . 20
2.3.4 Robot Motion Datasets . . . 21
2.4 Motion Control Policies . . . 21
2.4.1 Reward-Based Policy Learning . . . 21
2.4.2 Motion Imitation and Trajectory Tracking . . . 23
2.4.3 Teacher-Student Distillation and Dataset Aggregation . . . 25
2.5 Latent Variable Models . . . 26
2.5.1 Autoencoders . . . 26
2.5.2 Variational Autoencoders . . . 26
2.5.3 Conditional Variational Autoencoders . . . 28
2.6 Diffusion Models . . . 29
2.6.1 Denoising Diffusion . . . 29
2.6.2 Conditioning and Guidance . . . 30

3 Related Work **31**
3.1 Sequential Skill Composition . . . 31
3.1.1 Composition by Construction . . . 31
3.1.2 Adapting the Skills . . . 32
3.1.3 Transition and Setup Policies . . . 33
3.2 Motion Generation . . . 33
3.2.1 Kinematic Motion In-Betweening . . . 33
3.2.2 Diffusion for Planning and Control . . . 34
3.2.3 Physics-Based Conditional Control . . . 34
3.3 Motion Tracking . . . 35
3.3.1 Tracking Single Motions . . . 35
3.3.2 Universal Motion Trackers . . . 35
3.4 Summary and Positioning . . . 36

4 Problem Formulation **39**
4.1 Skills . . . 39
4.2 The Bridge . . . 41
4.3 Windows . . . 43
4.4 Assumptions and Scope . . . 45
4.5 Intrinsic Difficulties . . . 45

5 Proposed Methods **47**
5.1 Target Selection . . . 47
5.2 Motion Corpus . . . 48
5.2.1 Selection . . . 49
5.2.2 Retargeting and Grounding . . . 51
5.2.3 Kinematic Filtering . . . 52
5.2.4 Motion Graphs . . . 53
5.2.5 Transition Windows . . . 54
5.2.6 Randomization and Augmentation . . . 55
5.2.7 Learning from a Corpus . . . 56
5.3 Goal-Conditioned Policy . . . 58
5.3.1 Architecture . . . 58
5.3.2 Training . . . 59
5.3.3 Limitations . . . 61
5.4 Conditional VAE Policy . . . 61
5.4.1 Teacher . . . 62
5.4.2 Student . . . 63
5.4.3 Training . . . 64
5.4.4 Limitations . . . 66
5.5 Diffusion Planner and Universal Tracker . . . 66
5.5.1 Planner . . . 66
5.5.2 Planner Training . . . 69
5.5.3 Universal Tracker . . . 70
5.5.4 Deployment . . . 72
5.5.5 Limitations . . . 72

6 Experiments **75**
6.1 Goals . . . 75
6.2 Experimental Setup . . . 75
6.2.1 Simulation Environment . . . 75
6.2.2 Robots . . . 76
6.2.3 Skill Pool . . . 76
6.2.4 Entry Sequences . . . 79
6.3 Transitions . . . 83
6.3.1 Walking to kicking . . . 84
6.3.2 Walking to jumping . . . 84
6.3.3 Jumping to walking . . . 84

7 Results **87**
7.1 Qualitative Results . . . 87
7.1.1 Soccer Demo . . . 88
7.1.2 Sokoban Demo . . . 90
7.2 Baselines . . . 96
7.2.1 Transition Policy Architecture and Training . . . 96
7.2.2 How a Transition Policy is used . . . 96
7.2.3 Instant Switch (No-op) . . . 97
7.3 Metrics . . . 97
7.3.1 Measurement after the Handover . . . 98
7.3.2 Measures for the G1 Skill Pool . . . 98
7.3.3 Parameter Selection . . . 100
7.3.4 Normalization . . . 100
7.3.5 Action Discontinuity . . . 101
7.4 Results . . . 101
7.4.1 Nominal Performance . . . 101
7.4.2 Bridge Parameter Selection . . . 102
7.4.3 Baselines after the Handover . . . 104
7.4.4 Bridge after the Handover . . . 104
7.4.5 Comparison . . . 105
7.4.6 Training Cost . . . 106
7.4.7 Action Discontinuity . . . 107

8 Conclusions **109**
8.1 Summary . . . 109
8.2 Future Work . . . 110

Bibliography **111**

List of Figures

1.1 The soccer task 1

2.1 Ground and abstract MDP example 14
2.2 Frames of a LAFAN1 take 20
2.3 Frames of an AMASS take 21
2.4 BABEL annotation of an AMASS take 22

4.1 A switch with a bridge 44

5.1 Entry points of a kicking skill 48
5.2 BABEL frame labels on a timeline 50
5.3 Retargeting and grounding 52
5.4 A stitched trajectory 54
5.5 A transition window 55
5.6 Target displacement 57
5.7 Conditional VAE bridge architecture 62
5.8 Diffusion planner and universal tracker 67
5.9 A plan of the diffusion planner 69

6.1 Walking skill of the G1 77
6.2 Jumping skill of the G1 78
6.3 Kicking skill of the G1 79
6.4 Entry sequences of walking and kicking 81
6.5 Placement of an entry point 83
6.6 Switch from walking to kicking 84
6.7 Switch from walking to jumping 85
6.8 Switch from jumping to walking 86

7.1 Scene of the soccer demo 89
7.2 A run of the soccer demo 90
7.3 Switch from walking to jumping in the soccer demo 91
7.4 Switch from walking to kicking in the soccer demo 91
7.5 Puzzle of the Sokoban demo 93
7.6 Scene of the Sokoban demo 94
7.7 A run of the Sokoban demo 95

Chapter 1

Introduction

1.1 Context

Learning individual motor skills on legged and humanoid robots has become remarkably effective. Deep reinforcement learning in simulation, followed by transfer to hardware, now produces agile behaviors that were out of reach a few years ago, such as parkour over rough terrain [1], blind locomotion in natural environments [2] and dynamic striking in robot soccer [3]. In each of these works a policy is trained for one behavior and validated in the conditions of that behavior. A useful robot, however, has to carry out tasks that require several behaviors one after the other, and how to obtain a robot that combines many skills in a single task is still an open question.

A simple example shows what such a task looks like. A humanoid robot playing soccer walks toward the ball, but an obstacle blocks its path. To reach the ball, it jumps over it, walks on, and kicks the ball into the goal (Figure 1.1). Each of

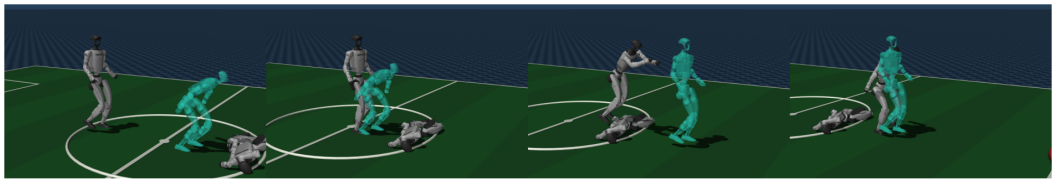


Figure 1.1. The soccer task, one of the demos of the thesis (Section 7.1.1). The robot walks toward a fallen robot that blocks its path, jumps over it, walks on and kicks the ball into the goal; each of these behaviors is realized by a separate policy, independently trained. Each panel shows the time and the policy in control. Between two skills, control passes through the bridge studied in this thesis (Section 1.4), and the translucent robot shows the state from which the next skill starts, which the bridge has to reach.

these behaviors requires a different motion and, if it is learned, a different objective: walking at a commanded velocity is naturally described by a reward, while a jump or a kick is easier to learn by reproducing a recorded human motion. The task breaks naturally into a sequence of shorter behaviors, each of which can be learned on its own. This scenario is one of the demos of the thesis (Section 7.1.1).

Several approaches address the problem of combining behaviors, with different trade-offs. Model-based control uses a model of the dynamics of the robot to compute its motion and also give formal guarantees under explicit assumptions [4, 5]. These methods, however, need accurate models, which become hard to provide as the behaviors become more varied, more dynamic and richer in contacts. End-to-end reinforcement learning instead trains a single policy for the whole task [2, 6]. When the task requires several behaviors, as the soccer task does, the reward of this single policy has to trade off many objectives at once, such as balancing, progress and safety, and when the policy fails at the end of a long task it is hard to tell which of its earlier actions caused the failure. A single policy also offers no separate parts that can be trained, tested or replaced on their own.

Modular skills offer an alternative in which the behaviors are kept separate. Each behavior is learned as its own policy, with its own objective and its own training data, and a task is carried out by combining several of them. What this decomposition leaves open is how the skills are combined. In concurrent composition several skills contribute to the command at the same time, as in skill blending for whole-body loco-manipulation [7], while in sequential composition control passes from one skill to the next. This thesis studies the sequential composition of whole-body skills of a humanoid robot.

1.2 Motivation

Keeping the skills separate has many advantages. Each skill can be obtained with the method that suits its behavior, for example a task reward for walking and a recorded motion for a jump or a kick (Section 2.4). Each skill can be trained and tested on its own, a failure can be traced to the skill in which it happened, and the set of skills available to the robot, its *pool*, can grow by adding new skills instead of training everything again.

Separate skills also separate two problems that a single policy has to solve together: deciding which behavior to perform, and performing it. The decision can be given to a high-level controller, which observes the task and the environment and decides which skill to run, with which parameters, and when to pass to the next one, while the skills take care of the motion. Such a controller can be of many kinds and can be chosen to be interpretable, with decisions that can be inspected and verified.

These advantages rest on an assumption: that control can actually pass from one skill to the next. Any controller that sequences skills faces the same limit: it can only request the switches that the skills happen to allow, and it has to know which ones they are. Making the switches themselves possible, without changing the skills, is one of the aspects investigated in this thesis.

1.3 The Problem

Sequential composition is not as simple as running one skill after another. A skill learned from experience works only from a limited set of states, those that resemble the states it met during training, and its behavior from any other state cannot be predicted. The options framework, the standard formalism for behaviors that extend over time [8], calls this set the *initiation set* of the skill, and makes precise when two skills can be executed one after the other: the state in which the first one terminates has to belong to the initiation set of the second (Section 2.2.1). Skills trained independently have no reason to satisfy this requirement: each of them is trained for its own behavior and knows nothing about the skill that follows it.

The requirement can be enforced in several ways, and each of them places a demand on the skills.

A common convention is to obtain every skill so that it starts from the robot standing still, and to bring the robot to rest at the end of every skill before the next one starts. The state at rest then must belong to the initiation set of every skill, and the requirement holds by construction. The convention, however, has to be respected by every skill of the pool, which requires care in how each of them is obtained: a skill whose behavior naturally starts from another state, such as a jump that starts from the crouch that prepares the take-off, has to be modified or trained again to start from rest, and a skill obtained elsewhere may not respect the convention at all. Stopping also takes time, breaks the continuity of the motion between two skills, and dissipates the momentum the robot has built up, which the next behavior could otherwise use.

The skills can also be adapted to each other. Adversarial skill chaining fine-tunes each skill so that its terminal states fall in the initiation set of the next one [9], later works refine the skills in both directions [10] or guide them toward states from which the next skill has a high value [11], and Robust Policy Gating trains each expert while it is randomly interrupted and replaced by another, so that it learns to start from the states that the others leave [12]. The skills then fit together, but each of them depends on its neighbors: it can no longer be trained or tested on its own, and adding a skill to the pool requires training the others again.

Finally, the skills can be left as they are, and a learned policy can be placed between them, which brings the robot from the state left by one skill to a state accepted by the next. Transition policies [13, 14] and setup policies [15] follow this principle, and setup policies, for example, raise the success rate of a simulated biped well above that of switching directly between its locomotion policies. Each of these policies, however, is trained for one incoming skill or for one pair of skills, on executions of the skills it connects, so their number grows with the pool, up to one for each ordered pair of skills, and every new skill requires training new ones.

In all these approaches the compatibility between two skills is obtained through work that is specific to those skills: training all of them under a shared convention, training them for each other, or training a policy for each of them or for each pair.

1.4 Approach and Research Questions

This thesis investigates whether this dependence on specific skills can be removed: whether a single intermediate policy, independent of the skills it connects, can make the switch between any two of them. This policy is called the *bridge*. When a skill ends, by its own termination or by an interruption, the bridge takes control of the robot and brings it to a target state from which the next skill can continue, and it hands control to the next skill once the target is reached, at the latest when a deadline expires. The bridge receives a description of the switch: the recent motion of the robot, which tells how the robot is moving when the bridge takes over, a short piece of the motion that the next skill will likely perform from the target state, which tells how the robot has to be moving when the bridge hands over, and the time available to realize a motion that connects the two. The skills are frozen: neither of them is trained again, and neither needs to know that a bridge exists. The target state can lie anywhere in the motion of the next skill, including states in which the robot is moving, so the switch does not depend on the robot coming to rest.

Since everything the bridge needs is contained in the description of the switch, it does not have to be trained on the skills. It can instead be learned from a corpus of human motion retargeted to the robot, in which every piece of motion between two states is an example of how the robot can go from one to the other. If this works, a single bridge would take the place of the transition policies of each skill or pair, and the pool could change without training the bridge again, as long as the states of the skills resemble the motion of the corpus.

Whether such a bridge can be learned is not obvious, and finding out is the purpose of this thesis. The advantages of a single bridge are not guaranteed either. A bridge shared by every pair of skills has to cover all of them, and it may connect each pair worse than a policy dedicated to it; it is worth training only if it costs less than training a policy for each pair; and a new skill can be connected without training only if the corpus contains motion similar to its states.

The central question of the thesis is therefore the following: can a single policy, learned without access to the skills it connects, bring a humanoid robot within a given time from the state in which one frozen skill ends to a state from which another frozen skill can continue? The thesis approaches it through more specific questions.

1. How can a switch between independently trained skills be stated precisely, and how does a bridge relate to the formalisms that already describe sequential composition?

2. Which learning architectures can address the problem, and what limits each of them?

3. How should a switch be evaluated, and how does a bridge compare with switching directly and with transition policies trained for each skill or skill pair?

1.5 Scope

All the experiments are carried out in simulation, with the MuJoCo Warp physics engine through the mjlab framework [16], and transfer to real hardware is not addressed. The thesis studies how the bridge reaches a given target state, and replaces the choice of the best target with a proxy (Section 5.1). Which skills to run, and when to switch between them, are decided by a high-level controller, which is also outside the scope of the thesis.

1.6 Outline

Chapter 2 introduces the background the thesis builds on: reinforcement learning and goal-conditioned policies; hierarchical reinforcement learning, from options to the abstract-action-aware options that provide the vocabulary of the problem; the human motion datasets and the retargeting tool from which the training data is built; the policies that control the robot, from reward-based learning to motion tracking and distillation; and the two families of generative models used by the architectures, latent variable models and diffusion models. Chapter 3 reviews the work on skill composition, motion generation and motion tracking that the thesis builds on. Chapter 4 formulates the bridging problem and fixes the terminology used in the rest of the thesis. Chapter 5 describes the proxy used for the selection of the target, the motion corpus and the pipeline that builds it, and the three architectures. Chapter 6 describes the experimental setup, the skills and the isolated transitions, and Chapter 7 presents the demos, introduces the baselines and the metrics, and reports and discusses the results. Chapter 8 summarizes the findings and the directions for future work.

1.7 Thesis Outcomes

The content of this thesis has been exploited in scientific and dissemination publications.

- *HumanoidSoccerMaze: a novel benchmark and a hierarchical solution approach*, article submitted to the RoboCup Symposium.
- *Skill Composition for Legged Robot Reinforcement Learning*, article for the Workshop on Artificial Intelligence and Robotics (AIRO 2026) at AIxIA 2026, with project page at https://ggldnl.github.io/Skill-Composition/.
- A dissemination article on skill transitions in legged robots, https://theprimary.com/science/2026-10-02/legged-robots-skill-transitions.

Videos of the transitions and demos, together with additional material, are available at https://ggldnl.github.io/Skill-Bridging.

Chapter 2

Background

This chapter introduces the background on which the rest of the thesis builds. It begins with reinforcement learning and hierarchical reinforcement learning, from Markov decision processes and options up to the abstract-action-aware options that the rest of the thesis uses to describe the composition of skills. It then presents the human motion data from which the training data of this thesis is obtained, together with the tool that retargets it to the robot, and the policies that control it, from reward-based learning to motion tracking and distillation. The chapter closes with two families of generative models: latent variable models and diffusion models.

2.1 Reinforcement Learning

2.1.1 Markov Decision Processes

Reinforcement learning concerns learning to act from interaction rather than from labeled examples: an agent observes a state, selects an action, receives a scalar reward, and transitions to a new state, inferring from this experience which actions are profitable in the long run. This interaction loop is formalized as a Markov Decision Process (MDP) [17], the standard model of sequential decision making under uncertainty: a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \rho_0 \rangle$. Here $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a})$ the transition kernel giving the distribution over the next state after taking action $\mathbf{a}$ in state $\mathbf{s}$, $R(\mathbf{s}, \mathbf{a})$ the reward function, $\gamma \in [0, 1)$ a discount factor and ρ_0 the distribution of initial states. For a robot both $\mathcal{S}$ and $\mathcal{A}$ are continuous: $\mathcal{S} \subseteq \mathbb{R}^{n_s}$ could collect root pose and twist together with the joint positions and velocities, and $\mathcal{A} \subseteq \mathbb{R}^{n_a}$ holds one command per actuated joint.

The agent's behavior is specified by a policy $\pi_\theta(\mathbf{a} \mid \mathbf{s})$, a distribution over the actions available in each state, parameterized by θ. Executing the policy generates a trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots)$: the initial state is drawn from ρ_0, each action from π_θ, and each subsequent state from the transition dynamics P. The return of a trajectory is defined as the sum of the rewards collected along it, discounted by γ. Discounting both keeps the return finite over an unbounded horizon and assigns greater weight to rewards received sooner. Learning amounts to finding the parameters that maximize the expected return over trajectories induced by the policy,

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}_t, \mathbf{a}_t) \right]. \tag{2.1}$$

Optimizing Equation (2.1) directly is difficult because the return evaluates an entire trajectory, whereas the policy must be improved through individual action choices. Value functions provide the link between these two levels. The state value $V^\pi(\mathbf{s})$ is the expected return from $\mathbf{s}$ onward when π is followed, while the action value $Q^\pi(\mathbf{s}, \mathbf{a})$ is the expected return when $\mathbf{a}$ is taken in $\mathbf{s}$ and π is followed thereafter. Their difference,

$$A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s}) \tag{2.2}$$

is the advantage. It tells whether taking a particular action in a particular state is better or worse than what the policy would normally do in that state.

2.1.2 Policy Optimization and Actor-Critic Methods

Finding an optimal policy can be approached in two ways, which differ in what is learned and in how an action is eventually chosen. *Value-based* methods learn only a value function, usually Q^π, and never represent the policy explicitly: behavior is read off the learned values by acting greedily, taking in each state the action with the largest estimate [18]. *Policy-based* methods directly optimize the policy parameters so that the distribution $\pi_\theta(\mathbf{a} \mid \mathbf{s})$ it produces leads to the highest expected return, which is the objective $J(\theta)$ of Equation (2.1). The distinction matters a lot. Acting greedily requires maximizing Q^π over the action space at every step, which is a lookup when the actions are few and discrete but is itself an optimization problem when $\mathcal{A} \subseteq \mathbb{R}^{n_a}$ is continuous. A parameterized policy avoids this by mapping the state directly to a distribution over actions, which can then be sampled. Sampling also supplies exploration during training, while the mean of the distribution is the action the policy currently considers best. Exploration is therefore intrinsic to a policy-based method, rather than injected from outside as it has to be when behavior is read off a value function.

Policy-based methods are in turn distinguished by how $J(\theta)$ is optimized. *Derivative-free policy search* treats the return as a black box and explores the parameter space directly, by hill climbing, simulated annealing or evolution strategies [19]; the objective need not be differentiable, but the search makes no use of how the individual actions contributed to it. *Policy-gradient methods* instead estimate $\nabla_\theta J(\theta)$ from sampled trajectories and ascend it. Following the gradient makes action preferences change smoothly and converge, under standard conditions, to a local optimum of J. But there are also downsides: nothing guarantees that this optimum is global; the gradient is estimated from stochastic returns, so it can have high variance and require many samples per useful update; and those samples are on-policy, collected under the current parameters and no longer valid once an update has changed them.

The policy gradient can be written as an expectation of the score of the policy

weighted by the advantage of the action taken [20]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \, \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \, A^\pi(\mathbf{s}_t, \mathbf{a}_t)\right]. \tag{2.3}$$

Raising the probability of an action in proportion to its advantage is what the expression prescribes: the score $\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$ is the direction in parameter space that makes the action taken more likely, and the advantage decides the sign and the size of the step along it. Implementations commonly drop the factor γ^t and treat every step of a batch alike, keeping γ only inside the advantage.

The advantage in Equation (2.3) is not known and has to be estimated from the collected experience. The simplest estimator replaces it with the return accumulated from t onward: the rewards collected before t cannot be affected by $\mathbf{a}_t$ and can be left out, and the resulting gradient estimate is still unbiased [21]. Its variance, however, grows with the horizon, since one sampled return is charged with explaining every action that contributed to it. Subtracting from that return a quantity that depends on the state but not on the action leaves the estimate unbiased and can reduce its variance considerably; the usual choice is the state value V^π, which turns the return into an estimate of the advantage of Equation (2.2). Actor-critic methods make this practical by learning a value function alongside the policy. The *actor* is the policy that produces actions; the *critic* is the value function that supplies the learning signal, as the baseline just described or as a bootstrap replacing the part of the return beyond a short window. The critic therefore serves training only: it does not select or refine actions after the actor has run, and at deployment only the actor is required.

Being on-policy also decides how large an update may be. The estimate in Equation (2.3) describes the policy that collected the batch and stays informative only in a neighborhood of it. Too small a step wastes the batch; too large a step moves the policy into a region that the collected advantages say nothing about, where performance can collapse, and recovery is slow because the next batch is gathered by the degraded policy. Trust Region Policy Optimization answers this by constraining the Kullback-Leibler divergence between the new policy and the one that collected the data at every update [22]. Proximal Policy Optimization (PPO) obtains a similar effect far more cheaply, replacing the constrained problem with a clipped surrogate objective [23]:

$$L^{\text{clip}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{2.4}$$

where $r_t(\theta) = \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)/\pi_{\theta_{\text{old}}}(\mathbf{a}_t \mid \mathbf{s}_t)$ is the ratio between the probability the updated policy assigns to the action that was taken and the probability assigned by the policy that collected the batch, $\hat{A}_t$ is the estimated advantage and ϵ the width of the clipping range. Taking the smaller of the two terms makes the objective a pessimistic estimate of the improvement: once the ratio leaves $1 \pm \epsilon$ in the direction the advantage favors, moving it further stops paying, so the update has no incentive to leave the neighborhood of the policy that collected the data. This is also what makes it safe to take several epochs of minibatch updates on the same batch, which is where much of the sample efficiency of PPO comes from. The estimate $\hat{A}_t$ is itself

usually formed by Generalized Advantage Estimation, which interpolates with a parameter λ between a one-step temporal-difference residual, of low variance but biased by the current value function, and the full return, unbiased but noisy [24]. These methods are well suited to simulated motor control because thousands of environments can collect on-policy experience in parallel [6].

For continuous control, the actor commonly parameterizes a diagonal Gaussian distribution. Its mean gives the nominal action and its standard deviation controls exploration during training and typically shrinks as it progresses. At deployment, selecting the mean makes the behavior deterministic.

Another frequent choice concerns what the action represents. Instead of commanding a joint torque directly, the policy can emit a vector of desired joint positions, one per control step, scaled to the robot's range and converted into torques by a proportional-derivative controller. This keeps the learned policy at a lower control frequency while a simple feedback loop handles fast joint-level corrections.

2.1.3 Goal-Conditioned Policies

A goal-conditioned policy receives, together with the state, a goal $\mathbf{g} \in \mathcal{G}$ describing what it should achieve, and acts according to $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{g})$ [25, 26]. The goal is made explicit in a goal-conditioned MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{G}, P, R, \gamma, \rho_0, \rho_{\mathbf{g}} \rangle$, which adds to the MDP of Section 2.1.1 the goal space $\mathcal{G}$ and the distribution $\rho_{\mathbf{g}}$ from which the goal of each episode is drawn, and in which the reward becomes a function of the goal as well, $R(\mathbf{s}, \mathbf{a}, \mathbf{g})$. The value functions $V^\pi(\mathbf{s}, \mathbf{g})$ and $Q^\pi(\mathbf{s}, \mathbf{a}, \mathbf{g})$ are learned jointly over states and goals. Universal value function approximators showed that a single network trained on a subset of goals can generalize to goals it was not trained on, provided that the goal space is smooth, with nearby goals calling for similar behavior [26].

The simplest reward of this kind is sparse: a positive signal when the goal is achieved, none otherwise. It is unambiguous but hard to learn from, since a policy that never reaches the goal receives no information about how close it came to it. Hindsight experience replay addresses this for off-policy learning by relabeling each recorded trajectory with a goal it did achieve, turning a failure for one goal into a success for another [27]. The alternative is a dense reward that measures progress, for example the negative distance between the current state and the goal. Such a reward is informative everywhere, but it is a shaped reward, which has to be designed by hand, and there is no universal definition for distance. A dense reward can also change what the agent learns: instead of reaching the goal, the policy may find a way to keep collecting the extra reward, for example by moving back and forth in a region where it is high. Potential-based shaping avoids this [28]. A potential $\Phi(\mathbf{s})$ assigns a value to each state, for example the negative distance to the goal, and the extra reward for a transition from $\mathbf{s}$ to $\mathbf{s}'$ is the change in potential,

$$F(\mathbf{s}, \mathbf{s}') = \gamma \, \Phi(\mathbf{s}') - \Phi(\mathbf{s}). \tag{2.5}$$

Moving closer to the goal is rewarded and moving away is penalized. When $\gamma = 1$ the two amounts are equal, so going back and forth earns nothing; when $\gamma < 1$ they differ slightly and a cycle can earn a small amount, but this does not change which behavior is best. Summed over a trajectory with their discounts, the extra rewards

cancel out except for those of the first and the last state, and for this reason adding them does not change the optimal policy.

Some goals must be reached within a deadline. Without a time signal, the process is non-Markovian, and the learned value function and policy average over situations that call for different behavior [29]: states that differ only in the time left become indistinguishable, even though they may call for very different actions. A policy asked to reach a goal by a deadline must therefore receive the time remaining, or an equivalent phase variable, as part of its input.

2.2 Hierarchical Reinforcement Learning

Hierarchical reinforcement learning splits control into levels: a high level decides which behavior should be executed, and a low level executes it through primitive actions. Two interfaces between the levels are common. In the first, the high level selects one among a set of temporally extended behaviors, formalized as options [8]. In the second, the high level selects a goal, and a single goal-conditioned policy (Section 2.1.3) is trained to reach any goal it receives [30, 31]. The two interfaces are compatible. The formalization of abstract-action-aware options, presented in Sections 2.2.4 and 2.2.5, is a novel concept that is under study within our research group.

2.2.1 Options

Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \ldots \rangle$ (Section 2.1.1), an *option* is a temporally extended action [8]

$$\omega = \langle \mathcal{I}, \pi(\mathbf{a} \mid \mathbf{s}), \beta(\mathbf{s}) \rangle, \tag{2.6}$$

where $\mathcal{I} \subseteq \mathcal{S}$ is the *initiation set*, the set of states in which the option can start, $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is the *intra-option policy*, which gives a probability distribution over actions in each state, and $\beta : \mathcal{S} \to [0, 1]$ is the *termination condition*, which gives the probability that the option terminates when it enters a given state. An option is started in a state of its initiation set, selects actions with its policy, and stops according to its termination condition. It is the standard formalism used whenever a behavior that spans many primitive actions has to be reused. A high-level policy that selects options, instead of primitive actions, decides only which behavior to execute, and leaves its execution to the option.

This definition separates three terms that are often used interchangeably. A *policy* is a mapping from states to distributions over actions, as in Section 2.1.1. A *skill* is a policy together with the states from which it is expected to succeed, its initiation set, and an *option* is a skill to which a termination condition has been added.

Two options can be executed one after the other only if the second one can start where the first one terminates. Let $\omega_1 = \langle \mathcal{I}_1, \pi_1, \beta_1 \rangle$ and $\omega_2 = \langle \mathcal{I}_2, \pi_2, \beta_2 \rangle$. Since $\beta_1(\mathbf{s})$ is the probability that ω_1 terminates in $\mathbf{s}$, the states in which ω_1 can terminate are those with $\beta_1(\mathbf{s}) > 0$. The sequence $\omega_1; \omega_2$ is *always sequenceable* if ω_2 can start in every state in which ω_1 can terminate,

$$\forall \mathbf{s} \in \mathcal{S} : \quad \beta_1(\mathbf{s}) > 0 \;\Rightarrow\; \mathbf{s} \in \mathcal{I}_2, \tag{2.7}$$

and *sometimes sequenceable* if ω_2 can start in at least one of them,

$$\exists \mathbf{s} \in \mathcal{S} : \quad \beta_1(\mathbf{s}) > 0 \ \wedge \ \mathbf{s} \in \mathcal{I}_2. \tag{2.8}$$

The first is the *composition condition*: when it holds, the composition of ω_1 and ω_2 is well defined wherever ω_1 terminates. When only the second holds, the composition is well defined only if ω_1 happens to terminate in a state of $\mathcal{I}_2$, and fails otherwise.

2.2.2 Local-Coordinate and Goal-Conditioned Options

The policy and the termination condition of Equation (2.6) act on the global state. Many skills, however, do not depend on where they are executed: moving one meter forward, or turning left, is the same behavior wherever it starts. A *local-coordinate option* expresses its policy and termination in a local state space Σ,

$$\omega = \langle \mathcal{I}, \pi(\mathbf{a} \mid \boldsymbol{\sigma}), \beta(\boldsymbol{\sigma}), \psi \rangle, \tag{2.9}$$

where $\psi : \mathcal{S} \to \Sigma$ maps global states to local ones, for example by a change of coordinates, relative to a *reference state* $\mathbf{s}_{\text{ref}} \in \mathcal{S}$, the origin of the local coordinates, captured when the option starts. The local state is $\boldsymbol{\sigma} = \psi(\mathbf{s} \mid \mathbf{s}_{\text{ref}})$, and the option is defined through it:

$$\mathcal{I} = \{\mathbf{s} \in \mathcal{S} \mid \psi(\mathbf{s} \mid \mathbf{s}_{\text{ref}}) \text{ is defined}\}, \tag{2.10}$$

$$\beta(\boldsymbol{\sigma}) = \beta(\psi(\mathbf{s} \mid \mathbf{s}_{\text{ref}})), \qquad \pi(\mathbf{a}_t \mid \boldsymbol{\sigma}_t) = \pi(\mathbf{a}_t \mid \psi(\mathbf{s}_t \mid \mathbf{s}_{\text{ref}})). \tag{2.11}$$

The inverse mapping ψ^{-1}, from local to global coordinates, is in general not needed. Since the option reasons in relative terms, the same skill works anywhere in the global space. Most humanoid skills are expressed this way, in a frame attached to the robot.

A *goal-conditioned option* turns the goal-conditioned policy of Section 2.1.3 into a family of options, one for each goal $\mathbf{g}$ in a goal space $\mathcal{G}$,

$$\omega(\mathbf{g}) = \langle \mathcal{I}, \pi(\mathbf{a} \mid \mathbf{s}, \mathbf{g}), \beta(\mathbf{s} \mid \mathbf{g}), \mathcal{G}, f \rangle, \tag{2.12}$$

where the mapping $f : \mathcal{S} \to \mathcal{G}$ defines the goal space and $\|\cdot\|$ is a norm in $\mathcal{G}$. The option terminates when the goal is reached within a tolerance ϵ,

$$\beta(\mathbf{s} \mid \mathbf{g}) = \begin{cases} 1 & \text{if } \|f(\mathbf{s}) - \mathbf{g}\| \leq \epsilon, \\ 0 & \text{otherwise,} \end{cases} \tag{2.13}$$

and its policy is trained with a positive reward R_+ when the goal is reached and the negative distance to it otherwise,

$$R(\mathbf{s}, \mathbf{a}, \mathbf{s}' \mid \mathbf{g}) = \begin{cases} R_+ & \text{if } \|f(\mathbf{s}') - \mathbf{g}\| \leq \epsilon, \\ -\|f(\mathbf{s}') - \mathbf{g}\| & \text{otherwise,} \end{cases} \tag{2.14}$$

which gives a dense learning signal. For instance, if $f(\mathbf{s})$ is the position of the agent, the option $\omega(\mathbf{g})$ reaches the position $\mathbf{g}$, and a single policy serves every position in $\mathcal{G}$. This reward assumes that the norm $\|\cdot\|$ in the goal space does not introduce

significant local minima in $\|f(\mathbf{s}') - \mathbf{g}\|$, so that reducing the distance to the goal leads the agent toward it. In a complex environment this may not hold: an obstacle can separate the agent from a goal that is close in $\mathcal{G}$, and moving straight toward the goal leads into the obstacle. The norm may then be replaced by a high-level planning engine, for example a path planner that measures the distance to the goal along a feasible path.

In a goal-conditioned option, the goal is defined through f on the global state, so the same behavior, executed from different initial states, corresponds to different goals. With $f(\mathbf{s})$ the position of the agent, reaching the point at a fixed displacement from the initial position requires a different goal $\mathbf{g}$ for every initial position. A *goal-conditioned local option* removes this dependence by combining the two previous constructions: the goal is defined on the local state, through a mapping $f_\psi : \Sigma \to \mathcal{G}$,

$$\omega(\mathbf{g}) = \langle \mathcal{I}, \pi(\mathbf{a} \mid \boldsymbol{\sigma}, \mathbf{g}), \beta(\boldsymbol{\sigma} \mid \mathbf{g}), \psi, \mathcal{G}, f_\psi \rangle, \tag{2.15}$$

with

$$\beta(\boldsymbol{\sigma} \mid \mathbf{g}) = \begin{cases} 1 & \text{if } \|f_\psi(\boldsymbol{\sigma}) - \mathbf{g}\| \leq \epsilon, \\ 0 & \text{otherwise,} \end{cases} \tag{2.16}$$

and its policy is trained with

$$R(\boldsymbol{\sigma}, \mathbf{a}, \boldsymbol{\sigma}' \mid \mathbf{g}) = \begin{cases} R_+ & \text{if } \|f_\psi(\boldsymbol{\sigma}') - \mathbf{g}\| \leq \epsilon, \\ -\|f_\psi(\boldsymbol{\sigma}') - \mathbf{g}\| & \text{otherwise.} \end{cases} \tag{2.17}$$

Goals are thus defined relative to the reference state, and the same goal denotes the same behavior from every initial state. For instance, if $f_\psi(\boldsymbol{\sigma})$ is the position of the agent in the frame of $\mathbf{s}_{\text{ref}}$, with the first axis along the heading of the agent at $\mathbf{s}_{\text{ref}}$ and the second toward its right, the goal $\mathbf{g} = (2, 1)$ denotes the point two meters ahead and one meter to the right of the initial position. A single option $\omega(\mathbf{g})$ then reaches this displacement from any initial state.

2.2.3 State Abstraction and ϕ-Relative Options

Hierarchical control often plans on a coarser description of the problem than the one in which it acts. The *ground MDP* $\langle \mathcal{S}, \mathcal{A}, \ldots \rangle$ is linked to a coarser *abstract MDP* $\langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, \ldots \rangle$ by a state abstraction

$$\phi : \mathcal{S} \to \bar{\mathcal{S}}, \tag{2.18}$$

which maps each ground state $\mathbf{s}$ to an abstract state $\bar{\mathbf{s}} = \phi(\mathbf{s})$. The relation is many-to-one, and each abstract state corresponds to the *block* of ground states that map to it,

$$[\bar{\mathbf{s}}] = \{\mathbf{s} \in \mathcal{S} \mid \phi(\mathbf{s}) = \bar{\mathbf{s}}\}, \tag{2.19}$$

so that the blocks form a partition of $\mathcal{S}$. Planning happens in the small abstract MDP, and produces a *plan*, a sequence of abstract actions $\langle \bar{a}_1, \ldots, \bar{a}_n \rangle$, while execution happens in the large ground MDP.

The concepts of this section are illustrated with the example of Figure 2.1. A mobile robot, the agent, has to move through a building made of identical square

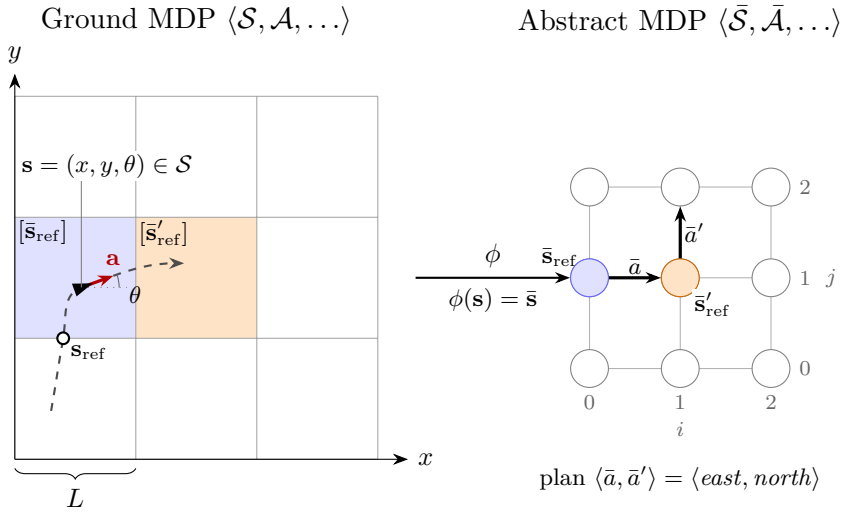


Figure 2.1. Ground and abstract MDP of the building example. Left: in the ground MDP, the state $\mathbf{s} = (x, y, \theta)$ is the pose of the agent and the ground actions $\mathbf{a}$ are small motions. The dashed line is a ground trajectory that enters the room $\bar{\mathbf{s}}_{\text{ref}}$ in the reference state $\mathbf{s}_{\text{ref}}$ and executes the abstract action $\bar{a} = east$, reaching the block $[\bar{\mathbf{s}}'_{\text{ref}}]$; in the local coordinates of $\mathbf{s}_{\text{ref}}$, this block is the target block E. Right: in the abstract MDP, each node is a room, a cell (i, j) of the grid, the abstraction ϕ of Equation (2.20) maps every pose to the cell that contains it, and the plan is a sequence of abstract actions.

rooms of side L, each connected to the adjacent ones. Deciding which rooms to cross to go from one room to another is easy, while moving the robot from one room to the next requires a long sequence of small motions. The problem therefore splits naturally into two levels: a high level that plans the sequence of rooms, and a low level that moves the robot between them. For planning, the building is abstracted as a grid in which each room is a cell. The ground state $\mathbf{s} = (x, y, \theta)$ of the agent is its position and heading on the floor, and its ground actions $\mathbf{a}$ are small motions, such as a step forward along the heading or a rotation by a small angle. The abstraction maps each ground state to the cell that contains it,

$$\phi(\mathbf{s}) = (\lfloor x/L \rfloor, \lfloor y/L \rfloor), \tag{2.20}$$

so that each block is the set of poses inside one room, and the abstract actions $\bar{\mathcal{A}} = \{north, south, east, west\}$ move the agent to the adjacent room in the given direction.

An abstract action is only a label in the abstract MDP: to be executed, it needs an option that realizes the corresponding transition through ground actions. The options defined with respect to ϕ provide this link between the two levels [32]. A *ϕ-relative transition option* between two abstract states $\bar{\mathbf{s}}_i$ and $\bar{\mathbf{s}}_j$,

$$\omega_{i \to j} = \langle \mathcal{I}_{i \to j}, \pi_{i \to j}(\mathbf{a} \mid \mathbf{s}), \beta_{i \to j}(\mathbf{s}), \phi \rangle, \tag{2.21}$$

can start anywhere in the block of $\bar{\mathbf{s}}_i$, $\mathcal{I}_{i \to j} = [\bar{\mathbf{s}}_i]$, and terminates when the agent enters the block of $\bar{\mathbf{s}}_j$, $\beta_{i \to j}(\mathbf{s}) = 1$ if $\phi(\mathbf{s}) = \bar{\mathbf{s}}_j$ and 0 otherwise. Its policy is trained with the sparse reward R_+ for entering $[\bar{\mathbf{s}}_j]$ and -1 for every other action. An *exit*

option $\omega_{i\to *}$ terminates instead as soon as the agent leaves $[\bar{\mathbf{s}}_i]$, whichever block it enters.

A *ϕ-relative local-coordinate option*

$$\omega_{i\to j} = \langle \mathcal{I}, \pi_{i\to j}(\mathbf{a} \mid \boldsymbol{\sigma}), \beta_{i\to j}(\boldsymbol{\sigma}), \phi, \psi \rangle \tag{2.22}$$

executes the same transition in local coordinates. Its reference state $\mathbf{s}_{\text{ref}} \in [\bar{\mathbf{s}}_i]$ is the state in which it enters $\bar{\mathbf{s}}_i$, $\mathcal{I} = [\bar{\mathbf{s}}_i]$, and the block it has to reach is expressed in the local frame as the *target block*

$$E = \{\psi(\mathbf{s} \mid \mathbf{s}_{\text{ref}}) \mid \mathbf{s} \in [\bar{\mathbf{s}}_j]\} \subset \Sigma, \tag{2.23}$$

so that the option terminates when $\boldsymbol{\sigma} \in E$, and its policy is rewarded with R_+ when $\boldsymbol{\sigma}' \in E$ and -1 otherwise. The behavior is more transferable, since the policy does not see absolute coordinates, but a separate option is still needed for every pair (i, j): everything that defines the task, where it starts, where it ends and what it is rewarded for, depends on the pair, while the policy only sees the local state.

2.2.4 Abstract-Action-Aware Options

A ϕ-relative option is tied to a pair of abstract states. If an abstract action has the same meaning in every abstract state in which it is possible, a single option per abstract action is enough. Let $\bar{\mathcal{S}}_{\bar{a}} \subseteq \bar{\mathcal{S}}$ be the set of abstract states in which the abstract action $\bar{a}$ is possible. An option for $\bar{a}$ is started in a reference state $\mathbf{s}_{\text{ref}}$, the state in which the agent enters the current abstract state $\bar{\mathbf{s}}_{\text{ref}} = \phi(\mathbf{s}_{\text{ref}})$, and has to reach $\bar{\mathbf{s}}'_{\text{ref}}$, the abstract state expected after the execution of $\bar{a}$. Its target block is

$$E = \{\psi(\mathbf{s} \mid \mathbf{s}_{\text{ref}}) \mid \mathbf{s} \in [\bar{\mathbf{s}}'_{\text{ref}}]\} \subset \Sigma. \tag{2.24}$$

The pair of abstract states is no longer fixed: any transition performed by $\bar{a}$ is denoted with $* \xrightarrow{\bar{a}} *$, where the asterisks stand for any abstract state.

We assume that options associated with abstract actions are valid and effective in all the ground states that correspond to abstract states in which the abstract actions are possible. Under this homogeneity assumption, executing $\bar{a}$ is the same problem, in local coordinates, from every state in which it can start. Whether this holds depends on the local coordinates: they must describe every abstract state in which $\bar{a}$ is possible in the same way. In the example of Figure 2.1, let ψ express the pose relative to the corner of the cell $(i, j) = \phi(\mathbf{s}_{\text{ref}})$ that contains the reference state,

$$\psi(\mathbf{s} \mid \mathbf{s}_{\text{ref}}) = (x - iL,\ y - jL,\ \theta). \tag{2.25}$$

For the abstract action *east*, the target block is then $E = [L, 2L) \times [0, L) \times [-\pi, \pi)$ from every cell, and since all the rooms are identical the agent moves in the same way in local coordinates wherever it is, so the assumption holds. The *abstract-action-aware ϕ-relative local-coordinate option* for $\bar{a}$ is

$$\omega_{*\xrightarrow{\bar{a}}*} = \langle \mathcal{I}, \pi_{*\xrightarrow{\bar{a}}*}(\mathbf{a} \mid \boldsymbol{\sigma}, \bar{a}), \beta_{*\to *}(\boldsymbol{\sigma} \mid \bar{a}), \phi, \psi \rangle, \tag{2.26}$$

with initiation set $\mathcal{I} = \{\mathbf{s} \in \mathcal{S} \mid \phi(\mathbf{s}) \in \bar{\mathcal{S}}_{\bar{a}}\}$, the ground states of every abstract state in which $\bar{a}$ is possible, and with termination condition and reward

$$\beta_{*\to*}(\boldsymbol{\sigma} \mid \bar{a}) = \begin{cases} 1 & \text{if } \boldsymbol{\sigma} \in E, \\ 0 & \text{otherwise,} \end{cases} \qquad R_{*\to*}(\boldsymbol{\sigma}, \mathbf{a}, \boldsymbol{\sigma}' \mid \bar{a}) = \begin{cases} R_+ & \text{if } \boldsymbol{\sigma}' \in E, \\ -1 & \text{otherwise.} \end{cases} \tag{2.27}$$

A plan $\langle \bar{a}_1, \dots, \bar{a}_n \rangle$ is executed by the composition $\langle \omega_{*\xrightarrow{\bar{a}_1}*}, \dots, \omega_{*\xrightarrow{\bar{a}_n}*} \rangle$, in which each option starts, with a new reference state, where the previous one terminated. If each abstract action of the plan is possible in the abstract state reached by the previous ones, the composition condition of Equation (2.7) holds by construction, since every option terminates in a block that belongs to the initiation set of the next one. The resulting policy is suboptimal, but very sample-efficient, since only $|\bar{\mathcal{A}}|$ options have to be trained.

The approach has three problems. First, where the abstraction is too coarse, the assumption does not hold and the shared option may fail. Second, the option might terminate in the first state of E it reaches, every such state being equally acceptable to it, and the next option has to start from whichever state it receives. Third, the reward of Equation (2.27) is sparse.

The third problem is solved by placing goals on the target block. With the mapping f_ψ of Section 2.2.2, the *target goals* of $\bar{a}$ and the target goal closest to a local state $\boldsymbol{\sigma}$ are

$$G_{\bar{a}} = \{f_\psi(\boldsymbol{\sigma}) \mid \boldsymbol{\sigma} \in E\} \subset \mathcal{G}, \qquad \mathbf{g}_{\text{closest}}(\boldsymbol{\sigma}, G_{\bar{a}}) = \arg\min_{\mathbf{h} \in G_{\bar{a}}} \|f_\psi(\boldsymbol{\sigma}) - \mathbf{h}\|. \tag{2.28}$$

The *abstract-action-aware ϕ-relative local-coordinate goal-conditioned option*

$$\omega_{*\xrightarrow{\bar{a}}*} = \langle \mathcal{I}, \pi_{*\xrightarrow{\bar{a}}*}(\mathbf{a} \mid \boldsymbol{\sigma}, \bar{a}), \beta_{*\to*}(\boldsymbol{\sigma} \mid \bar{a}), \phi, \psi, f_\psi \rangle \tag{2.29}$$

has the same initiation set and termination condition, and its policy is trained with the negative distance to the closest goal in the next abstract state,

$$R_{*\to*}(\boldsymbol{\sigma}, \mathbf{a}, \boldsymbol{\sigma}' \mid \bar{a}) = \begin{cases} R_+ & \text{if } \boldsymbol{\sigma}' \in E, \\ -\|f_\psi(\boldsymbol{\sigma}') - \mathbf{g}_{\text{closest}}(\boldsymbol{\sigma}', G_{\bar{a}})\| & \text{otherwise,} \end{cases} \tag{2.30}$$

which is dense and keeps one option per abstract action. The second problem is addressed in the next section.

2.2.5 2-Abstract-Action-Aware Options

The abstract-action-aware option executes $\bar{a}$ without knowing what comes next. Consider instead two consecutive transitions in the abstract space, performed by two abstract actions $\bar{a}$ and $\bar{a}'$,

$$* \xrightarrow{\bar{a}} \bar{\mathbf{s}}, \qquad \bar{\mathbf{s}} \xrightarrow{\bar{a}'} *, \tag{2.31}$$

and denote the double transition with $* \xrightarrow{\bar{a},\bar{a}'} *$. The state in which the execution of $\bar{a}$ ends, called the *transition state* between $\bar{a}$ and $\bar{a}'$, is the state from which $\bar{a}'$ will

be executed. The abstract-action-aware option only requires it to be in the block of the next abstract state, and therefore in the initiation set $\mathcal{I}_{\bar{a}'}$ of $\bar{a}'$,

$$\mathbf{s}_{\text{end}} \in [\bar{\mathbf{s}}'_{\text{ref}}] \subseteq \mathcal{I}_{\bar{a}'}. \tag{2.32}$$

This guarantees that $\bar{a}'$ can start, but not that it is executed well: every state of the block is equally acceptable to the abstract-action-aware option, while some are better than others for $\bar{a}'$. Taking $\bar{a}'$ into account, the choice of the transition state can be posed in two ways: as a *classification problem*, ending the execution of $\bar{a}$ in a state of $[\bar{\mathbf{s}}'_{\text{ref}}]$ from which $\bar{a}'$ is executed well enough (whose value $V_{\bar{a}'}(\mathbf{s})$ is above an acceptance level), and as an *optimization problem*, ending it in the state from which $\bar{a}'$ is executed best,

$$\mathbf{s}_{\text{end}} \in \arg\max_{\mathbf{s}\in[\bar{\mathbf{s}}'_{\text{ref}}]} V_{\bar{a}'}(\mathbf{s}), \tag{2.33}$$

where $V_{\bar{a}'}(\mathbf{s})$ is the expected return of the option for $\bar{a}'$ started in $\mathbf{s}$. The 2-abstract-action-aware option, an option for $\bar{a}$ that knows that $\bar{a}'$ comes next, can address either problem. It executes only $\bar{a}$, not the sequence of the two abstract actions, but it chooses its transition state in $\mathcal{I}_{\bar{a}'}$ according to $\bar{a}'$, and to do so it needs a measure of how good a transition state is for $\bar{a}'$.

This measure is defined through the abstract-action-aware option $\omega_{*\xrightarrow{\bar{a}'}*}$ of $\bar{a}'$ (Equation (2.26)), which is trained beforehand. The quality of a local state $\boldsymbol{\sigma} \in E$ as a transition state toward $\bar{a}'$ is a function

$$\mathit{good}(\cdot \mid \bar{a}') : E \to [0, 1]. \tag{2.34}$$

Let $\mathbf{s} \in [\bar{\mathbf{s}}'_{\text{ref}}]$ be the ground state such that $\boldsymbol{\sigma} = \psi(\mathbf{s} \mid \mathbf{s}_{\text{ref}})$. With an abuse of notation, $\mathit{good}(\boldsymbol{\sigma} \mid \bar{a}') \doteq \mathit{good}(\mathbf{s} \mid \bar{a}')$: the ground state is always available during execution, so the inverse mapping ψ^{-1} is not needed. The expected return $V^{\pi}_{*\xrightarrow{\bar{a}'}*}(\mathbf{s})$ of executing $\omega_{*\xrightarrow{\bar{a}'}*}$ from $\mathbf{s}$, with $\mathbf{s}$ as its new reference state and under its reward $R_{*\to*}(\cdot \mid \bar{a}')$, can be taken from the critic learned while training $\pi_{*\xrightarrow{\bar{a}'}*}$, or estimated by Monte Carlo, from N rollouts of $\pi_{*\xrightarrow{\bar{a}'}*}$ starting from $\mathbf{s}$, each one ending when $\beta_{*\to*}(\cdot \mid \bar{a}') = 1$ or after $T_{\max}$ steps:

$$\hat{V}_{\bar{a}'}(\mathbf{s}) = \frac{1}{N} \sum_{k=1}^{N} \sum_{t=0}^{T_k-1} \gamma^t R_{*\to*}(\boldsymbol{\sigma}_t^{(k)}, \mathbf{a}_t^{(k)}, \boldsymbol{\sigma}_{t+1}^{(k)} \mid \bar{a}'). \tag{2.35}$$

With $V_{\min}$ and $V_{\max}$ the bounds of the return of $\omega_{*\xrightarrow{\bar{a}'}*}$, the quality of the transition state is the normalized value

$$\mathit{good}(\boldsymbol{\sigma} \mid \bar{a}') = \text{clip}_{[0,1]}\left(\frac{\hat{V}_{\bar{a}'}(\mathbf{s}) - V_{\min}}{V_{\max} - V_{\min}}\right). \tag{2.36}$$

With the sparse reward of Equation (2.27), for example, $V_{\max} = R_+$, in case of immediate success, and $V_{\min} = -(1-\gamma^{T_{\max}})/(1-\gamma)$, when there is no success within $T_{\max}$ steps. The set of abstract actions is extended with a terminal action *STOP*, always present at the end of a plan, for which $\mathit{good}(\boldsymbol{\sigma} \mid \mathit{STOP}) = 1$ for every $\boldsymbol{\sigma} \in E$.

The *2-abstract-action-aware ϕ-relative local-coordinate option* for $\bar{a}$ followed by $\bar{a}'$ is

$$\begin{aligned}\omega_{*\xrightarrow{\bar{a},\bar{a}'}*} = \langle\, \mathcal{I}_{\bar{a}},\ \pi_{*\xrightarrow{\bar{a},\bar{a}'}*}(\mathbf{a} \mid \boldsymbol{\sigma}, \bar{a}, \bar{a}'),\\ \beta_{*\to*}(\boldsymbol{\sigma} \mid \bar{a}, \bar{a}'),\ \phi,\ \psi,\ f_\psi \,\rangle,\end{aligned} \tag{2.37}$$

where the reference state $\mathbf{s}_{\text{ref}}$, the abstract states $\bar{\mathbf{s}}_{\text{ref}}$ and $\bar{\mathbf{s}}'_{\text{ref}}$, the target block E, the target goals $G_{\bar{a}}$, the closest goal $\mathbf{g}_{\text{closest}}$ and the initiation set $\mathcal{I}_{\bar{a}} = \{\mathbf{s} \mid \phi(\mathbf{s}) \in \bar{\mathcal{S}}_{\bar{a}}\}$ are those of $\bar{a}$, as in Section 2.2.4, and $\bar{a}'$ enters only through $good(\cdot \mid \bar{a}')$. Given an acceptance threshold $\tau \in (0, 1]$, the termination condition is defined in one of two ways, corresponding to the two problems above. As a classification problem,

$$\beta_{*\to*}(\boldsymbol{\sigma} \mid \bar{a}, \bar{a}') = \begin{cases} 1 & \text{if } \boldsymbol{\sigma} \in E \text{ and } good(\boldsymbol{\sigma} \mid \bar{a}') \geq \tau, \\ 0 & \text{otherwise,} \end{cases} \tag{2.38}$$

and as an optimization problem,

$$\beta_{*\to*}(\boldsymbol{\sigma} \mid \bar{a}, \bar{a}') = \begin{cases} good(\boldsymbol{\sigma} \mid \bar{a}') & \text{if } \boldsymbol{\sigma} \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{2.39}$$

The policy $\pi_{*\xrightarrow{\bar{a},\bar{a}'}*}$ is trained with the reward

$$\begin{aligned} &R_{*\to*}(\boldsymbol{\sigma}, \mathbf{a}, \boldsymbol{\sigma}' \mid \bar{a}, \bar{a}') = \\ &\quad \begin{cases} R_+ & \text{if } \boldsymbol{\sigma}' \in E \text{ and } good(\boldsymbol{\sigma}' \mid \bar{a}') \geq \tau, \\ -(1 - good(\boldsymbol{\sigma}' \mid \bar{a}')) & \text{if } \boldsymbol{\sigma}' \in E \text{ and } good(\boldsymbol{\sigma}' \mid \bar{a}') < \tau, \\ -\|f_\psi(\boldsymbol{\sigma}') - \mathbf{g}_{\text{closest}}(\boldsymbol{\sigma}', G_{\bar{a}})\| & \text{otherwise,} \end{cases} \end{aligned} \tag{2.40}$$

and, for the optimization definition of β, the first two cases can be replaced by $R_+ \cdot good(\boldsymbol{\sigma}' \mid \bar{a}')$ for $\boldsymbol{\sigma}' \in E$.

The third case of Equation (2.40) is the dense reward of Equation (2.30), which leads the option to the target block. The second case gives a dense signal inside the target block, toward better transition states: without it, the distance term would be zero for every $\boldsymbol{\sigma}' \in E$, and the option would have no incentive to move from an acceptable transition state to a better one. With either termination condition, the option no longer stops at whichever state of E it reaches first, but at a transition state $\mathbf{s}_{\text{tr}} \in \mathcal{I}_{\bar{a}'}$, with $\boldsymbol{\sigma}_{\text{tr}} = \psi(\mathbf{s}_{\text{tr}} \mid \mathbf{s}_{\text{ref}})$, chosen according to its quality for $\bar{a}'$. With the classification definition, it stops at the first state of E that is good enough for $\bar{a}'$. With the optimization definition, the better a state is for $\bar{a}'$, the more likely the option is to stop there, and its policy is rewarded for reaching states of high quality: this approximates the optimization problem of Equation (2.33), with the value of $\bar{a}'$ measured by its abstract-action-aware option, which executes $\bar{a}'$ without knowing what follows it. For example, if $\bar{a}'$ requires the agent to move in a different direction from $\bar{a}$, the option for $\bar{a}$ alone may end with the agent moving fast in the direction of $\bar{a}$, so that it has to stop and turn before it can execute $\bar{a}'$; the option for $\bar{a}$ followed by $\bar{a}'$ can instead end with the agent already turning toward the direction of $\bar{a}'$.

A plan $\langle \bar{a}_1, \ldots, \bar{a}_n \rangle$ is executed by the composition of the options

$$\langle \omega_{*\xrightarrow{\bar{a}_1,\bar{a}_2}*},\ \omega_{*\xrightarrow{\bar{a}_2,\bar{a}_3}*},\ \ldots,\ \omega_{*\xrightarrow{\bar{a}_{n-1},\bar{a}_n}*},\ \omega_{*\xrightarrow{\bar{a}_n,STOP}*} \rangle, \tag{2.41}$$

where the last option, since $good(\cdot \mid STOP) = 1$, reduces to the abstract-action-aware option of $\bar{a}_n$.

The 2-abstract-action-aware option looks one abstract action ahead. A more general concept is a *plan-conditioned policy*, conditioned on the whole future plan,

$$\pi(\mathbf{a} \mid \boldsymbol{\sigma}, \bar{a}_0, \bar{a}_1, \ldots, \bar{a}_k), \tag{2.42}$$

where $\bar{a}_0$ is the abstract action in progress, $\boldsymbol{\sigma}$ the local state relative to it, and $\bar{a}_1, \ldots, \bar{a}_k$ the future plan to the high-level goal. An internal mechanism of the policy computes the switch of abstract action,

$$\pi(\mathbf{a} \mid \boldsymbol{\sigma}, \bar{a}_0, \bar{a}_1, \ldots, \bar{a}_k) \rightarrow \pi(\mathbf{a} \mid \boldsymbol{\sigma}, \bar{a}_1, \bar{a}_2, \ldots, \bar{a}_k), \tag{2.43}$$

in which $\bar{a}_0$ is considered terminated and $\bar{a}_1$ becomes the abstract action in progress.

2.3 Human Motion Data

Much of what a humanoid robot should do is easier to show than to describe with a reward: how a person walks, turns, stops or jumps is captured directly by recordings of people doing it. Recorded human motion is therefore the main source of data for the policies of this thesis: the tracking policies of Section 2.4.2 learn to reproduce it, and the bridge is trained on a corpus built from it. This section describes how human motion is recorded and transferred to the body of a robot, the public datasets and tools from which the corpus of this thesis is built (Section 5.2), and the motions published already retargeted to a robot, from which some of the skills used in the experiments are trained (Section 6.2.3).

2.3.1 Motion Capture and Retargeting

A human performer is recorded with a motion capture system, and the motion is then *retargeted* to the robot: it is mapped from the human body to the body of the robot, which differs from it in size, proportions, number of joints and joint limits. The result, called a *kinematic trajectory*, is a sequence of robot poses and velocities that describes how the robot should move, but not whether it can: it does not account for physics, actuator limits or contacts; nothing ensures the robot is able to execute it.

GMR [33] is a general retargeting tool for humanoid robots. It reads a human motion and solves, for each frame, an inverse kinematics problem that places the key bodies of the robot on those of the human, scaled by the height of the performer. Each frame is solved independently of the others, and the problem does not include the floor. The result therefore keeps the noise of the recording and adds that of the retargeting: the robot can hover above the ground or sink below it, its height can drift within a take, a foot that should be planted can slide, and the motion can jitter from one frame to the next. How the corpus of this thesis deals with these artifacts is described in Sections 5.2.2 and 5.2.3.

2.3.2 LAFAN1

LAFAN1 was introduced by Harvey et al. for learning motion transitions [34]. It contains long, continuous takes performed by five actors and recorded at 30 Hz, in which each actor performs a sequence of related movements around a theme, such as walking, running, jumping, dancing or fighting. Since the takes are continuous, they contain not only steady motions, but also the starts, stops, turns and changes of speed between them. Figure 2.2 shows a part of one take.

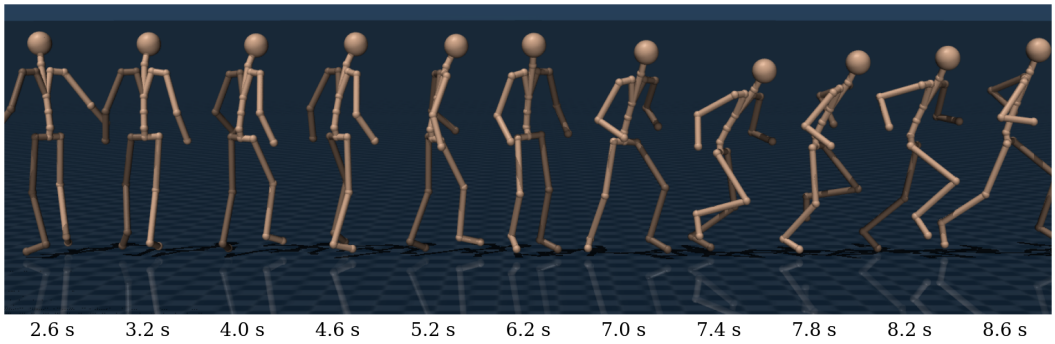


Figure 2.2. Frames of a LAFAN1 take, rendered on the skeleton. The take is continuous, so it contains the changes between the motions as well as the motions themselves.

2.3.3 AMASS and BABEL

AMASS [35] is a large collection of human motion capture. It unifies datasets recorded in different laboratories with different marker setups, fifteen in its first release, by fitting the same parametric body model to all of them, so that every take is described in the same way. Some of these are CMU, the motion capture database of the Carnegie Mellon University [36], KIT, the whole-body human motion database of the Karlsruhe Institute of Technology [37], ACCAD, recorded at the Advanced Computing Center for the Arts and Design of the Ohio State University [38], and BMLrub, recorded at the BioMotion Lab [39]. The version annotated by BABEL, described below, contains 13,220 takes, more than 43 hours of motion from 346 subjects [40].

Size, however, does not imply relevance. Each source dataset was recorded for its own research purpose, and AMASS gathers them as they are, without selecting their content. Locomotion is therefore only a part of the collection: much of it shows people sitting, dancing, playing sports, handling objects or holding still poses, and even a single take can mix several of these, as the one of Figure 2.3 does. Using AMASS for a specific purpose requires selecting the relevant motion first, which is what the labels of BABEL make possible.

BABEL [40] is a semantic layer on top of AMASS: it describes with natural language what happens in its takes, with labels at two levels. A sequence label describes the take as a whole, while frame labels divide it into segments, each with a start time, an end time and a description of the action performed in that interval. Every take of AMASS has a sequence label, while frame labels are available for 10,892 takes, more than 37 hours of motion. Every label carries the original free-text

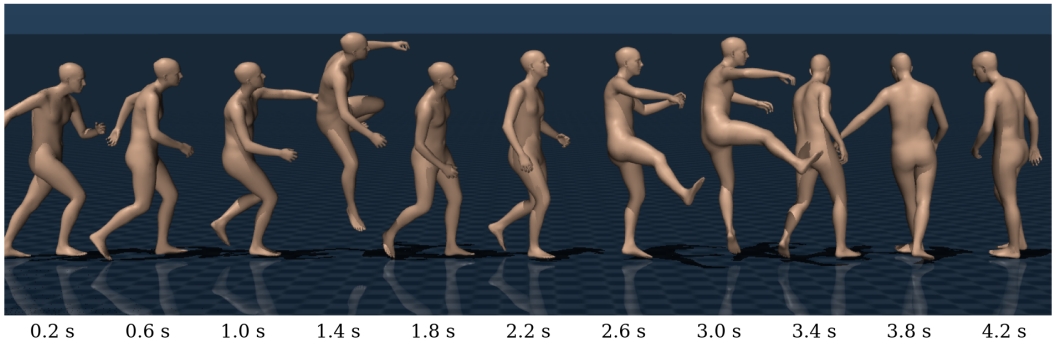


Figure 2.3. Frames of an AMASS take, rendered with the SMPL-X body model. Within a few seconds the performer runs, jumps, walks and kicks.

description written by the annotator, a cleaned version of it, and one or more action categories taken from a fixed set of more than 250. Frame labels may overlap, so a single instant can be covered by several labels. BABEL also provides a split of the takes into training, validation and test sets. Figure 2.4 shows how a take is annotated.

2.3.4 Robot Motion Datasets

LAFAN1 and AMASS contain human motion, which has to be retargeted before a robot can use it. Several works on humanoid control, however, publish the motions they train on already retargeted to the robot they use, and the Unitree G1 is by far the most common one. Two of these releases provide the reference motions of some of the skills used in the experiments of this thesis (Section 6.2.3).

ASAP [41] is a framework for learning agile whole-body skills in simulation and transferring them to a real humanoid robot. Its policies first learn in simulation to track human motions retargeted to the G1, and are then adapted with data collected on the real robot, through a model that learns the difference between the simulated and the real dynamics. Together with the framework, ASAP publishes the motions it uses, retargeted to the G1 with the wrist joints fixed, as sequences of root poses and 23 joint angles recorded at 30 Hz. They include highly dynamic motions, such as jumps.

PAiD [42] is a framework for learning football skills on the G1, and publishes the kicking motions on which its policies are trained: thirteen kicks performed by humans and retargeted to the G1, each labeled with the leg that strikes the ball.

2.4 Motion Control Policies

2.4.1 Reward-Based Policy Learning

A policy for an individual behavior can be obtained by specifying what should be achieved through a reward and optimizing the policy for it. Locomotion rewards, for example, may combine commanded velocity tracking, upright posture, energy use, smoothness and penalties for unsafe contacts. This approach is attractive because it

```
{
  "1833": {
    "babel_sid": 1833,
    "url": "https://babel-renders.s3.eu-central-1.amazonaws.com/001833.mp4",
    "feat_p": "BMLmovi/BMLmovi/Subject_55_F_MoSh/Subject_55_F_21_poses.npz",
    "dur": 8.3,
    "seq_ann": {
      "babel_lid": "8b112ce1-2546-4675-9efa-27db9fcfee29",
      "anntr_id": "c24c8ba1-f43d-48db-9f76-165d8264bda1",
      "mul_act": true,
      "labels": [
        {
          "raw_label": "walk back and forth",
          "proc_label": "walk back and forth",
          "seg_id": "3ca74412-f765-445b-b51b-6c2e72316010",
          "act_cat": ["walk", "forward movement", "backwards movement"]
        }
      ]
    },
    "frame_ann": {
      "babel_lid": "aabf3b2d-c8c7-47ac-aa75-4621d262434e",
      "anntr_id": "c0974846-d198-425d-846a-ce6808343841",
      "mul_act": true,
      "labels": [
        {
          "raw_label": "standing",
          "proc_label": "stand",
          "seg_id": "841150c3-030d-4369-a2e1-cf460f4a988a",
          "act_cat": ["stand"],
          "start_t": 0,
          "end_t": 0.341
        },
        {
          "raw_label": "pacing",
          "proc_label": "pace",
          "seg_id": "59af352a-0f5e-446e-9690-18e5d1d9ebaf",
          "act_cat": ["walk"],
          "start_t": 0.341,
          "end_t": 7.862
        }
      ]
    }
  }
}
```


Figure 2.4. The BABEL annotation of one AMASS take, as given in the documentation of the dataset [40]. `feat_p` points to the AMASS take and `dur` is its duration in seconds. `seq_ann` holds the sequence label, which describes the take in its entirety, and `frame_ann` the frame labels, each with its interval `start_t`–`end_t`. Every label carries the description written by the annotator (`raw_label`), its cleaned version (`proc_label`) and its action categories (`act_cat`).

does not prescribe a motion in advance. The policy is free to discover any behavior that solves the task, and large-scale simulation has made this approach effective, for example for locomotion on difficult terrain [2, 6].

The same freedom becomes a liability when the intended behavior is dynamic and specific: writing a reward that encodes exactly the intended motion is challenging, its terms can pull in opposite directions and small changes of their weights can change the learned behavior drastically. The policy also optimizes based on the reward and not the intention behind it, so a reward that looks correct is usually exploited: contact approximations, actuator limits and integration artifacts offer cheap return for motions that no physical robot could reproduce. Exploration is a further obstacle, since success may require a long, coordinated sequence of actions before any reward is observed at all, and such a sequence is unlikely to be discovered by chance.

Reward shaping can teach a useful gait, but a jump or a kick are usually easier to reproduce when a reference motion supplies the missing structure.

2.4.2 Motion Imitation and Trajectory Tracking

Motion imitation replaces the task objective with a time-indexed reference motion: rather than stating what should be achieved and leaving the rest of the trajectory free, the reward states what the motion should look like at each instant. The reference is a sequence of poses, usually a kinematic trajectory retargeted from motion capture (Section 2.3), and the policy is asked to be, at time t, in the pose the reference prescribes for that time. Its observation is its own physical state together with the reference state it should currently occupy, commonly extended with a short window of future reference frames so that it can anticipate what is coming rather than react to it. The reward penalizes differences in quantities such as joint, end-effector and root positions and velocities. Two properties follow from this construction. The signal is dense, since every step of the episode is scored against a known target, which removes most of the exploration problem that a sparse task reward creates. And the reference stays purely kinematic: it is the policy that has to realize it under the simulated dynamics, contacts and actuator limits, so what is learned is a controller for the motion rather than a playback of it.

DeepMimic is the work that made this idea practical [43]. Its setup is deliberately narrow: one motion capture clip, one policy trained to reproduce it, and nothing learned across clips. The policy observes the state of the character together with a phase variable that says how far into the clip it currently is, so that a pose encountered at two different moments of the motion is treated as two different situations. The phase variable is the time signal that a policy with a deadline needs (Section 2.1.3), and a tracking policy can itself be read as a goal-conditioned policy whose goal is the upcoming reference frame and whose deadline is the next control step. Its actions are target angles for joint-level proportional-derivative controllers rather than torques, which leaves fast joint corrections to those controllers. The reward is a weighted sum of terms comparing the character with the clip at the same phase, in joint angles, joint velocities, end-effector positions and center of mass, and it is granted at every step, so the policy is told how well it is doing throughout the motion instead of only at its end. Two training techniques complete the recipe

and are now standard in motion tracking: reference-state initialization and early termination. Early termination ends an episode as soon as tracking has failed badly enough that the remaining steps would teach nothing.

Reference-state initialization (RSI) decides where an episode begins [43]. If every training episode starts at the first frame of a clip, early states are visited often and later states are reached only after the policy has already tracked the preceding motion successfully. Learning then progresses slowly and failure near the beginning can prevent the policy from seeing the rest of the clip. RSI instead samples a reference time and initializes the character from the corresponding pose and velocity, so that the policy receives direct experience throughout the motion from the start of training.

RSI is more than a speed optimization. It trains the policy to start from a distribution of dynamic states along the reference, including states with nonzero momentum and active contacts. This makes a tracking policy resumable from different phases. It does not, however, make the policy valid from an arbitrary state. The sampled state is still close to its own reference, and the policy is not required to recover from the terminal distribution of an unrelated policy. That mismatch is the central obstacle in skill composition.

This recipe was enough to learn from single clips motions as dynamic as backflips, spin kicks and cartwheels. Two further properties matter for what follows. The learned controller is not a playback: because training exposes it to states the recording never contains, it also learns how to return to the clip after a push or a bad landing. And the imitation reward can be combined with a task reward when the motion also has to achieve something, such as hitting a target, in which case the clip supplies the manner of the motion and the task term its goal.

Later work broadened the relation between data and control. Adversarial Motion Priors learn a style reward from motion data instead of manually specifying a frame-by-frame tracking reward [44]. Adversarial Skill Embeddings learn a reusable latent skill space from a larger motion collection [45]. Perpetual Humanoid Control and PULSE pursue general controllers and motion representations capable of following diverse human motions rather than one narrow clip [46, 47]. These works show that one controller can cover many trajectories when it is conditioned on the desired motion.

A policy of this kind is called a universal motion tracker. The motion it follows is no longer fixed by training but given as an input: the observation contains the upcoming frames of the reference, usually expressed relative to the current state of the robot, in place of the phase variable of DeepMimic, which only identifies a moment of one particular clip. The price of this generality is that a single policy has to share its capacity among all the motions of the dataset, and the most dynamic ones can be tracked less accurately than by a policy dedicated to them, which is why training often gives more weight to the clips that the policy still fails to follow. A universal tracker separates the decision of how the robot should move, left to whatever produces the reference, from the execution of that motion, and it can therefore follow references that were never recorded. The works that develop this idea are reviewed in Section 3.3.2.

Tracking also changes the status of a motion dataset. Kinematic motion capture may contain foot sliding, penetration, missing scene objects or configurations that a

given robot cannot realize exactly, since nothing in the recording was constrained by that robot's dynamics, contacts or actuator limits. What a tracker produces is a different kind of data: a state sequence that the simulated robot actually executed, together with the actions and contacts that produced it. A clip that has been tracked successfully is therefore evidence of physical feasibility on that robot, and not only of visual plausibility.

2.4.3 Teacher-Student Distillation and Dataset Aggregation

Distillation is a family of techniques for transferring the behavior of a trained controller into a second policy. The trained controller acts as the teacher and the second policy as the student, which is trained by supervised learning on the teacher's decisions instead of by reward. The simplest of these techniques is behavior cloning, in which the student minimizes a supervised loss on observation-action pairs collected from the teacher,

$$L_{\mathrm{BC}}(\theta) = \mathbb{E}_{(\mathbf{o},\mathbf{a}^*)\sim\mathcal{D}}\left[\ell(\pi_\theta(\mathbf{o}), \mathbf{a}^*)\right], \tag{2.44}$$

where $\mathbf{a}^*$ is the teacher action, $\pi_\theta(\mathbf{o})$ the action the student produces for the observation $\mathbf{o}$ and ℓ a regression loss, typically a squared error, or a negative log-likelihood when the student is stochastic. Distillation can be used to hide some of the student's inputs, combine several teachers into one policy or compress a large model. It also separates two objectives: reinforcement learning can first solve the hard physics problem, after which supervised learning can teach a different architecture to reproduce the resulting behavior.

Behavior cloning alone assumes that the states the student meets once it is in control are the states the teacher visited while the dataset was being recorded. In closed-loop control this assumption breaks almost immediately. The student reproduces the teacher only approximately, so it acts slightly differently; a slightly different action leads to a slightly different next state, which the teacher never visited and on which the student was never trained, so its next action is worse still. The error compounds until the robot is in a situation the dataset says nothing about. This mismatch between the states seen during training and the states seen during execution is known as covariate shift.

DAgger is the technique that removes it, by collecting the training data exactly where the problem appears: along the student's own trajectories [48]. Training proceeds in rounds. The current student is executed and the states it visits are recorded; the teacher is then asked what it would have done in each of those states; these newly labeled states are added to the dataset, and the student is retrained on the union of every round collected so far (initial teacher demonstrations included). This aggregation, which gives the method its name, prevents the student from forgetting the states that earlier rounds had already taught it to handle. Wherever the student drifts, the states it drifts into enter the dataset with the teacher's correction attached, so the next round knows how to recover from them. The teacher is used as a source of labels for the student's own state distribution, rather than as a source of further demonstrations of its own behavior.

2.5 Latent Variable Models

A latent variable model describes each data point $\mathbf{x}$ through a hidden variable $\mathbf{z}$, called the latent, from which the data point can be reconstructed or generated. The Autoencoder learns to compress data into a latent code and to reconstruct it, but gives no guarantees about the structure of the latent space and defines no distribution from which new codes can be drawn. The Variational Autoencoder supplies that distribution and becomes a generative model ($\hat{\mathbf{x}}$ can be generated from $\mathbf{z}$). The Conditional Variational Autoencoder makes generation goal-conditioned.

2.5.1 Autoencoders

An Autoencoder consists of two networks [49]. The encoder $f_\phi : \mathbb{R}^{n_x} \to \mathbb{R}^{n_z}$ maps a data point to a variable in latent space $\mathbf{z} = f_\phi(\mathbf{x})$, and the decoder $g_\theta : \mathbb{R}^{n_z} \to \mathbb{R}^{n_x}$ maps the variable back to the data space, producing a reconstruction $\hat{\mathbf{x}} = g_\theta(\mathbf{z})$. The two are trained together to minimize a reconstruction loss over a dataset $\mathcal{D}$, most commonly the mean squared error (MSE)

$$L_{\text{AE}}(\phi, \theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \left[\left\| \mathbf{x} - g_\theta(f_\phi(\mathbf{x})) \right\|^2 \right]. \tag{2.45}$$

Choosing $n_z \ll n_x$ makes the latent a bottleneck: the networks have to discover a compact description of the data that retains what is needed to reconstruct it.

The Autoencoder is not a generative model. Nothing in Equation (2.45) tells how the latent space is structured: the encoder only has to assign each data point a variable from which the decoder can recover it. These variables therefore occupy scattered and irregular regions of $\mathbb{R}^{n_z}$, and there is no distribution over the latent space from which new ones could be drawn.

2.5.2 Variational Autoencoders

The Variational Autoencoder (VAE) makes the autoencoder generative by fixing in advance how the latents are distributed [50]. This distribution is called the prior, $p(\mathbf{z})$, and it is usually the standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. If the decoder learns to turn any latent drawn from the prior into a realistic data point, generating new data is simple: draw $\mathbf{z}$ from $p(\mathbf{z})$ and decode it.

In a VAE, the decoder is probabilistic. Instead of a single output, it defines a distribution $p_\theta(\mathbf{x} \mid \mathbf{z})$, the probability of the data point $\mathbf{x}$ given the latent $\mathbf{z}$. In practice this is a Gaussian centered on the output of the network, so the decoder still produces a reconstruction $\hat{\mathbf{x}}$, and $p_\theta(\mathbf{x} \mid \mathbf{z})$ is high when $\hat{\mathbf{x}}$ is close to $\mathbf{x}$.

The encoder is probabilistic too. A given data point could have been produced by many latents, some more likely than others. The distribution of these latents, $p_\theta(\mathbf{z} \mid \mathbf{x})$, is called the posterior. Computing it exactly would require evaluating the decoder on every possible latent, which is not feasible. The encoder therefore learns an approximation of it, $q_\phi(\mathbf{z} \mid \mathbf{x})$. Instead of a single latent, it outputs the mean $\boldsymbol{\mu}_\phi(\mathbf{x})$ and the standard deviation $\boldsymbol{\sigma}_\phi(\mathbf{x})$ of a Gaussian,

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_\phi(\mathbf{x}), \operatorname{diag}(\boldsymbol{\sigma}^2_\phi(\mathbf{x}))), \tag{2.46}$$

and during training a latent is sampled from this Gaussian and passed to the decoder.

The VAE is trained by minimizing

$$L_{\mathrm{VAE}}(\phi, \theta) = -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] + D_{\mathrm{KL}} \left(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\right). \tag{2.47}$$

The first term is the reconstruction loss: the latent sampled from the encoder must decode back to $\mathbf{x}$. With a Gaussian decoder, it is the squared error of Equation (2.45), up to a scale and a constant. The second term is the Kullback-Leibler (KL) divergence between the encoder distribution and the prior, which measures how different the two distributions are. It pushes the encoder distribution of every data point toward the prior, so that the latents of all data points end up in the region covered by the prior. This is what gives the latent space its structure: a latent drawn from the prior at generation time falls where the decoder has been trained. For a Gaussian encoder and a standard normal prior, the KL divergence has a closed form,

$$D_{\mathrm{KL}} \left(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(\mathbf{0}, \mathbf{I})\right) = \frac{1}{2} \sum_{j=1}^{n_z} \left(\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1\right). \tag{2.48}$$

The loss in Equation (2.47) has a precise meaning. The *evidence* is the probability that the model assigns to a data point once every possible latent has been taken into account, $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})\, d\mathbf{z}$. Maximizing it is the natural training objective, but the integral cannot be computed. Its logarithm, however, decomposes as

$$\log p_\theta(\mathbf{x}) = -L_{\mathrm{VAE}}(\phi, \theta) + D_{\mathrm{KL}} \left(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p_\theta(\mathbf{z} \mid \mathbf{x})\right), \tag{2.49}$$

where the last term measures how far the encoder is from the true posterior. Since a KL divergence is never negative, $-L_{\mathrm{VAE}}$ is a lower bound on the log-evidence, called the *evidence lower bound* (ELBO), and the bound is tight when the encoder matches the posterior. Minimizing L_{VAE} therefore raises a guaranteed lower bound on the likelihood of the training data, and at the same time brings the encoder closer to the posterior [50].

The two terms work against each other. The reconstruction loss wants each data point to have its own precise latent, while the KL divergence wants every encoder distribution to equal the prior. A weight β on the KL term sets the balance: $\beta = 1$ gives the standard VAE, and larger values give a more regular latent space but less accurate reconstructions [51]. If the KL term wins completely, the encoder outputs the prior for every data point, the latent carries no information, and the decoder learns to ignore it. This is called posterior collapse [52]. A common remedy is to start training with $\beta = 0$ and increase it gradually, so that the decoder learns to use the latent before the KL term takes effect.

Sampling is not differentiable, so gradients cannot flow from the decoder back to the encoder through a sampled latent. The reparameterization trick solves this by writing the sample as

$$\mathbf{z} = \boldsymbol{\mu}_\phi(\mathbf{x}) + \boldsymbol{\sigma}_\phi(\mathbf{x}) \odot \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{2.50}$$

where the noise $\boldsymbol{\epsilon}$ is drawn independently of the network. The randomness is now in $\boldsymbol{\epsilon}$, and the latent is a differentiable function of the encoder outputs [50].

A trained VAE generates data, but it offers no control over which data it generates. Sampling the prior yields a data point from the whole distribution the model was trained on, with no means of asking for a particular class of objects.

2.5.3 Conditional Variational Autoencoders

Many prediction problems are one-to-many: the same input admits several correct outputs. A deterministic regressor trained with a squared error learns the conditional mean of the outputs, which is appropriate when they vary around a single value and inappropriate when they form distinct modes, since the mean of two valid outputs might not be valid by itself. For example, a character that has to walk from one standing pose to another can take the first step with either the left or the right foot, both motions are valid. A regressor trained on both learns their average. What is needed is a model that, given the input, produces one of the valid outputs rather than their average.

The conditional variational autoencoder (CVAE) extends the VAE with an observed condition $\mathbf{c}$ supplied to all its components [53]. Generation now proceeds by drawing a latent from a prior $p(\mathbf{z} \mid \mathbf{c})$ and decoding it with $p_\theta(\mathbf{x} \mid \mathbf{z}, \mathbf{c})$, and the encoder, which sees both the data point and the condition, approximates the posterior with $q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{c})$. The loss is the same as Equation (2.47), with the condition added to every distribution and the weight β on the KL term. On each pair $(\mathbf{x}, \mathbf{c})$ of the dataset, training minimizes

$$L_{\text{CVAE}} = -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{c})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}, \mathbf{c})\right] + \beta D_{\text{KL}}\left(q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{c}) \,\|\, p(\mathbf{z} \mid \mathbf{c})\right). \tag{2.51}$$

The condition carries what the output is required to be, and the latent carries whatever else varies among the outputs compatible with it, such as which of several modes is realized. Because the encoder sees the target output during training, it can encode in the latent the mode that explains it. At inference the encoder is discarded, a latent is drawn from the prior and decoded together with the condition, and different latents yield different valid outputs for the same condition.

The prior $p(\mathbf{z} \mid \mathbf{c})$ in Equation (2.51) can be chosen in two ways. The simplest is to keep the standard normal distribution of the VAE for every condition, $p(\mathbf{z} \mid \mathbf{c}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. The alternative is a learned prior $p_\psi(\mathbf{z} \mid \mathbf{c})$: a separate network that takes the condition and outputs the mean and standard deviation of a Gaussian [53]. Since the KL term in Equation (2.51) depends on ψ, the prior is trained jointly with the encoder and the decoder. Minimizing the KL term with respect to ψ fits $p_\psi(\mathbf{z} \mid \mathbf{c})$ to the approximate posteriors $q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{c})$ of the data points that share the condition $\mathbf{c}$. The prior thus learns to predict, from the condition alone, the region of the latent space that the encoder assigns to that condition. A learned prior is preferable when the distribution of valid outputs differs substantially across conditions. A fixed prior forces the approximate posteriors of all conditions toward the same distribution, and this constraint becomes harder to satisfy, without degrading the reconstruction, as the differences between conditions grow.

Posterior collapse takes a specific form in the conditional case. The decoder now has a second source of information, and when the condition alone nearly determines the output, it can ignore the latent without any loss in reconstruction. Collapse then reflects the data as much as the model: if every condition in the dataset is paired with essentially one output, there is no multimodality for the latent to encode. When the output is generated as a sequence, one latent is commonly held fixed for the whole sequence, so that it represents a single, coherent choice; sampling an independent latent at every step lets the choice itself change along the sequence.

2.6 Diffusion Models

2.6.1 Denoising Diffusion

Diffusion models generate data by learning to reverse a gradual corruption process [54, 55]. The corruption, or forward process, starts from a data point $\mathbf{x}_0$ and adds a small amount of Gaussian noise in each of K steps,

$$q(\mathbf{x}_k \mid \mathbf{x}_{k-1}) = \mathcal{N}(\sqrt{\alpha_k}\,\mathbf{x}_{k-1}, (1-\alpha_k)\mathbf{I}), \qquad k = 1, \ldots, K, \tag{2.52}$$

where the noise schedule $\alpha_1, \ldots, \alpha_K \in (0,1)$ is fixed in advance. The index k counts noise levels. Each step shrinks the signal slightly and adds noise, so the data are progressively destroyed, and with a suitable schedule $\mathbf{x}_K$ is indistinguishable from a sample of $\mathcal{N}(\mathbf{0}, \mathbf{I})$. A composition of Gaussian steps is Gaussian, so any intermediate level can be reached in a single step: with $\bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i$,

$$\mathbf{x}_k = \sqrt{\bar{\alpha}_k}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_k}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{2.53}$$

Generation runs the process backwards. When the forward steps are small, each reverse step is itself approximately Gaussian, and a network is trained to approximate it with $p_\theta(\mathbf{x}_{k-1} \mid \mathbf{x}_k) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_k, k), \sigma_k^2 \mathbf{I})$, whose variance σ_k^2 is fixed by the schedule and whose mean is learned. Sampling starts from pure noise $\mathbf{x}_K \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and applies the learned reverse step K times, removing a little noise at each, until a sample $\mathbf{x}_0$ is obtained.

The model is a latent variable model in the sense of Section 2.5.2, with the noisy versions $\mathbf{x}_1, \ldots, \mathbf{x}_K$ as latents and the forward process as a fixed encoder that is not learned. Its evidence $p_\theta(\mathbf{x}_0)$ is again intractable, and it is trained by maximizing the same kind of evidence lower bound as the VAE (Equation (2.47)). For Gaussian steps the bound reduces to a remarkably simple objective. Ho et al. parameterize the reverse mean through a network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k)$ that predicts the noise contained in $\mathbf{x}_k$,

$$\boldsymbol{\mu}_\theta(\mathbf{x}_k, k) = \frac{1}{\sqrt{\alpha_k}} \left(\mathbf{x}_k - \frac{1-\alpha_k}{\sqrt{1-\bar{\alpha}_k}}\, \boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k) \right), \tag{2.54}$$

and train it with a regression loss obtained from the bound by dropping the weight of each of its terms, a simplification that they found to improve the quality of the samples [55]:

$$L_{\mathrm{DM}}(\theta) = \mathbb{E}_{\mathbf{x}_0 \sim \mathcal{D},\, k \sim \mathcal{U}\{1,K\},\, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\big(\sqrt{\bar{\alpha}_k}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_k}\,\boldsymbol{\epsilon},\, k\big) \right\|^2 \right]. \tag{2.55}$$

A training step thus draws a data point, a noise level and a noise sample, corrupts the data point with Equation (2.53) and asks the network to recover the noise. Since $\mathbf{x}_0$ follows from $\mathbf{x}_k$ and $\boldsymbol{\epsilon}$ through Equation (2.53), predicting the noise is equivalent to predicting the clean sample, and some models, in particular for human motion, let the network output $\hat{\mathbf{x}}_0$ directly [56]. Predicting the noise is also equivalent, up to a scale, to estimating the score $\nabla_{\mathbf{x}_k} \log p_k(\mathbf{x}_k)$, where p_k is the density of the data noised to level k: the score is the direction in which this density increases, and $\boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k) \approx -\sqrt{1-\bar{\alpha}_k}\,\nabla_{\mathbf{x}_k} \log p_k(\mathbf{x}_k)$. This connects diffusion models to score-based generative modeling [57] and underlies the guidance methods of Section 2.6.2.

Two properties distinguish diffusion models from the models of Section 2.5. Each reverse step is a simple Gaussian, but their composition can represent a distribution with many separate modes: the early steps, taken at high noise, determine which mode a sample falls into, and the later ones refine its details, so that a sample commits to one mode instead of averaging over them. And there is no learned encoder and no divergence term to balance, so posterior collapse does not arise. The price is the cost of sampling, since a sample requires K evaluations of the network, commonly hundreds; deterministic samplers that skip levels reduce this to tens without retraining [58].

2.6.2 Conditioning and Guidance

A diffusion model trained as above, like the VAE, samples from the whole data distribution. It becomes conditional in the same way as the CVAE, by giving the denoiser an observed condition $\mathbf{c}$ as an additional input, $\boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k, \mathbf{c})$, and training it on pairs $(\mathbf{x}_0, \mathbf{c})$ with the loss of Equation (2.55). The reverse process then samples from $p(\mathbf{x}_0 \mid \mathbf{c})$.

Guidance methods act on sampling instead, to strengthen a condition or to add one. Because the denoiser estimates a score, a condition can be imposed by adding the gradient of a log-likelihood to it: by Bayes' rule, $\nabla_{\mathbf{x}_k} \log p_k(\mathbf{x}_k \mid \mathbf{c}) = \nabla_{\mathbf{x}_k} \log p_k(\mathbf{x}_k) + \nabla_{\mathbf{x}_k} \log p_k(\mathbf{c} \mid \mathbf{x}_k)$. Classifier guidance trains a separate classifier on noisy samples and adds the gradient of its log-likelihood, scaled by a weight, to the score estimated by the denoiser, which for a noise prediction means subtracting it, multiplied by $\sqrt{1-\bar{\alpha}_k}$ [59]. Classifier-free guidance dispenses with the classifier: a single network is trained both with and without the condition, by replacing $\mathbf{c}$ with a null token $\varnothing$ in a fraction of the training examples, and at sampling time the two predictions are combined as

$$\tilde{\boldsymbol{\epsilon}} = (1 + w)\, \boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k, \mathbf{c}) - w\, \boldsymbol{\epsilon}_\theta(\mathbf{x}_k, k, \varnothing), \tag{2.56}$$

where $w \geq 0$ sets the strength of the guidance [60]. Larger weights produce samples that agree more closely with the condition, at the cost of diversity. The same mechanism accommodates objectives that are not learned at all: the gradient of a differentiable cost, such as a penalty on the violation of a constraint, can be added at each reverse step to steer samples toward low cost, a technique used for planning with diffusion models [61].

Guidance is soft. It makes samples that satisfy the condition more likely, without making any of them satisfy it exactly.

Chapter 3

Related Work

The problem studied in this thesis lies where three lines of research meet. The first is sequential skill composition, which studies how skills obtained separately can be executed one after the other, and which is the setting of this thesis. The second is motion generation, which studies how to synthesize a motion that satisfies a set of constraints, such as given keyframes and a given duration. The third is motion tracking, which studies how to train a policy that makes a robot follow a reference motion. This chapter reviews the three lines in this order, and closes with a summary in Section 3.4.

3.1 Sequential Skill Composition

In the terms of Section 2.2.1, two skills can be executed one after the other when the state in which the first one terminates belongs to the initiation set of the second (Equation (2.7)). For a skill learned from experience the initiation set is not given and has to be estimated, and the states in it are not all equally good starting points: some are better than others for the skill that starts from them (Section 2.2.5). The methods reviewed in this section obtain the composition condition in different ways, and they are grouped here by the part of the composition on which they act. The skills can be built so that the condition holds by construction (Section 3.1.1), or adapted to each other after they have been trained (Section 3.1.2). Otherwise the skills are left as they are, and a policy can be placed between them (Section 3.1.3).

3.1.1 Composition by Construction

The question of when one controller can safely hand the system over to another was first posed in model-based control. Burridge et al. compose feedback controllers sequentially through their regions of attraction [62]: each controller is associated with the set of states from which it is guaranteed to reach its goal, and one controller can be followed by another whenever its goal lies inside the region of attraction of the next. A sequence of controllers is then built backward from the final goal, adding at each step a controller whose goal lies inside the region of attraction of a controller already in the sequence, so that the set of states from which the final goal can be reached grows with every controller added. LQR-trees build such a

sequence automatically: they grow a tree of trajectories toward the goal, stabilize each one with a time-varying linear quadratic regulator, and compute for each one a region of states from which the regulator is guaranteed to bring the system onto the trajectory, so that every state covered by the tree is guaranteed to reach the goal [4]. Funnel libraries apply the same idea to motion planning [5]. Around each motion primitive of a library they precompute a funnel, the set of states that the primitive is guaranteed to keep within known bounds over time, also in the presence of bounded disturbances, and at run time they chain the primitives whose funnels fit into each other, which makes it possible to plan in real time while keeping the guarantees. Skill chaining carries the same construction into reinforcement learning [63]. Options are created backward from the goal of a task: the first option learns to reach the goal, and each new option learns to reach the initiation set of an option already in the chain. Since there is no model from which to compute it, the initiation set of each option is learned as a classifier, from the states in which the option was started and whether it succeeded from them, and deep skill chaining extends the approach to continuous control with deep reinforcement learning [64].

In all these works the composition condition holds because the parts are built for each other. The region of attraction is the model-based counterpart of the initiation set, and under their modeling assumptions the model-based methods guarantee the composition, which learned methods do not. They need, however, an accurate model of the dynamics, and computing regions of attraction becomes intractable as the number of state variables grows and as contacts make the dynamics change abruptly, which is the case of a humanoid robot performing dynamic whole-body motions. Skill chaining replaces the model with estimates learned from experience, as Section 4.1 does to define the good set of a skill, but each option is built for the one that follows it, in a chain toward a single goal.

3.1.2 Adapting the Skills

Skills that were trained separately can be adapted to each other afterward. Adversarial skill chaining fine-tunes the skills of a long manipulation task in sequence [9]. Each skill is trained further from the states in which the previous one terminates, which enlarges its initiation set. SCaR refines the skills with constraints in both directions, bringing the terminal states of each skill toward the initial states of the next one, and the initial states of each skill toward the terminal states of the previous one [10]. Sequential Dexterity chains dexterous manipulation skills and learns, for each of them, a transition feasibility function, which estimates from a short window of recent observations how well the next skill performs from the current state, and uses it to fine-tune the previous skill toward states that the next one can handle [65]. Robust Policy Gating adapts the skills of a humanoid fighting robot in a different way [12]. Its experts are trained together, and during training control passes at random moments from one expert to another, which continues from the state left by the first: each expert therefore learns to start from the states the others leave, and to abandon its own motion at any moment. Once trained, the experts are frozen, and a gating network blends their actions to smooth the switches.

3.1.3 Transition and Setup Policies

The works closest to this thesis leave the skills unchanged and place a learned policy between two of them, which brings the robot from the state left by the first to a state from which the second can succeed. Lee et al. train one transition policy for each primitive skill of a hierarchical agent [13]. Rewarding the transition policy only when the next primitive eventually succeeds would give a sparse and delayed signal, so each primitive is paired with a proximity predictor, a network trained on transitions that did and did not lead to a success of the primitive, which estimates how close a state is to the initiation set of the primitive. The increase of this estimate along the trajectory of the transition policy is used as a dense reward. Byun and Perrault train one transition policy for each ordered pair of skills, with a reward given by a discriminator that tells the transitions of the transition policy apart from those of the next skill, so that the transition policy learns to move as the next skill does [14]. A separate Q-network for each pair decides at every step whether to keep the transition policy in control or to hand over to the next skill, and it is rewarded by whether the next skill then succeeds. Setup policies apply the same principle to a simulated biped that crosses difficult terrain with a set of pre-trained policies, a default walking policy and one policy for each kind of obstacle [15]. When an obstacle is detected, a setup policy trained for the policy of that obstacle takes control from the walking policy. It is rewarded with the value function of the target policy, which estimates how well the target policy performs from a state, weighted down where that estimate is likely to be over-optimistic, in states the target policy never visited during training, and the setup policy itself decides when to hand over. With setup policies, the biped crosses random sequences of obstacles that it almost never crosses when switching directly between the policies.

These works show that an intermediate policy can make a switch between frozen skills succeed where a direct switch fails. The transition policies of Lee et al. are one of the baselines against which the bridge is compared (Section 7.2).

3.2 Motion Generation

3.2.1 Kinematic Motion In-Betweening

Motion in-betweening is the problem of generating the frames of a motion between given keyframes. Motion graphs, introduced by Kovar et al., are an early data-driven solution: they connect the frames of a motion database where two clips are similar, and generate new motion by walking the graph and blending across the junctions [66]. Harvey et al. formulate in-betweening as learned transition generation [34]. Their recurrent model receives the past context of the motion, a target pose and the time left before the target has to be reached, and generates the frames in between. The time is supplied through a time-to-arrival embedding, added to the internal representation of the network so that the same pair of endpoints produces a different motion for a different duration, and noise is added to the target during training to make the transitions robust and varied. The same work introduced LAFAN1, the dataset of long and continuous takes that forms part of the corpus used in this thesis (Section 5.2).

Diffusion models have since become the main tool for generating human motion. The Motion Diffusion Model (MDM) generates a whole motion at once with a transformer that denoises all the frames together, and adds geometric losses on foot contacts, positions and velocities to make the samples physically plausible [56]. MDM can already complete a motion whose first and last frames are fixed, by inpainting at sampling time. CondMDI turns this into a flexible in-betweening model: it is trained with random masks over frames and joints, receives the known values and the mask as input, and therefore generates motions consistent with keyframes that can be placed at any time and can constrain any subset of the joints [67].

These works show that a target pose and a duration are natural conditions for generating a transition, and that a diffusion model can produce diverse transitions for the same keyframes instead of averaging them. They also show how a window between known frames can be filled by inpainting. Their output, however, is a sequence of kinematic poses, generated for an animated character that has no dynamics. Nothing guarantees that a robot can execute the motion, and we only know how the robot should move, not which commands produce that movement under contacts, disturbances and actuator limits.

3.2.2 Diffusion for Planning and Control

Diffusion models have also been applied to decision making, where they generate a whole trajectory of the agent instead of a single action, and the works differ in what the trajectory contains. Diffuser generates the states and the actions of a trajectory together [61]. The Decision Diffuser generates only the sequence of states, conditioned on the return or on the constraints to satisfy, and recovers the action that leads from each state to the next with a separate inverse dynamics model [68].

For humanoid robots, BeyondMimic combines diffusion with motion tracking [69]. Its first stage trains tracking policies that reproduce highly dynamic human motions on a real Unitree G1. Its second stage distills them into a single diffusion model that generates the next states and actions of the robot together. At deployment the generated trajectory is steered toward the task through guidance (Section 2.6.2), so the same model performs different tasks without being trained again.

These works show that a diffusion model can represent several valid ways to complete the same request, with the boundary conditions held fixed. They differ in what is diffused. Diffuser and BeyondMimic generate states and actions together, so the generated actions control the robot directly, while the Decision Diffuser generates only states and relies on another model to obtain the actions.

3.2.3 Physics-Based Conditional Control

Generated motion can be made executable by tracking it with a physics-based controller, and several systems are built on this split between a kinematic generator and a physical executor. DReCon generates the reference with motion matching, which selects at every moment the clip of a database that best fits the user's command, and tracks it with a policy trained with reinforcement learning, so that a simulated character responds to commands in real time [70]. PhysDiff addresses the gap between generation and execution inside the diffusion process: at some of

the denoising steps it projects the motion onto what a physics-based tracking policy actually achieves in simulation, which removes floating, foot sliding and ground penetration from the samples [71]. CLoSD closes the loop between the two parts [72]. Its diffusion planner generates a short motion conditioned on text and on target locations, a universal tracking policy executes it, and the planner then generates the next piece of motion starting from the state the simulated character actually reached, so that tracking errors do not accumulate.

The same split has reached real humanoid robots. TextOp drives a humanoid with text commands that can change during execution: an autoregressive motion diffusion model generates short kinematic trajectories from the current command, and a tracking policy executes them on the robot [73].

MaskedMimic takes the opposite route and trains one physics-based policy that acts directly from partial constraints [74]. Its first stage is a tracking policy that follows complete reference motions, trained with reinforcement learning. In the second stage, this policy becomes the teacher of a student that sees only part of the reference: the future frames are masked at random, so that the student learns to act from constraints on any subset of joints at any future time, as well as from text and objects of the scene when they are available. The student is a conditional variational autoencoder with a learned prior. During training the encoder sees the complete future motion and produces a latent as a correction to the prior, while at deployment the latent comes from the prior alone, and the student is trained with DAgger on the states it visits itself.

3.3 Motion Tracking

3.3.1 Tracking Single Motions

The tracking approach of Section 2.4.2 was introduced by DeepMimic, which trains one policy for each motion capture clip [43]. The approach has since been carried to real humanoid robots, where retargeting, domain randomization and careful tuning of rewards and actuators allow a policy trained in simulation to reproduce dynamic human motions on the hardware. BeyondMimic is a recent example on the Unitree G1: its tracking policies reproduce motions as dynamic as sprints, jumps and spin kicks on the real robot [69].

3.3.2 Universal Motion Trackers

A universal motion tracker is a single policy trained to follow any reference motion, including motions it never saw during training. Its observation contains the upcoming frames of the reference, and it is trained on a large motion dataset, so that following a new motion means changing the reference rather than training a new policy. In computer animation, Perpetual Humanoid Control (PHC) tracks nearly all of the AMASS dataset with a simulated humanoid [46]. It grows a set of primitive policies progressively, each trained on the motions that the previous ones fail to track, and combines them with a composer network; the same policy also learns to recover from falls and to return to the reference, so that it can run without being reset. PULSE distills PHC into a latent space of motion, trained like a conditional VAE with a

learned prior, which can then be used by other policies as an action space for new tasks [47].

These works show that one policy can execute a very large set of motions, and that tracking is a convenient interface between a generator that decides how the robot should move and a controller that makes it move.

3.4 Summary and Positioning

The works reviewed in this chapter offer different ways to compose skills, and each of them obtains its result through a requirement on the skills or on the setting. Model-based composition states precisely when one controller can hand the system over to another, and under its assumptions it guarantees that the handover succeeds, but it needs a model of the dynamics and regions of attraction. Adapting the skills to each other removes the need for anything between them, at the cost of coupling every skill to its neighbors: a skill can no longer be trained or tested on its own, and adding a skill means training the others again. Transition and setup policies leave the skills unchanged, but each of them is trained for one incoming skill or for one pair, on executions of the skills it connects, so their number grows with the pool and every new skill requires new ones. The convention of starting and ending every skill with the robot at rest, finally, makes every pair of skills compatible, at the price of obtaining every skill under the same constraint and of stopping between them (Section 1.3).

The other lines of research provide tools rather than a way to compose skills. Motion generation produces transitions between given keyframes in a given time, and diffusion models in particular produce several of them for the same keyframes instead of their average, but the result is kinematic motion that a robot may not be able to execute. Motion tracking executes almost any reference motion under physics, but does not decide which motion to execute. The systems that join a kinematic generator to a physics-based tracker, and MaskedMimic, which acts directly from partial constraints on its future motion, show how the two can be combined. None of them, however, is designed to hand control over to another policy: their constraints describe the motion to be performed, nothing requires the robot to arrive within a given time to a target state, velocities included, from which a different policy continues.

This thesis studies a different approach, the bridge of Chapter 4, which takes elements from several of these lines. Like a transition policy, it is a learned policy placed between two skills that are left unchanged. Unlike a transition policy, it is a single policy for every pair of skills, and it is not trained on executions of the skills: it receives only the motion before the switch, the motion of the incoming skill after a target state and a deadline, a request with the shape of a motion in-betweening problem, and it is learned from a corpus of human motion. Its task is also stricter: it has to reach a complete target state within the deadline, while a transition policy aims at any state from which the incoming skill is likely to succeed. Whether such a bridge can be learned, and how it compares with the alternatives, is what the following chapters investigate. Chapter 5 presents three architectures, each building on a different part of the work reviewed here: a goal-conditioned policy trained

with reinforcement learning, a conditional VAE distilled from a tracking teacher as in MaskedMimic, and a diffusion planner that inpaints the transition between the two ends of the request, followed by a universal tracker that executes it. Chapter 7 compares the bridge with a direct switch and with transition policies trained for each incoming skill (Section 7.2).

Chapter 4

Problem Formulation

This chapter formalizes the problem addressed in this thesis and fixes the notation used in the rest of it, building on the options framework of Section 2.2. The setting is a humanoid robot whose behaviors are produced by separate learned policies, and the problem is how to pass the control of the robot from one of them to the next. Section 4.1 describes the skills to be composed and what it means for a state to be good for a skill; Section 4.2 introduces the bridge, an option placed between two skills, and states the problem of learning it; Section 4.3 describes the information the bridge receives; Section 4.4 states the assumptions and the scope of the problem, and Section 4.5 explains what makes it hard. The methods proposed to solve it are presented in Chapter 5.

4.1 Skills

Many tasks are too long or too varied to be solved by a single skill, but they can be solved by a sequence of skills, each of which takes over from the previous one. The skills available to the robot form a *pool*

$$\Omega = \{\omega_1, \dots, \omega_{n_\omega}\}, \qquad \omega_k(\mathbf{g}) = \langle \mathcal{I}_k(\mathbf{g}), \pi_k(\mathbf{a} \mid \boldsymbol{\sigma}, \mathbf{g}), \beta_k(\boldsymbol{\sigma} \mid \mathbf{g}), \psi_k \rangle, \qquad \mathbf{g} \in \mathcal{G}_k, \tag{4.1}$$

in which every skill is a goal-conditioned local-coordinate option (Section 2.2.2): its policy π_k and its termination condition β_k act on the local state $\boldsymbol{\sigma} = \psi_k(\mathbf{s} \mid \mathbf{s}_{\text{ref}})$, expressed in a frame attached to the robot, so that the skill behaves in the same way wherever it is executed, and $\mathcal{I}_k$ is its initiation set. The goal $\mathbf{g}$, taken from the goal space $\mathcal{G}_k$ of the skill, describes what the skill has to achieve, and is not necessarily a state to be reached. A walking skill, for example, has the goal of keeping a velocity, such as 1 m/s. For a kicking skill the goal is to kick the ball, so that it leaves for example with a speed above a threshold and in a given direction, while the position of the ball is part of the state. A tracking skill has the goal of following its reference trajectory. When the goal alone does not specify the task completely, it can be complemented by a set of parameters $\mathbf{x}$ that specify it further [75], for example the foot with which the ball is kicked; to keep the notation light, these parameters are considered part of $\mathbf{g}$ in the following. A skill with a single goal is the special case in which $\mathcal{G}_k$ has a single element. Each skill is trained on its own and with its own objective, and is then frozen: some are tracking policies, trained to follow

one reference trajectory (Section 2.4.2), while others are trained to maximize a task reward (Section 2.4).

A task is solved by a *plan*, a sequence of skills $\langle \omega_{k_1}(\mathbf{g}_1), \omega_{k_2}(\mathbf{g}_2), \ldots \rangle$ chosen by a high level. The abstract level of Section 2.2.3 is here the pool itself: an abstract action is the choice of a skill, and the high level chooses the skill that suits the situation. A robot moving on a flat floor, for example, can be controlled by a blind walking skill, and when it reaches uneven ground with obstacles, the high level can switch to a walking skill that perceives the terrain. Who chooses the plan, and how, is outside the scope of this thesis (Section 4.4). Each skill is trained for its own behavior and knows nothing about the skill that will follow it. A skill of the plan ends in one of two ways: it can terminate through its own termination condition, or it can be interrupted by the high level. Some skills can only end in the second way. Not terminating does not mean having no goal: a walking skill whose goal is to keep a velocity of 1 m/s pursues it at every step, but the goal is maintained rather than reached, so its termination condition is $\beta_k = 0$ in every state, and the skill carries on for as long as it is left in control. From the point of view of composition the two ways are equivalent, since in both the skill leaves the robot in a state from which the next skill may not be able to start; an interruption is simply a termination condition imposed from outside. The skill that ends is called the *source skill*, and the skill that has to start is called the *incoming skill*.

Two skills can always be executed one after the other only if the composition condition of Equation (2.7) holds, that is, if every state in which the source skill can end belongs to the initiation set of the incoming skill. Keeping the skills independent guarantees that each of them can be trained and tested on its own, but nothing guarantees that the source skill, even when it ends by itself, leaves the robot in a state of the initiation set of the incoming skill. In general, two independent skills are therefore only sometimes sequenceable (Equation (2.8)), and the switch succeeds only when the source skill happens to end in a suitable state.

The condition is also hard to meet by chance, because a skill depends on much more than the posture of the robot. The *posture* is given by the *pose* of the base, its position and orientation, together with the *configuration* $\mathbf{q}$, the vector of the joint angles. The posture alone does not tell how the robot is moving: the same posture can be held standing still or passed through in the middle of a run. Adding the linear and angular velocities of the base and the *generalized velocities* $\dot{\mathbf{q}}$ of the joints gives the *state* $\mathbf{s}$ of the robot. A single state, however, describes an instant and not the motion the robot is performing. The same state, for example, can be crossed by a walk that is speeding up, by a walk that is just starting and by a walk that is about to stop, and a skill that has to continue the motion behaves correctly only if it can tell them apart. For this reason, the motion on either side of a switch is described in Section 4.3 by a sequence of states rather than by a single one.

For a skill trained with reinforcement learning, the initiation set is not given explicitly, and it cannot be computed exactly either, since the states are continuous and infinitely many. It can, however, be estimated from experience, by starting the skill from many states and recording from which ones it succeeds; in [63] the initiation set of each skill is learned this way as a classifier, and [64] extends the approach to continuous control with deep reinforcement learning. As stated in Section 2.2.5, the states from which a skill can start are not all equally good, and

the quality of a state $\mathbf{s}$ for the skill ω_k started with goal $\mathbf{g}$ can be measured by how well the skill performs when it is started there. Let $J_k(\mathbf{s}, \mathbf{g})$ be the score of a run of $\omega_k(\mathbf{g})$ started in $\mathbf{s}$, with the goal kept fixed for the whole run, and lasting at most T_k control steps. For a skill trained with reinforcement learning the score can be its own return, as in Equation (2.36); in general it is any measure of whether the skill does what it is meant to do, for example whether it keeps tracking its reference without falling, or whether it completes its task. The runs are stochastic, so the quality is defined through the expected score, normalized between the lowest and the highest score that a run can obtain,

$$good(\mathbf{s} \mid \omega_k, \mathbf{g}) = \mathrm{clip}_{[0,1]}\left(\frac{\mathbb{E}[J_k(\mathbf{s}, \mathbf{g})] - J_k^{\min}}{J_k^{\max} - J_k^{\min}}\right). \tag{4.2}$$

This is the quality of a transition state defined in Equation (2.36), with the abstract-action-aware option of the next abstract action replaced by the skill itself. Given an acceptance level $\tau \in (0, 1]$, the states of quality at least τ form the *good set* of the skill for that goal,

$$\mathcal{I}_k^{\tau}(\mathbf{g}) = \{\mathbf{s} \in \mathcal{S} \mid good(\mathbf{s} \mid \omega_k, \mathbf{g}) \geq \tau\}, \tag{4.3}$$

which is the initiation set of the skill at acceptance level τ: the set of states from which the skill, started with goal $\mathbf{g}$, is expected to succeed, where what counts as success is fixed by the score and by τ.

4.2 The Bridge

The *2-abstract-action-aware option* of Section 2.2.5 deals with a problem close to this one, but starts from a composition that is already well defined. It is built on the abstract-action-aware option, which terminates in the block of the next abstract state, and this block is contained in the initiation set of the next option, so the composition condition of Equation (2.7) holds by construction. What the 2-abstract-action-aware option adds is the choice of the state of the block in which to terminate: knowing the next abstract action, it ends in a state from which the next option performs well enough, as a classification problem, or in the state from which it performs best, as an optimization problem. The skills of the pool offer no such starting point. They are not defined with respect to a common abstraction, and the state in which the source skill ends is usually not in the initiation set of the incoming skill, so the composition condition itself is missing, before any choice of a good state can be made. Obtaining both from the skills would require changing every skill: each skill would have to be trained again for every skill that may follow it, giving up to n_ω^2 options instead of n_ω, and the whole pool would have to be trained again whenever a skill is added. Each skill would also have to be designed with its successors in mind, and could no longer be trained and tested on its own. This thesis investigates whether the skills can instead be kept as they are, by placing a separate option between every two consecutive skills of a plan, the *bridge*, whose only purpose is to lead the robot from the state in which the source skill ends to a state that is good for the incoming skill. A plan $\langle \omega_{k_1}(\mathbf{g}_1), \omega_{k_2}(\mathbf{g}_2), \ldots, \omega_{k_n}(\mathbf{g}_n) \rangle$

would then be executed as

$$\langle \omega_{k_1}(\mathbf{g}_1),\ \omega_{\mathrm{b}}(\omega_{k_2}, \mathbf{g}_2),\ \omega_{k_2}(\mathbf{g}_2),\ \ldots,\ \omega_{\mathrm{b}}(\omega_{k_n}, \mathbf{g}_n),\ \omega_{k_n}(\mathbf{g}_n) \rangle, \tag{4.4}$$

where $\omega_{\mathrm{b}}(\omega_{k'}, \mathbf{g}')$ denotes the bridge toward the incoming skill $\omega_{k'}$ started with goal $\mathbf{g}'$.

The bridge is itself an option,

$$\omega_{\mathrm{b}}(\omega_{k'}, \mathbf{g}') = \langle \mathcal{I}_{\mathrm{b}},\ \pi_{\mathrm{b}}(\mathbf{a} \mid \mathbf{s}, \omega_{k'}, \mathbf{g}'),\ \beta_{\mathrm{b}}(\mathbf{s} \mid \omega_{k'}, \mathbf{g}') \rangle, \tag{4.5}$$

which differs from a skill in its three elements. Its initiation set $\mathcal{I}_{\mathrm{b}}$ must contain every state in which a skill of the pool can end. A skill that terminates by itself ends in one of its terminal states, but a skill can also be interrupted at any moment, so $\mathcal{I}_{\mathrm{b}}$ must contain every state that the skills of the pool can visit. Its termination condition is that of the 2-abstract-action-aware option, with the quality of Equation (4.2) measured for the incoming skill: as a classification problem, the bridge terminates as soon as the robot is in the good set of the incoming skill,

$$\beta_{\mathrm{b}}(\mathbf{s} \mid \omega_{k'}, \mathbf{g}') = \begin{cases} 1 & \text{if } \mathbf{s} \in \mathcal{I}^{\tau}_{k'}(\mathbf{g}'), \\ 0 & \text{otherwise,} \end{cases} \tag{4.6}$$

and, as an optimization problem, it terminates with probability $\mathit{good}(\mathbf{s} \mid \omega_{k'}, \mathbf{g}')$. Its policy, finally, depends on the incoming skill and its goal, and not on the source skill: everything that matters about the source skill is contained in the state in which it left the robot, together with the states that preceded it. Such a bridge would take the form of a 2-abstract-action-aware option: like it, it is a policy that acts knowing which option will run after it, the incoming skill, and it decides where to terminate according to the quality of the states for that option. Unlike it, the bridge has no abstract action of its own to execute, since its only task is the transition itself. A source skill ω_k followed by the bridge toward $\omega_{k'}$ would thus start where ω_k starts and end in a state that is good for $\omega_{k'}$, as the 2-abstract-action-aware option for the pair $(\omega_k, \omega_{k'})$ would, without training ω_k again. Two parts of the 2-abstract-action-aware option would become simpler as a consequence. First, a skill would no longer need to know which skill follows it, because in Equation (4.4) the option that follows a skill is always the bridge. Second, the terminal states of a skill would no longer need to lie in the initiation set of the next skill: the composition condition of Equation (2.7) would be required only between a skill and the bridge, where it holds by the definition of $\mathcal{I}_{\mathrm{b}}$, and between the bridge and the incoming skill, where it holds by Equation (4.6). The price is a delay: the incoming skill does not take over as soon as the source skill ends, because the robot first spends some time under the control of the bridge. A single bridge could serve every pair of skills, in the same way as a single abstract-action-aware option serves every abstract state in which its abstract action is possible: under an assumption of the same kind as the homogeneity assumption of Section 2.2.4, which here requires that bridging toward a state is the same problem in local coordinates whatever skills are involved, one option would replace the n_ω^2 options of the pairs.

The bridge cannot take an arbitrary time to reach a suitable target state. The high level that requests the switch also gives a *deadline*, a maximum number of

control steps $h_{\max}$ for which the bridge may act, after which it has to give control to the incoming skill. The deadline is not part of the options framework, in which an option takes as long as it needs; it is added here as a bound on the duration of the bridge, and if the bridge has not reached a good state for the incoming skill when the deadline expires, the switch fails. The bridge does not have to use all the time it is given: it hands over as soon as it reaches its target, which can happen well before the deadline.

The termination of Equation (4.6) cannot be used directly, because the good set of the incoming skill is known only through an estimate, and is not a convenient way to tell the bridge where it has to go. The bridge is therefore given a single *target state* B, taken from the good set of the incoming skill, which the bridge has to reach within a tolerance. The bridge is then a 2-abstract-action-aware option whose goal is one state of the good set of the next skill, rather than the whole set. The bridge terminates, and the incoming skill takes over, as soon as the robot is in B, which has to happen within the deadline. Ideally, B is the best state of the good set that the robot can reach within the deadline,

$$B \in \operatorname*{arg\,max}_{\mathbf{s} \in \mathcal{R}_{h_{\max}}(A)} \mathit{good}(\mathbf{s} \mid \omega_{k'}, \mathbf{g}'), \tag{4.7}$$

where A is the state in which the source skill ends and $\mathcal{R}_{h_{\max}}(A)$ the set of states that the robot can reach from A within $h_{\max}$ control steps. The best state of the good set in absolute terms may be out of reach, so the choice of B is a trade-off between its quality and how easily it can be reached from A in the time given. The problem therefore splits into two parts: *selecting* the target state, and *reaching* it within the deadline. This thesis focuses on the second part, and uses for the first a proxy, defined in Section 5.1.

4.3 Windows

Let $\mathbf{s}_t$ be the state of the robot at control step t, and $\mathbf{a}_t$ the action applied in it. A switch is described by two instants: the *switch* t_s, at which the source skill ends, by termination or by interruption, and the bridge takes control, and the *handover* $t_s + h$, at which the bridge terminates and the incoming skill takes control. The interval $[t_s, t_s + h]$ is the *transition window*, during which the bridge is in control, and its *duration* h is the number of actions that the bridge selects before handing over; $h = 0$ corresponds to a direct switch, in which the incoming skill accepts the state in which the source skill ended. The bridge has to reach the target state within the deadline $h_{\max}$ of Section 4.2, which therefore bounds the duration,

$$0 \leq h \leq h_{\max}, \tag{4.8}$$

so that the handover happens at step $t_s + h_{\max}$ at the latest. Figure 4.1 shows the three phases of a switch.

The state in which the source skill ends is the *start state*,

$$A = \mathbf{s}_{t_s}. \tag{4.9}$$

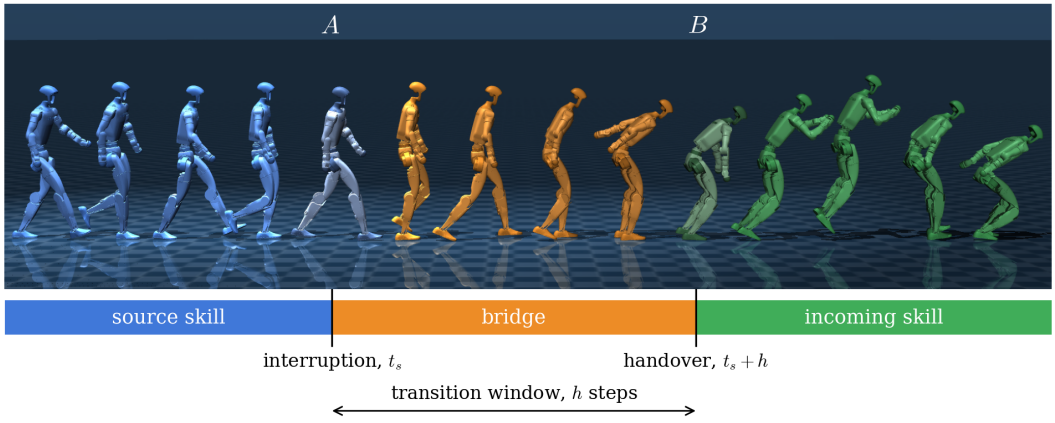


Figure 4.1. A switch with a bridge. The source skill (blue) runs until it ends at step t_s, in the start state A. The bridge (orange) then controls the robot for h steps in the transition window, and at the handover, at step $t_s + h$, no later than the deadline $t_s + h_{\max}$, the incoming skill (green) takes over from the state reached by the bridge.

As discussed in Section 4.1, A alone does not tell how the robot is moving. The source motion is therefore described by its last n states, which form the *history*

$$\mathcal{H}_A = (\mathbf{s}_{t_s-n+1}, \ldots, \mathbf{s}_{t_s}), \qquad \mathbf{s}_{t_s} = A. \tag{4.10}$$

The history ends with A: it contains the start state itself, preceded by the $n-1$ states through which the robot arrived there.

On the other side of the transition window, the target state B belongs to the good set of the incoming skill $\omega_{k'}$ started with goal $\mathbf{g}'$, chosen as discussed in Section 4.2. If the bridge hands over in B, at step $t_s + h$, then $\omega_{k'}(\mathbf{g}')$ takes the robot through the states $\mathbf{s}_{t_s+h}, \mathbf{s}_{t_s+h+1}, \ldots$, starting from $\mathbf{s}_{t_s+h} = B$. The first m of them form the *future*

$$\mathcal{F}_B = (\mathbf{s}_{t_s+h}, \ldots, \mathbf{s}_{t_s+h+m-1}), \qquad \mathbf{s}_{t_s+h} = B. \tag{4.11}$$

Like the history, the future is a sequence of states, and it starts with B: it contains the target state itself, followed by the $m-1$ states that the incoming skill goes through after it. The future tells how the incoming motion continues after B, and it is the description of the incoming skill and of its goal that the bridge receives in place of the skill itself: a walking skill whose goal is to walk faster, for example, gives a future in which the robot speeds up. It describes the nominal continuation from B: after a real handover, the incoming skill starts from the state reached by the bridge, and its motion coincides with $\mathcal{F}_B$ only as far as that state coincides with B.

The current state, the history and the future are expressed in local coordinates (Section 2.2.2), in a frame attached to the robot, for example with its origin on the floor below the pelvis and its first axis along the direction the robot faces, at the moment of the switch.

A switch is thus defined by the *request* $(\mathcal{H}_A, \mathcal{F}_B, h_{\max})$, which contains both ends of the switch: A as the last state of the history, and B as the first state of the future. At each step t of the transition window the bridge selects its action from the current

state and the request,

$$\mathbf{a}_t \sim \pi_{\mathrm{b}}(\mathbf{a} \mid \mathbf{s}_t,\ \mathcal{H}_A,\ \mathcal{F}_B,\ h_{\max}). \tag{4.12}$$

The request has the shape of a motion in-betweening problem [34]: two known pieces of motion, the history and the future, with a gap between them that has to be filled. The bridging problem is to find a policy of this form that, starting from A, brings the robot to B within the deadline,

$$\mathbf{s}_{t_s} = A, \qquad \left\| f(\mathbf{s}_{t_s+h}) - f(B) \right\| \leq \epsilon, \quad \text{for some } h \leq h_{\max}, \tag{4.13}$$

where f selects the quantities of the state that are compared, expressed in a common frame attached to the robot, as f_ψ in Section 2.2.2, and ϵ is the tolerance within which B counts as reached.

The same three parts, a history, a stretch of motion and a future, are used in Section 5.2.5 to cut training examples from recorded motion. A training example is a request whose answer is known: A and B are two states of the same recording, h is the number of steps between them, and the recorded motion is one way of bringing the robot from A to B in h steps, which realizes the request for any deadline $h_{\max} \geq h$.

4.4 Assumptions and Scope

The problem is defined entirely by what happens at the boundaries of the transition window. Everything the bridge needs to know about the source skill is contained in $\mathcal{H}_A$, and everything it needs to know about the incoming skill is contained in $\mathcal{F}_B$, so the bridge does not need to know which skills it connects, and the skills do not need to know that a bridge exists. The skills are frozen: they are never trained again to make a switch possible.

Which skills to execute, with which goals and when to interrupt a skill that does not end by itself are decided by the high level, and depend on how the skills are used. The skills can be chained in a fixed sequence, with all of these choices made in advance, or the choices can be made while the robot runs, by a high-level controller that dynamically reacts to the environment. The formulation is the same in both cases, and how the choices are made is outside the scope of this thesis. The selection of the target state of Equation (4.7), instead, is part of the problem, since it requires knowing the good set of the incoming skill, which the high level is not expected to know.

Not every request can be realized. The good set of the incoming skill may contain no state that can be reached from A within the deadline, for example because the incoming skill expects a speed that the robot cannot build up in that time. The bridge is therefore required to realize only the requests that the physics of the robot allows, and not every request it may receive.

4.5 Intrinsic Difficulties

Some difficulties come from the problem itself, and any method that addresses it has to face them, including the ones proposed in Chapter 5 to investigate it; other

difficulties come from learning the bridge from data, as those methods do, and are discussed in Section 5.2.7.

The first difficulty is that there is no single right answer. A humanoid robot can go from A to B within the deadline in many ways, taking a different number of steps, placing its feet differently or changing speed at a different moment. The relation between a request and a motion that realizes it is therefore not a function, which would give one motion for each request, but a distribution over many valid motions.

The second difficulty is precision under a deadline. The bridge must reach B with a small error before the deadline expires, which is hard for an underactuated system such as a humanoid robot. Its base is not driven directly by any actuator, so the robot can change the motion of its body only through the forces its feet exert on the ground, and only through the feet that are in contact. Whether the robot can reach B in time is therefore often decided well before the deadline, when the last steps are placed, and errors made early cannot always be corrected at the end. The target state also carries velocities, so the robot cannot reach it and wait there: the bridge has to arrive with the right motion, and not only at the right place.

The third difficulty is that the region the bridge actually has to reach is known only approximately. The incoming skill does not need exactly B, since its good set contains other states around it, but the shape of this set is not known in advance, and the estimate of it covers only the states from which the skill was tried. Reaching B within a tolerance is a proxy for reaching the good set, and the two can disagree: some errors are harmless, such as being slightly behind along the same motion with a matching lack of speed, while other errors of the same size take the robot out of the good set. The problem is more severe for skills that must interact with objects at a precise instant, for example with some trajectory tracking policies: a small error in the position of the robot relative to the object can prevent the contact the skill relies on.

The fourth difficulty is the size of the initiation set of the bridge. Since a skill can end in many states, and can be interrupted at any moment of its motion, the bridge must be able to start from every state that any skill of the pool can visit, together with every state that a target may require, and a single option has to cover all of them.

Chapter 5

Proposed Methods

This chapter presents several attempts at solving the problem of Chapter 4, in the order in which they were developed. All of them are instances of the bridge as defined there: a single policy shared by every pair of skills, which receives the history $\mathcal{H}_A$, the future $\mathcal{F}_B$ and the deadline, and has to bring the robot to the target state B before the deadline expires. Since they share this interface, the architectures can be compared on the same requests, and they differ only in how the bridge is structured and trained. All of them learn from the same motion corpus, described in Section 5.2. Section 5.3 then presents a goal-conditioned policy trained with reinforcement learning on a shaped reward, Section 5.4 a conditional variational autoencoder distilled from a tracking teacher, and Section 5.5 a two-stage pipeline made of a diffusion-based planner and a universal tracker. The limitations of each architecture motivate the next one.

5.1 Target Selection

As discussed in Section 4.2, a switch requires selecting the target state B and reaching it within the deadline. We study the second part and use a simple proxy for the selection. The proxy restricts the targets to the *on-policy* states of the incoming skill, the states that the robot actually goes through when the skill runs in nominal conditions.

The bridge does not receive B alone, but the future $\mathcal{F}_B$ that starts with it (Equation (4.11)), so the proxy is applied to sequences of states. An *entry sequence* of the incoming skill $\omega_{k'}$ is a sequence of m consecutive states recorded while $\omega_{k'}$ runs with its goal $\mathbf{g}'$. An entry sequence has the same form as a future, and its first state is the *entry point*. At a switch, the entry point is the target state B and the entry sequence is the future $\mathcal{F}_B$. The incoming skill continues from an entry point as it does in its own executions, so the quality of an entry point (Equation (4.2)) is close to its highest value by construction, and the entry point lies in the good set of the skill (Equation (4.3)).

An entry sequence contains only states, like the futures of the training windows (Section 5.2.5), but unlike them it is kept together with its goal $\mathbf{g}'$. The futures of the windows are cut from the motion corpus, where there are no skills and therefore no goals. An entry sequence comes from a skill, and it shows how that skill continues

only when it is run with the same goal $\mathbf{g}'$. The bridge receives only the entry sequence, and the goal is passed to the incoming skill.

The selection of Equation (4.7) therefore reduces to choosing the entry point that the bridge reaches best from A within the deadline. This thesis does not automate this choice. Several entry points of each skill are recorded and evaluated (Section 6.2.4). Figure 5.1 shows those of a kicking skill, in the execution they are taken from. One of them is picked by hand for each skill and used in all the tests (Section 7.3), so that the switches test different kinds of mismatch: in pose for the jump, whose first entry point is a deep crouch, and in velocity for the kick, whose fourth entry point, at step 124, is already in motion.

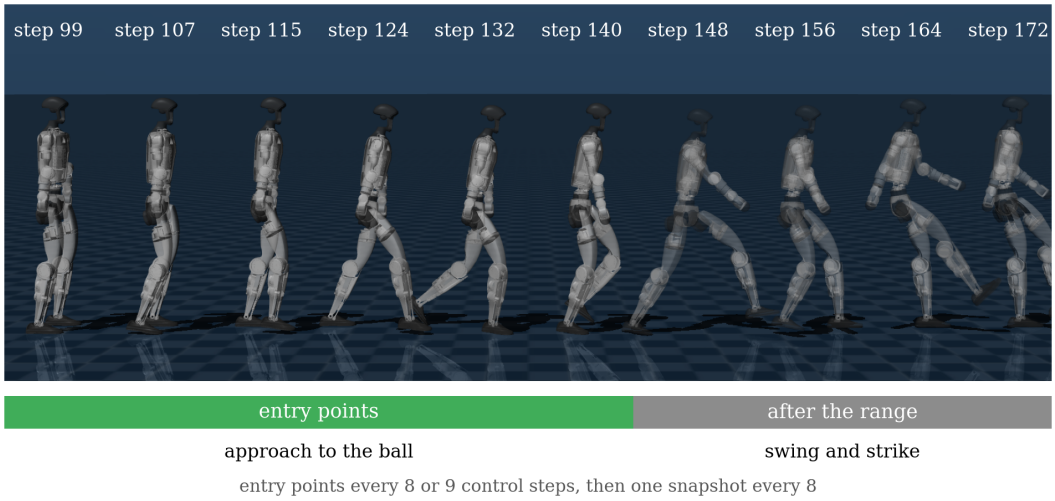


Figure 5.1. The entry points of a kicking skill, in the execution they are taken from. The six entry points, labeled with their control step, lie in the approach to the ball. The fainter states after them complete the execution with the swing and the strike.

5.2 Motion Corpus

The bridge is not trained on the skills of the pool, but on a separate corpus of kinematic trajectories (Section 2.3.1). The transition policies described in Section 3.1.3 are derived from the skills they connect: training them requires collecting rollouts of those skills, and every new skill requires training new networks. A bridge trained on a corpus only has to learn how the robot can move from one state to another, whichever skills produced those states. Whether the pool can then change without training the bridge again depends on how well the corpus covers the states of the skills, which is examined in Chapter 6.

No public dataset can be used as it is. Motion capture is recorded on human bodies, and most of it is motion that a switch will never require. Starting from two public datasets, LAFAN1 and AMASS with the BABEL labels (Section 2.3), this thesis develops a pipeline that turns them into training data for the bridge, in six stages:

1. *selection* keeps only the motion relevant to a switch (Section 5.2.1);

2. *retargeting and grounding* map it to the robot and repair its contact with the floor (Section 5.2.2);

3. *kinematic filtering* marks the frames a robot should not be asked to reproduce (Section 5.2.3);

4. *motion graphs* generate new trajectories that join unrelated motions (Section 5.2.4);

5. *windowing* cuts training examples in the form the bridge requires (Section 5.2.5);

6. *randomization and augmentation* vary the examples during training (Section 5.2.6).

The pipeline relies on existing work for the datasets, for the retargeting tool GMR [33], for the idea of the motion graph [66], for the format of the windows, taken from motion in-betweening [34], and for the standard randomizations of mirroring, observation noise and physics randomization. The rest is developed in this thesis.

Source	Split	Clips	Minutes	Frames	Valid frames
AMASS CMU	train	407	39.3	117,941	112,087 (95%)
AMASS CMU	val	148	14.5	43,485	42,962 (99%)
AMASS KIT	train	509	32.1	96,295	92,376 (96%)
AMASS KIT	val	162	10.7	32,149	31,279 (97%)
LAFAN1	train	21	79.4	238,072	218,276 (92%)
LAFAN1	val	4	16.5	49,507	48,230 (97%)
Stitched	train	10,000	513.3	1,540,000	1,540,000 (100%)
Stitched	val	2,000	102.7	308,000	308,000 (100%)

Table 5.1. Size of the corpus per source and split. Frames are counted at the control rate of 50 Hz, after retargeting and grounding, and the valid frames are those that pass the kinematic filter (Section 5.2.3). The stitched trajectories are generated with the motion graph (Section 5.2.4).

5.2.1 Selection

Both datasets contain a large share of motion that a switch never requires, so the first stage of the pipeline decides which part of them to keep. The criterion is the same for the two datasets. The bridge has to connect skills, so it should learn the motion that such connections are made of: walking, running, turning, stopping, crouching and jumping, together with the changes between them. Motions such as fighting and dancing, which both datasets contain, are rarely required by a switch, and they are also more aggressive and harder to execute. Keeping them would spend part of the capacity of the bridge on motion it does not need, and could make it less reliable on the motion it does need. The criterion is applied through the annotations that each dataset provides: the themes of the takes for LAFAN1, and the BABEL labels for AMASS.

LAFAN1. The takes are selected by theme. The takes of walking, running, sprinting and jumping are kept, together with the takes that mix several actions, which are the most valuable for the bridge because they contain the changes between motions. The takes about dancing, fighting, falling, moving on the ground or over obstacles are discarded. The takes of one of the five actors are held out for validation.

AMASS. AMASS has no themes, and a single take often mixes motions to keep with motions to discard (Figure 2.3), so selecting whole takes would either lose most of the locomotion or keep much of what the bridge should not learn. The selection is therefore made at the level of intervals, using the BABEL frame labels, with the procedure summarized in Algorithm 1. Only the CMU and KIT subsets of AMASS (Section 2.3.3) are used: in other subsets, such as BMLrub and ACCAD, many takes interact with objects or hold poses without the labels always saying so. A take is discarded entirely if any of its labels, or its file name, mentions an action that the bridge should not learn, such as sitting, fighting and dancing. An object or a pose usually shapes the motion around it, even where it is not labeled, which is why the whole take is discarded when the labels mention interacting with objects, carrying or throwing. In the remaining takes, the segments whose categories include locomotion (walking, running, jogging, turning, stepping) or lowered and propulsive motion (crouching, squatting, hopping, jumping, leaping), possibly together with neutral context such as standing, stopping or moving the arms, are kept; overlapping segments are merged and only the regions at least one second long are retained. Figure 5.2 shows the frame labels of a take on a timeline, and the region that the selection keeps. After retargeting and grounding, the selected motion amounts to about 1.6 hours, 54 minutes from CMU and 43 from KIT (Table 5.1). The training and validation splits provided by BABEL are kept.

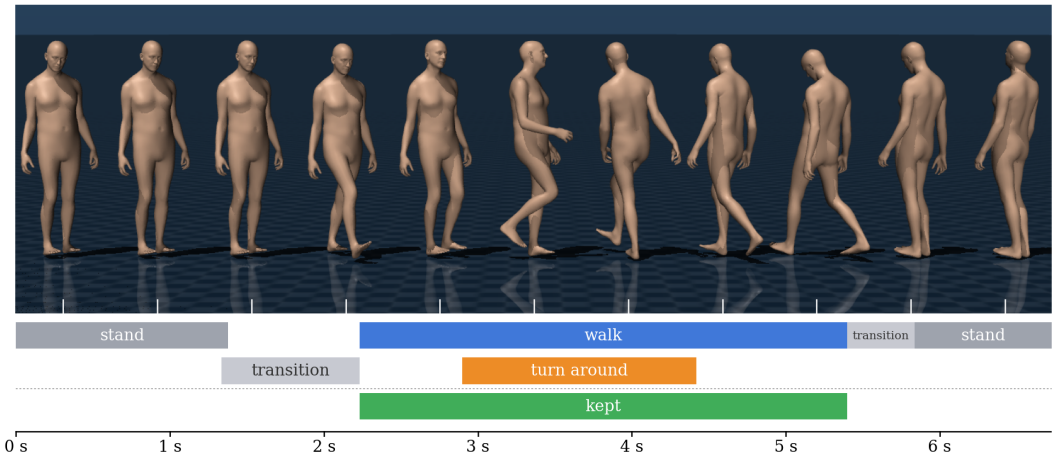


Figure 5.2. The BABEL frame labels of an AMASS take on a timeline. The walk and the turn overlap, and the transitions connect the walk to the standing phases at the two ends. The last row shows the region kept by Algorithm 1: the walk, including the turn, while the standing phases and the transitions are left out.

Algorithm 1 Semantic selection of AMASS motion with the BABEL labels.

Require: takes $\mathcal{T}$ with their BABEL labels; subsets $\mathcal{S}$; allowed categories $\mathcal{A}$; neutral context $\mathcal{C}$; denied terms $\mathcal{D}$; minimum length $\ell_{\min}$

$\mathcal{R} \leftarrow \emptyset$
for each take $T \in \mathcal{T}$ from a subset in $\mathcal{S}$ **do**
  if the file name or any label of T mentions a term in $\mathcal{D}$ **then**
    continue ▷ discard the whole take
  end if
  $kept \leftarrow \emptyset$, $cut \leftarrow \emptyset$
  for each label (c, t_0, t_1) of T, with categories c and interval $[t_0, t_1)$ **do**
    if $c \cap \mathcal{A} \neq \emptyset$ **and** $c \subseteq \mathcal{A} \cup \mathcal{C}$ **then**
      add $[t_0, t_1)$ to $kept$
    else if $c \not\subseteq \mathcal{C}$ **then**
      add $[t_0, t_1)$ to cut
    end if
  end for
  merge the overlapping intervals of $kept$
  remove from them every interval of cut
  add to $\mathcal{R}$ the remaining intervals at least $\ell_{\min}$ long
end for
return $\mathcal{R}$

5.2.2 Retargeting and Grounding

The selected motion of both datasets is retargeted to the robot with GMR (Section 2.3.1). The retargeting runs at 30 Hz, and we resample the result to the control rate of 50 Hz, interpolating positions and joint angles. The velocities of the root and of the joints are then computed by central finite differences, and the poses and velocities of all the bodies of the robot by forward kinematics, so that every frame contains a complete and consistent state of the robot.

GMR solves each frame on its own and does not know where the floor is, so a retargeted clip often hovers a few centimeters above the ground, or sinks below it, and the height can drift within the clip. For a kinematic animation this is a minor flaw, but for the bridge it is not. We therefore add a grounding step after the retargeting, in two parts. Each clip is first checked as a whole. The height of the lower sole is computed at every frame, a low quantile of it is taken as the height of the floor, and the clip is rejected if it sinks more than 5 cm below the floor or if its feet touch the ground in less than 20% of the frames. The accepted clips are then grounded frame by frame. A foot is considered a supporting foot when it moves slower than 0.25 m/s and is within 8 cm of the floor of the clip; at every frame with a supporting foot, the whole body is shifted vertically so that the lowest supporting foot touches the floor, while the frames without support, such as the flight phase of a run, take a shift interpolated from their neighbors. No foot is ever left below the floor. The shift is smoothed over 0.1 s, limited to 15 cm, and its time derivative is added to the vertical velocities, so that positions and velocities stay consistent. Figure 5.3 shows poses from the two datasets next to their retargeted and grounded

counterparts.

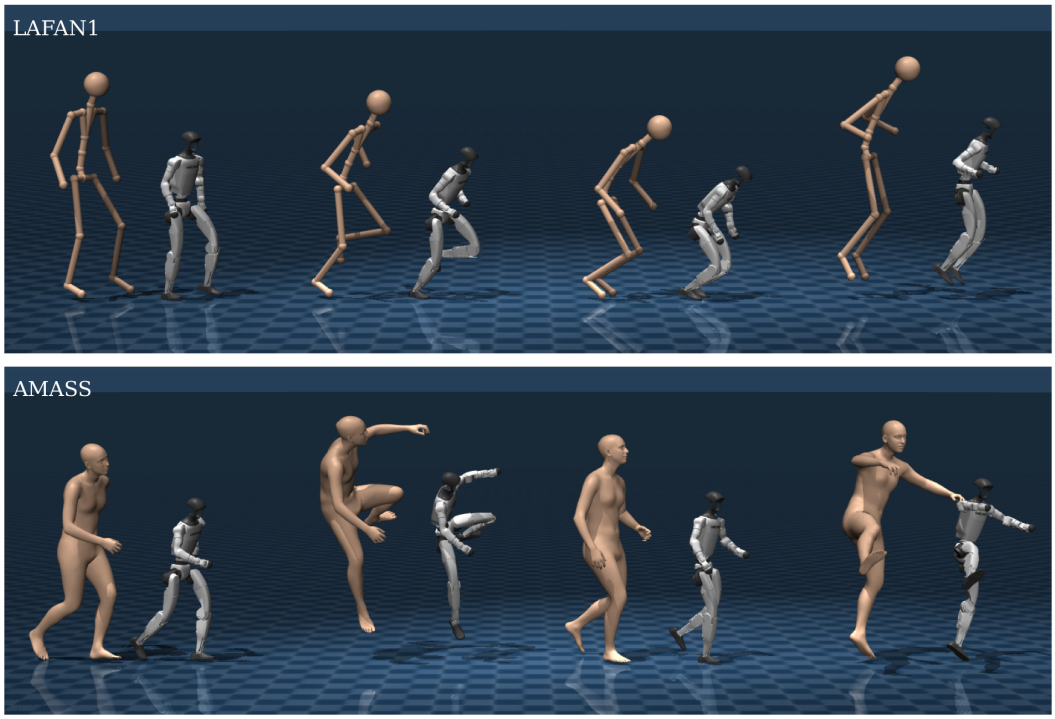


Figure 5.3. Retargeting and grounding. Each human pose, from LAFAN1 (top) and AMASS (bottom), is shown next to the corresponding pose of the robot after retargeting with GMR and grounding.

5.2.3 Kinematic Filtering

Grounding fixes the height of a clip, but not what the clip asks the robot to do. Retargeted human motion can still contain frames that a humanoid robot should not be asked to reproduce during a bridging attempt, either because the performer did something the bridge should not learn, or because the retargeting failed. A kinematic filter checks every frame of every clip, and marks as invalid the frames in which the root is tilted too far from upright, as when lying or getting up, or the pelvis is too low; the root moves, turns or accelerates too fast; a joint moves or accelerates too fast; a foot sinks below the floor; or both feet are off the ground for too long. A flight phase that is too long is removed, so a jump may be split into two pieces, while its crouch, push-off and landing remain in the corpus. Since a training example is a contiguous piece of a clip, no example can contain an invalid frame. The same filter is applied to both datasets.

The limits of the filter depend on the robot and on what it is able to achieve. A robot capable of highly dynamic motion can keep running strides, hops and deep crouches, which a less capable robot has to leave out, so each robot gets its own limits, and its corpus keeps only the motion that robot can execute.

5.2.4 Motion Graphs

Selection, grounding and filtering clean the recorded motion, but they cannot add what the recordings do not contain. Two gaps remain. The first is that every take is one continuous performance, so in every window cut from it the start state and the target state belong to the same motion, while at deployment they come from two different skills (Section 5.2.7). The second is that, paradoxically, some of the simplest states are rare: standing still, for example, appears mostly at the beginning and at the end of a take, so few windows start from a standing robot, and a bridge trained on this data would be able to perform complex transitions while failing at simple ones. To close both gaps, we augment the corpus with trajectories that were never recorded, generated with a motion graph [66]. In its original form, a motion graph connects the frames of different clips that look alike and produces new motion by walking the graph (Chapter 3). Here the idea is used for a narrower purpose: each generated *stitched trajectory* joins a piece of one clip to a piece of an unrelated clip, so that a window cut across the seam starts in one motion and ends in another.

Two frames are compared through a feature vector designed for this purpose. It does not depend on the heading of the robot or on its position on the floor, and contains the height of the pelvis and its tilt, the joint angles and velocities, the linear velocity of the root in the heading frame and its yaw rate, the positions of the feet relative to the pelvis and their velocities, and which feet are in contact with the floor. Each component is weighted so that the distances of the different quantities become comparable. The contact flags receive a weight much larger than the others, so that two frames with a different supporting foot are never considered alike, and a stitched trajectory never swaps the supporting foot at the seam.

Most of the corpus is walking, so transitions drawn at random would almost all join a walk to another walk. Transitions are therefore drawn without repetition, with probabilities that make every pair of action categories equally likely, such as a walk followed by standing or a run followed by a crouch, where the category of a frame is the BABEL category or the LAFAN1 theme of its clip. This also makes the rare states, such as standing still, appear on the two sides of a seam much more often than in the recordings. Each drawn transition gives one stitched trajectory, made of the 0.4 s of the first clip that lead to the seam and the 2.7 s of the second clip that follow it. The second piece is rotated about the vertical axis and translated on the floor so that its root starts where the first piece ends, facing the same direction, and the remaining difference in pelvis height, orientation and joint angles is faded out over the first 0.2 s after the seam with a cosine profile. Body poses and velocities are then computed again by forward kinematics and finite differences, as for the recorded clips, and a stitched trajectory in which the kinematic filter rejects any frame is discarded entirely. Figure 5.4 shows a stitched trajectory.

Windows (Section 5.2.5) are cut from a stitched trajectory only if they contain the seam, with the start state in the first piece and the target state in the second, so every one of them shows a transition between two motions that the recorded clips do not contain. The stitched trajectories are not recordings of a real performer, and the seam can contain small imperfections, which are absorbed when the motion is executed in simulation. They make up the largest part of the corpus: Table 5.1 counts 10,000 stitched trajectories for training, about 8.5 hours, against about 2.5

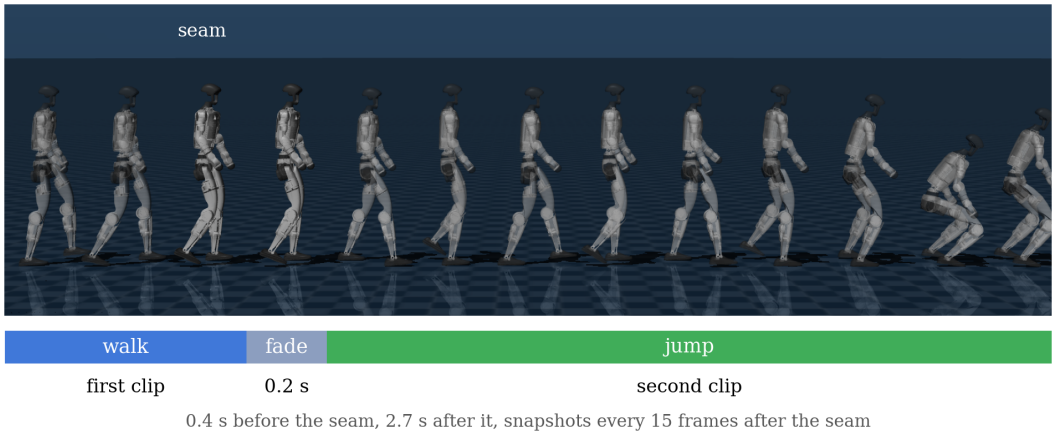


Figure 5.4. A stitched trajectory: 0.4 s of a walk, followed by 2.7 s of a jump clip of BABEL, in which the performer walks on and then crouches to take off. The two frames on either side of the seam are drawn stronger. The difference between them is faded out over the first 0.2 s of the second clip.

hours of recorded training motion.

5.2.5 Transition Windows

The clips, recorded and stitched, are cut into windows, the training examples of Section 4.3. The format follows motion in-betweening [34], which also cuts its training examples from continuous takes. A window contains a history, the motion that follows it, and a future. If the start state is frame r of a clip, $A = \mathbf{s}_r$, and h is the duration, the window is

$$W = (\underbrace{\mathbf{s}_{r-n+1}, \ldots, \mathbf{s}_r}_{\mathcal{H}_A},\ \mathbf{s}_{r+1}, \ldots, \mathbf{s}_{r+h-1},\ \underbrace{\mathbf{s}_{r+h}, \ldots, \mathbf{s}_{r+h+m-1}}_{\mathcal{F}_B}), \tag{5.1}$$

with target state $B = \mathbf{s}_{r+h}$. The notation is that of Section 4.3: the frame r takes the place of the switch step t_s in the history of Equation (4.10) and in the future of Equation (4.11), and n and m are the lengths of the history and of the future. The history and the future are what the bridge is given, while the states between A and B are available only during training, where they can be used, for example, as a reference for the reward. Every window is expressed in the heading frame of its start state, as the request of Section 4.3, so it does not depend on where the motion was recorded or in which direction the performer was facing.

Windows are drawn in two steps. The start is drawn first, uniformly among the frames that allow at least the shortest duration together with the history and the future required, and the duration is then drawn uniformly among the durations still available from that start. From a stitched trajectory, only the starts that make the window cross the seam are allowed (Section 5.2.4). Training windows come only from the training splits of the two datasets and of the stitched trajectories, and evaluation windows from the validation splits; the stitched trajectories of each split are generated only from the clips of that split. Splitting by performance rather than by frame matters, because adjacent frames of a clip are nearly identical, and a split

by frame would place near duplicates on both sides and overstate how well a bridge generalizes. Figure 5.5 shows an example of a window.

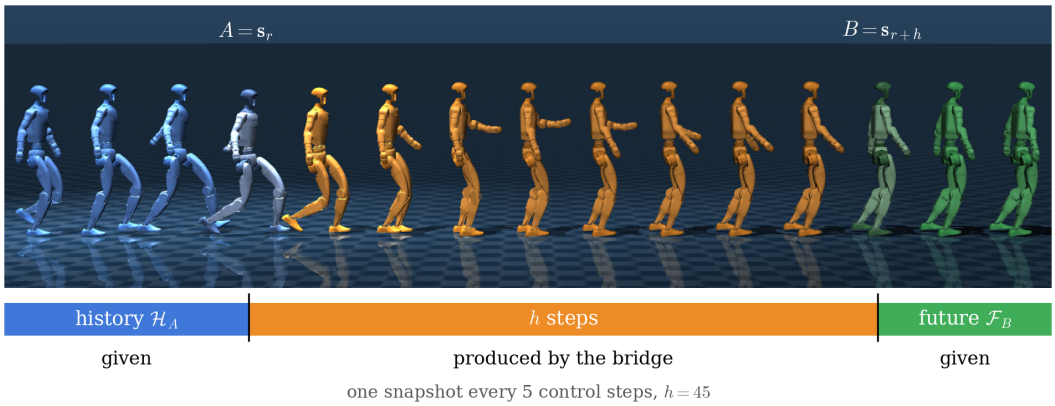


Figure 5.5. A transition window with $h = 45$, drawn with one snapshot every 5 control steps. The history $\mathcal{H}_A$, ending in the start state A, and the future $\mathcal{F}_B$, starting from state B, are given to the bridge, while the h steps between them are what the bridge has to produce.

5.2.6 Randomization and Augmentation

Several randomizations increase the variety of the windows seen during training, or make the bridge robust to the differences between the corpus and the states it meets at deployment. *Target displacement* and *time rescaling* change the request of a window (where B lies relative to A and how much time is given to reach it), while keeping the window a consistent example of how to connect them, and they address the domain shift and the duration mismatch discussed in Section 5.2.7. The others are standard techniques of motion learning and of transfer from simulation, applied here as they are.

Target displacement. In half of the windows, the history and A are shifted horizontally by a random offset $\boldsymbol{\delta}$ of at most 1 cm, while B stays where it is. Since the window is then expressed in the heading frame of A, the history and A look exactly as before, and the one thing that changes is where B lies relative to A. The frames in between are bent smoothly to connect A to the displaced B: with $\phi = k/h$ the phase of frame k of the window, the root position of the frame is shifted by $w(\phi)\,\boldsymbol{\delta}$, with

$$w(\phi) = 1 - 3\phi^2 + 2\phi^3, \tag{5.2}$$

which is one at A and zero at B, with zero slope at both ends, so the motion is unchanged near the endpoints. The root linear velocity of each frame is corrected by the time derivative of the shift, so that positions and velocities stay consistent. Only the root is moved, while the joints are left untouched, so the window still describes a coherent posture at every frame; as a consequence, a foot that is planted during the window slides together with the root, by at most

the size of the offset. This inconsistency is one reason to keep the offset small, and it is also the reason why the displacement is applied only when training the diffusion-based planner of the third architecture (Section 5.5.2): its plans are executed by a universal tracker, which follows a kinematic reference without having to reproduce it exactly and absorbs small inconsistencies. Figure 5.6 illustrates the displacement.

Time rescaling. The duration of the window is changed from h to h', with h'/h drawn between 0.8 and 1.25, and the motion between A and B is resampled in time. Frame k' of the new window, at phase $\phi = k'/h'$, takes the pose that the original window has at time $\sigma(\phi)\, h$, interpolated between the two nearest frames, where

$$\sigma(\phi) = c\,\phi + (1 - c)\left(3\phi^2 - 2\phi^3\right), \qquad c = \frac{h'}{h}. \tag{5.3}$$

The function σ goes from zero to one, so both endpoints are preserved exactly, and its slope at both ends is c, so the speed of the motion at A and at B is the same as in the original window, while the motion in between is accelerated or slowed down. Velocities are computed again from the resampled positions.

Mirroring. The window is reflected across the sagittal plane of the robot, exchanging left and right. A motion that starts with the left foot becomes one that starts with the right foot, so the corpus contains two ways of reaching mirrored endpoints, one for each foot.

Observation noise. Uniform noise is added to the proprioceptive observations of the policies trained in simulation, that is, the goal-conditioned policy (Section 5.3), the student of the conditional VAE (Section 5.4.2) and the universal tracker (Section 5.5.3); the planner (Section 5.5.1), trained offline without a simulator, does not observe the robot and does not use it. The noise reaches 0.1 m/s on the root linear velocity, 0.2 rad/s on the root angular velocity, 0.05 on the direction of gravity, 0.01 rad on the joint positions and 0.5 rad/s on the joint velocities.

Physics randomization. This randomization changes the simulated physics, so it applies only to the policies trained in simulation, and not to the planner, which is kinematic and never runs a simulator. At the start of training, the friction of the feet is drawn between 0.3 and 1.2, the center of mass of the base is shifted by a few centimeters, and a bias of up to 0.01 rad is added to the joint encoders.

5.2.7 Learning from a Corpus

Learning the bridge from windows adds four difficulties to those of Section 4.5, and every architecture of this chapter has to face them.

The first is that each request is demonstrated only once. A request can be realized by many motions, as discussed in Section 4.5, but a window shows only one of them, and since states are continuous, the same request practically never

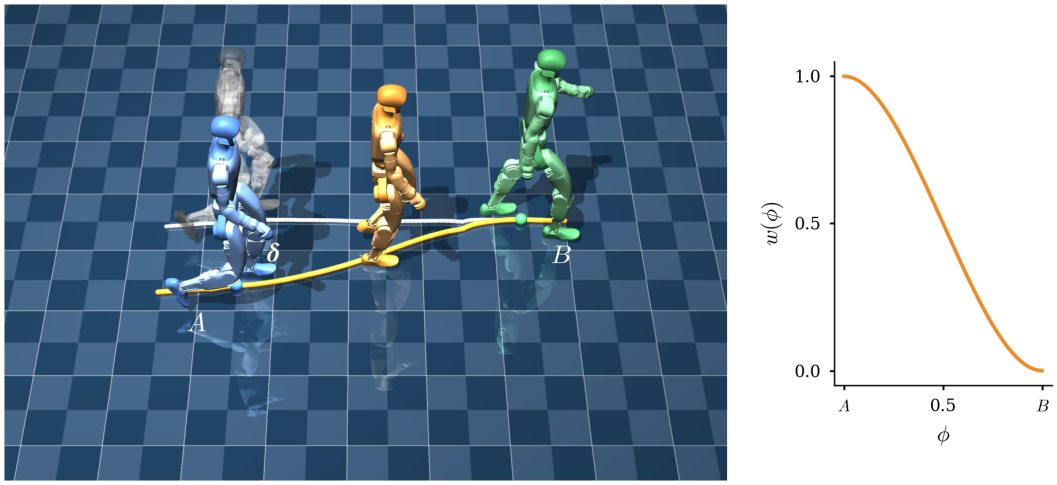


Figure 5.6. Target displacement. Left: the history and A are shifted by $\boldsymbol{\delta}$ while B stays in place, and the root path of the window (orange) is bent from the original one (white) to connect them; the offset is exaggerated for readability. Right: the weight $w(\phi)$ of the shift along the window, one at A and zero at B, with zero slope at both ends.

appears twice. A model trained on the corpus therefore never sees the alternatives for the same request side by side. Different windows, however, can have nearly the same request: two windows may both start from a walk at the same speed and end with the robot standing still, with nearly equal start states, target states and durations, and yet one performer stopped in two steps and the other in three. When the requests are this close and the motions differ, a model trained to reproduce the examples tends to learn their average, which is usually not a valid motion.

The second is a domain shift between training and deployment. During training, A and B come from the same window of the same clip, so they always belong to one continuous motion, whereas at deployment A is left by the source skill and B is a state of the incoming skill, which come from two different motions. The requests seen at deployment therefore follow a different distribution from those seen in training. The motion graphs of Section 5.2.4 reduce the shift, since they create trajectories in which two unrelated motions are connected.

The third, and possibly the largest, concerns the duration. During training the duration of a window is given by the data: it is the time the performer actually took to go from A to B, so it always suits the motion between them. At deployment the duration is chosen by whoever requests the switch, and it may be too short or too long for the motion between A and B. Drawing the duration of each window at random, and stretching the motion in time (Section 5.2.6), expose the bridge to durations that differ from the recorded ones, but only within a limited range.

The fourth is that the corpus is kinematic. The filters remove the frames that are clearly wrong, but they cannot guarantee that every window can be executed, and at deployment A and B are physical states of the skills, which differ slightly from any frame of the corpus. The bridge must therefore tolerate small differences between the states it has learned from and the states it is given.

5.3 Goal-Conditioned Policy

The first architecture developed in this thesis is the most direct answer to the problem: a single goal-conditioned policy (Section 2.1.3), trained with reinforcement learning, which receives the request and outputs actions. The goal of the policy is the target state B, not the recorded motion: it is free to reach B in any way. A reward given only at the deadline would however be too sparse to learn from, so the policy is also rewarded for moving like the recorded motion between A and B, which guides its exploration and gives its motion a natural style, without requiring it to reproduce the recording.

5.3.1 Architecture

The observation of the policy is made of a command, which describes the request, and of proprioception, which describes the robot. The command expresses the target state relative to the robot rather than in world coordinates, so that the same request looks the same wherever the robot is and whichever way it faces. It contains six differences between the target state and the current state, one for each component of the state: the position of the root, its orientation, its linear and angular velocities, and the positions and velocities of the joints. It also contains a clock, which tells the policy how much of the transition window has passed and how much is left.

To write the command, let $k = t - t_s$ be the number of steps since the interruption and $\eta_t = \min(k/h, 1)$ the fraction of the window that has passed. Let $\mathbf{R}(\mathbf{r})$ be the rotation matrix of a quaternion $\mathbf{r}$, which turns a vector from the frame of the root into the world frame, so that $\mathbf{R}(\mathbf{r}_t)^\top$ does the opposite; let $\mathbf{R}_z(\psi_t)$ be the rotation about the vertical axis by the current heading ψ_t of the root, that is, the direction the robot faces on the floor; and let $\rho_6(\cdot)$ be the six-dimensional representation of a rotation, formed by the first two columns of its matrix, which, unlike a quaternion, varies continuously with the rotation. Denoting with a superscript B the components of the target state, the command is

$$\begin{aligned}\mathbf{c}_t = \big(&\mathbf{R}_z(\psi_t)^\top(\mathbf{p}^B - \mathbf{p}_t),\ \rho_6\Big(\mathbf{R}(\mathbf{r}_t)^\top\mathbf{R}(\mathbf{r}^B)\Big),\\ &\mathbf{R}(\mathbf{r}_t)^\top(\mathbf{v}^B - \mathbf{v}_t),\ \mathbf{R}(\mathbf{r}_t)^\top(\boldsymbol{\omega}^B - \boldsymbol{\omega}_t),\\ &\boldsymbol{\theta}^B - \boldsymbol{\theta}_t,\ \dot{\boldsymbol{\theta}}^B - \dot{\boldsymbol{\theta}}_t,\ 1 - \eta_t,\ \eta_t\big).\end{aligned} \tag{5.4}$$

The first entry is the displacement to the target position, rotated by the heading only, so that it says how far the target is ahead of, beside or above the robot without depending on the tilt of the pelvis. The second is the rotation that brings the current orientation onto the target one. The third and fourth are the differences of the root velocities, expressed in the frame of the root, and the fifth and sixth the differences of the joint positions and velocities. The last two entries form the clock: the fraction of the window still left and the fraction already passed. With n_θ joints the command has $17 + 2n_\theta$ entries. The proprioception contains the linear and angular velocity of the root in its own frame, the direction of gravity in the root frame, the joint positions and velocities relative to the default pose, and the previous action, for $9 + 3n_\theta$ entries. For example, with $n_\theta = 23$ joints, the command has 63 entries, the proprioception 78, and the whole observation 141.

The policy outputs one number for each joint, and the raw action $\mathbf{u}_t$ is turned into joint-position targets as

$$\boldsymbol{\theta}_t^{\text{cmd}} = \boldsymbol{\theta}^{\text{default}} + \mathbf{s} \odot \mathbf{u}_t, \tag{5.5}$$

where $\mathbf{s}$ contains a scale for each joint, derived from the effort limit and the stiffness of its actuator, and $\odot$ is the element-wise product. A zero action therefore commands the default pose.

Actor and critic are separate multilayer perceptrons with three hidden layers of 512, 256 and 128 units and ELU activations, each with a running normalization of its inputs. The actor outputs the mean of a Gaussian distribution over the actions. Its standard deviation, one value for each joint, is not computed by the network but learned as a separate parameter, so the amount of exploration is the same in every state. The critic receives the same observation without noise and outputs a value estimate. At deployment the mean of the Gaussian is used, so the policy is deterministic. The policy has no memory: the clock gives it the timing and the previous action gives it one step of history, but contacts are not observed, so any contact phase that cannot be inferred from the current state is hidden from it.

5.3.2 Training

The reward compares the state of the robot with a reference through an error vector of eight error terms. For a state $\mathbf{s}$ and a reference $\hat{\mathbf{s}}$, the error terms are the root position error, the root orientation error, the root linear and angular velocity errors, and the largest position and velocity errors among the joints of the legs and waist and among the joints of the arms:

$$\begin{aligned}\mathbf{e}(\mathbf{s}, \hat{\mathbf{s}}) = (&\|\mathbf{p} - \hat{\mathbf{p}}\|_2,\ d_{\text{SO(3)}}(\mathbf{r}, \hat{\mathbf{r}}),\ \|\mathbf{v} - \hat{\mathbf{v}}\|_2,\ \|\boldsymbol{\omega} - \hat{\boldsymbol{\omega}}\|_2,\\ &\|\boldsymbol{\theta}_{\text{leg}} - \hat{\boldsymbol{\theta}}_{\text{leg}}\|_\infty,\ \|\dot{\boldsymbol{\theta}}_{\text{leg}} - \dot{\hat{\boldsymbol{\theta}}}_{\text{leg}}\|_\infty,\ \|\boldsymbol{\theta}_{\text{arm}} - \hat{\boldsymbol{\theta}}_{\text{arm}}\|_\infty,\ \|\dot{\boldsymbol{\theta}}_{\text{arm}} - \dot{\hat{\boldsymbol{\theta}}}_{\text{arm}}\|_\infty),\end{aligned} \tag{5.6}$$

where $d_{\text{SO(3)}}$ is the angle of the relative rotation. Using the worst joint in each group means that a single badly placed joint dominates its error term instead of being hidden by an average over the body. Each error term has a tolerance, collected in the vector

$$\boldsymbol{\epsilon} = (0.05,\ 0.05,\ 0.15,\ 0.30,\ 0.08,\ 0.80,\ 0.05,\ 0.75), \tag{5.7}$$

in meters, radians, meters per second or radians per second, and the robot is said to have arrived when every error term is within its tolerance. This binary test is too sparse to learn from, since it gives no information about how far the robot is from arriving. The errors are therefore turned into a smooth score, built in three steps.

The first step makes the error terms comparable. Each error term e_j is normalized by a reference scale b_j, its tolerance, which makes it dimensionless and comparable with the other error terms despite their different units, and the result is compressed by a logarithm:

$$z_j = \log\left(1 + \frac{e_j}{b_j}\right). \tag{5.8}$$

The map is zero for a null error, approximately linear, $z_j \approx e_j/b_j$, for errors small with respect to the tolerance, and logarithmic for large ones. It therefore preserves the resolution of the score near the target, where the policy has to be precise, while bounding the contribution of a large error term, so that it cannot dominate the others.

The second step combines the error terms into one number,

$$D = 0.7 \max_j z_j + 0.3 \operatorname{mean}_j z_j, \tag{5.9}$$

a weighted blend of the worst error term and of the average one. The worst error term has most of the weight, because a single large error can be enough to make the incoming skill fail, however small the others are; the average keeps every error term in the objective, so that reducing an error that is not the worst still increases the score. The weights 0.7 and 0.3 were chosen by hand.

The third step turns the combined error into a score between zero and one:

$$S(\mathbf{e}; \mathbf{b}) = \frac{1}{1+D} = \frac{1}{1 + 0.7 \max_j z_j + 0.3 \operatorname{mean}_j z_j}. \tag{5.10}$$

The score is one when all errors are zero and decreases as they grow, but it never becomes exactly zero, and it decreases slowly. This form was designed in this work as an alternative to the exponential kernels $\exp(-e^2/\sigma^2)$ commonly used in motion imitation [43]. An exponential kernel becomes practically zero, together with its gradient, as soon as an error is a few times larger than its scale, so an error term that starts far from its tolerance stops contributing to the reward and the policy can neglect it. With the logarithm and the reciprocal, every error term keeps a gradient however large it is, so the policy always has a direction in which to improve.

The reward has two main terms, both built on this score. The first is a continuous measure of how far the robot is from the recorded motion: at every step, the route term compares the state with the corresponding state of the window, the reference $\hat{\mathbf{s}}_t = \mathbf{s}_{r+k}$,

$$r_t^{\text{route}} = S(\mathbf{e}(\mathbf{s}_t, \hat{\mathbf{s}}_t);\, \boldsymbol{\epsilon})\,. \tag{5.11}$$

The second is a terminal term, which scores how close the robot is to the target, once, at the deadline:

$$r_t^{\text{terminal}} = \mathbb{I}[k = h]\; S(\mathbf{e}(\mathbf{s}_t, B);\, \boldsymbol{\epsilon})\,, \tag{5.12}$$

Here, the terminal term is zero at every step except the last step of the window, $k = h$. Scoring the target only at the deadline is deliberate: a reward for being close to B at any time would let the policy pass through B early, collect the reward and move on. The complete reward is

$$\begin{aligned} r_t = r_t^{\text{route}} + 8\, r_t^{\text{terminal}} - 0.05\|\mathbf{u}_t - \mathbf{u}_{t-1}\|_2^2 - 0.002\|\mathbf{u}_t - 2\mathbf{u}_{t-1} + \mathbf{u}_{t-2}\|_2^2 \\ - 10\, \ell_{\text{lim}}(\mathbf{s}_t) - 20\, \mathbb{I}[\text{fall}_t], \end{aligned} \tag{5.13}$$

where the third and fourth terms penalize changes of the action and their second differences, ℓ_{lim} penalizes joint positions beyond their soft limits, and the last term penalizes a fall. All terms except the terminal one are multiplied by the control

period of 0.02 s. No reward is given for energy, foot placement or contacts, which are left to the recorded route and to the simulated physics.

An episode ends either at the deadline or with a fall, detected when the root is tilted by more than about 46° from upright. The episode does not end when the robot first enters the tolerance region. The deadline is treated as a time limit, so the value function can estimate returns beyond it, while a fall is treated as a true end of the episode. The policy is trained with PPO (Section 2.1.2) for up to 15,000 iterations, a learning rate of 10^{-3} and an entropy coefficient of 0.005.

Each training episode is one window of the corpus, with a duration between 0.3 and 1.2 s, that is, $15 \leq h \leq 60$ control steps. The window is placed at the origin of the simulated environment. At the start of a training episode, the simulator is set directly to the first state of the window, A, including root and joint velocities, with joint positions clipped to their limits. Since the corpus is kinematic, the robot cannot always reproduce the recorded motion exactly, which is one of the reasons why the recorded motion is used only as guidance in the reward, and never as a condition for success.

5.3.3 Limitations

The trained policy does not copy the recorded motions: it learns to produce a motion between two states within the given time, with a style of its own. That motion, however, tends to be an average of the motions of the corpus, which is the first difficulty described in Section 5.2.7. Where similar requests are connected by different motions, the policy learns a compromise between them. The average motion generalizes poorly to requests outside the corpus, and its precision at the deadline is not sufficient: the state and the action at the handover are often too far from those the incoming skill expects for it to continue. This motivates an architecture with a latent variable, which can separate the different motions for the same request instead of averaging them.

5.4 Conditional VAE Policy

A conditional variational autoencoder (Section 2.5.3) models the distribution of the outputs compatible with a condition rather than a single output: each value of the latent variable is decoded into one output, and different values give different outputs for the same condition. This second architecture is built on the hypothesis that a latent variable can capture what the goal-conditioned policy averages together, that is, the aspects in which the motions that realize the same request differ: the style of the motion, the number of steps, their pacing and their spacing. Each value of the latent variable would then select one of these motions, so that they are kept separate instead of being averaged, and the latent variable could even be used at deployment to choose a style of motion. The architecture follows the teacher-student scheme of MaskedMimic [74]: a teacher that sees the recorded motion is trained first, and the bridge is then distilled from it as a conditional variational autoencoder that sees only the request. Figure 5.7 shows the networks of the student, described below, and how they are connected.

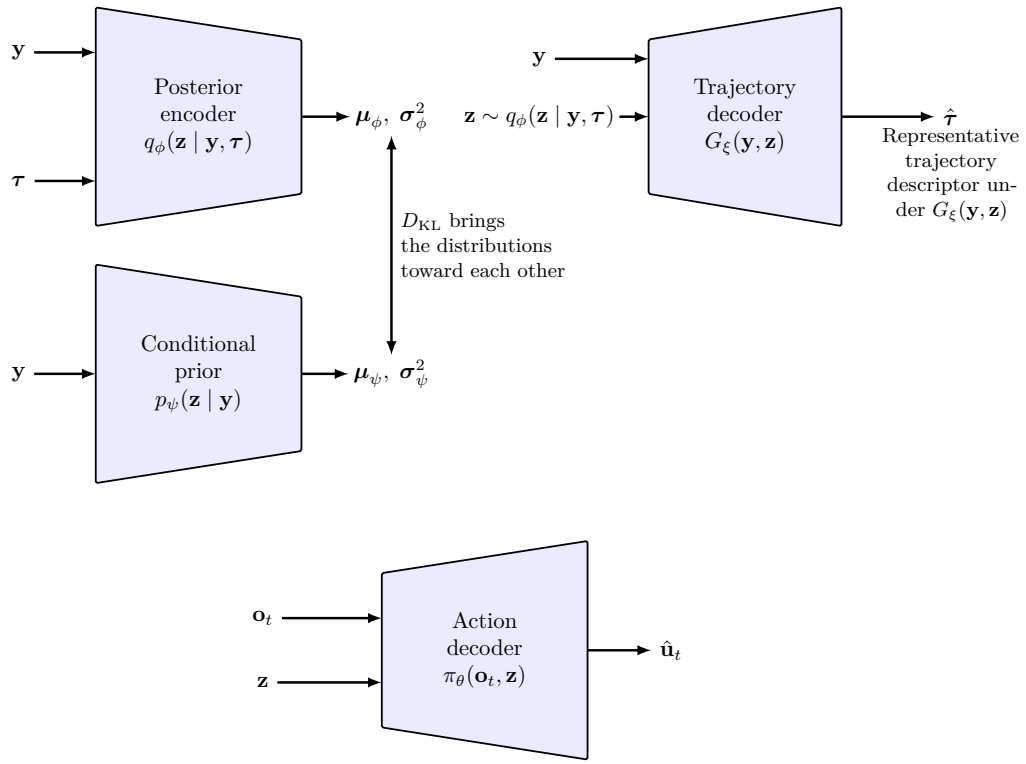


Figure 5.7. The request $\mathbf{y}$ conditions the posterior encoder, the prior and the trajectory decoder. During training, the posterior encoder also receives the recorded trajectory descriptor $\boldsymbol{\tau}$ and produces a latent distribution. The KL term brings this distribution toward the conditional prior, allowing the prior to provide meaningful latents at deployment. The trajectory decoder reconstructs a descriptor of the motion, including poses and foot contacts; it does not produce motor commands. The separate action decoder is the policy: it uses the current observation and the same latent to produce one robot action at each control step.

5.4.1 Teacher

The teacher is a universal tracker, a policy trained with PPO to follow any clip of the corpus rather than a single one. Like the bridge, it is a goal-conditioned policy, but its goal is different: as discussed in Section 2.1.3, a tracking policy can be read as a goal-conditioned policy whose goal is the upcoming part of its reference, and the goal of the teacher is the next frames of the recorded window rather than the target state alone. It observes privileged information that the bridge will not have: the poses and velocities of all tracked bodies expressed in the root frame, the joint positions and velocities, the previous action, and the differences between the robot and the next frames of the reference. Its reward combines smooth tracking terms for the root, the bodies, the joints of the legs and of the arms, and the feet, with penalties on the action rate, the joint limits and self-collisions, and episodes end when the root or the end effectors drift too far from the reference. For the teacher there is nothing ambiguous in a window: the recording says how to go from A to B, and the problem reduces to tracking it.

5.4.2 Student

The student is the bridge. Its observation $\mathbf{o}_t$ extends that of the goal-conditioned policy (Section 5.3.1) with the poses and velocities of the feet relative to the root and with their recent contacts. These quantities make the phase of the motion observable: two states with similar joint positions can require different actions depending on which foot is supporting the robot. The observation still contains the target state and the clock, so the policy remains closed-loop and can correct its motion when the simulated robot departs from the recording.

The request is also stored once, when the window opens, in a fixed condition $\mathbf{y}$. It contains the start state A, the target state B, the duration h, and the foot state at the start and at the end of the window. Unlike $\mathbf{o}_t$, which changes at every control step, $\mathbf{y}$ is constant for the whole transition. The latent $\mathbf{z} \in \mathbb{R}^{n_z}$ is constant as well. It represents one complete choice of motion between the two endpoints, including the number and order of the steps, their timing and spacing, and the remaining aspects of style. The implementation uses $n_z = 8$, deliberately small so that the latent describes the motion at the scale of a window rather than storing every pose independently.

During training, the recorded window is represented by a trajectory descriptor $\boldsymbol{\tau}$. A fixed number of frames is sampled uniformly between A and B; each frame contains the root pose and velocity, the poses and velocities of both feet in the root frame, and the two binary foot contacts. The root quantities are expressed in the heading frame of A, centered at the root of A and rotated by its heading, so that the same motion produces the same descriptor wherever it starts on the floor and whichever direction the robot initially faces. A change in the contact sequence describes a touchdown, so the number of steps and the foot that takes each one are properties of $\boldsymbol{\tau}$.

The student consists of four networks: a posterior encoder, a conditional prior, a trajectory decoder and an action decoder. The posterior and prior assign a distribution over the compact latent $\mathbf{z}$; the trajectory decoder uses that latent to describe a complete motion, and the action decoder uses the same latent to control the robot. The trajectory decoder's output is a descriptor of the motion, not a sequence of motor commands: it specifies quantities such as the root and foot poses and velocities, and when each foot contacts the floor. The action decoder is the policy that produces one motor command at every control step. All four networks are multilayer perceptrons with ELU activations and hidden layers of 512, 256 and 128 units, and the observations and trajectory descriptors are normalized with running statistics.

During training, each window is first encoded by the posterior encoder, which sees the request and the recorded motion and outputs a diagonal Gaussian over the latent,

$$q_\phi(\mathbf{z} \mid \mathbf{y}, \boldsymbol{\tau}) = \mathcal{N}\Big(\boldsymbol{\mu}_\phi(\mathbf{y}, \boldsymbol{\tau}), \operatorname{diag} \boldsymbol{\sigma}^2_\phi(\mathbf{y}, \boldsymbol{\tau})\Big) . \tag{5.14}$$

One latent is sampled from it and kept fixed for the whole window. Since the encoder sees the recording, two windows with similar endpoints but different motions can receive different latents. The action decoder is the policy that controls the robot: at each control step it receives the current observation and the latent, and produces

one action,

$$\hat{\mathbf{u}}_t = \pi_\theta(\mathbf{o}_t, \mathbf{z}). \tag{5.15}$$

In the rollouts it drives the simulated robot from A, and on the states it visits the frozen teacher supplies the target action $\mathbf{u}_t^T$, as described in Section 5.4.3. The trajectory decoder receives only the request and the latent, and reconstructs the whole descriptor in a single pass,

$$\hat{\boldsymbol{\tau}} = G_\xi(\mathbf{y}, \mathbf{z}). \tag{5.16}$$

Its output is compared with $\boldsymbol{\tau}$ but never executed. It serves to force the latent to describe the complete motion, including its contact sequence, rather than only what a single action needs, and since it does not receive $\mathbf{o}_t$, it cannot bypass the latent by reading the motion from the current state. For the contact channels it outputs the probability that each foot touches the floor at each frame; thresholding these probabilities gives the contact sequence of $\hat{\boldsymbol{\tau}}$, and with it the steps of the reconstructed motion. Finally, the conditional prior receives only the request,

$$p_\psi(\mathbf{z} \mid \mathbf{y}) = \mathcal{N}\Big(\boldsymbol{\mu}_\psi(\mathbf{y}), \operatorname{diag} \boldsymbol{\sigma}_\psi^2(\mathbf{y})\Big). \tag{5.17}$$

No latent is sampled from it during training. It enters only the KL term of the loss, which brings it toward the posterior and so teaches it which latents are plausible for a request without seeing the recording. By keeping the two distributions close, this term lets the prior provide meaningful latents at deployment, when the recorded trajectory is unavailable.

At deployment there is no recording, so neither the descriptor nor the posterior encoder is available, and the teacher is not used. When the transition begins, one latent is sampled from the prior and kept fixed until the handover, and the action decoder produces one action per control step from the current observation and that latent. The trajectory decoder is not needed for control, but it can decode the sampled latent to show the motion that the latent represents. The action decoder is therefore the same in both phases, and what changes is the source of its latent: the posterior during training and the prior at deployment, which the KL term brings to agree. Sampling another latent for the same request produces another possible transition.

The footsteps are not represented by a separate discrete variable. A model could first choose the number of steps and the foot that takes each one, and then sample the style of the motion, but the two choices would come from different distributions and nothing would guarantee that they are compatible, for instance that the sampled style belongs to a motion with the chosen number of steps. With a single latent, one draw of $\mathbf{z}$ determines both the reconstructed contact sequence and the actions, so the footsteps and the style always describe the same motion. The number of steps is not predicted directly but follows from the contacts that the latent encodes.

5.4.3 Training

The teacher is trained first and then frozen, and the student is trained from it with DAgger (Section 2.4.3). Each rollout starts from a corpus window: the simulator

is initialized in its start state A, one latent is sampled from the posterior of the window, and the action decoder controls the robot until the handover. At every visited state the teacher observes the recorded future of the window and supplies the action $\mathbf{u}_t^T$ that would bring the robot back toward the recording. A rollout ends early under the same conditions as the episodes of the teacher, when the root or the end effectors drift too far from the reference, since beyond that point the teacher can no longer provide a meaningful correction. The observations and teacher actions collected along the rollout are stored together with the request and the descriptor of the window, and the student is updated on them. Because these states are visited by the student itself, they include its own errors, and training on them reduces the compounding error that behavior cloning alone would suffer at deployment.

The rollout latent is taken from the posterior rather than from the prior. The teacher always steers the robot toward the recorded motion, which is the motion that the posterior latent encodes, so the student executes the same motion for which it receives corrections. A latent sampled from the prior could select a different but equally valid motion, for instance one that starts with the other foot, and the teacher actions would then contradict the motion that the student is executing. In addition, at the start of training the prior has not yet been fitted to the posterior, and its latents carry no information.

For an update, the latent of each stored window is sampled again from the current posterior with the reparameterization trick (Section 2.5.2),

$$\mathbf{z} = \boldsymbol{\mu}_\phi(\mathbf{y}, \boldsymbol{\tau}) + \boldsymbol{\sigma}_\phi(\mathbf{y}, \boldsymbol{\tau}) \odot \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{5.18}$$

and shared by all the samples of that window. The latent of the rollout is not reused, because the encoder has changed since the data were collected and because the loss must reach the encoder through $\mathbf{z}$. Let $\mathcal{T}$ be the set of control steps collected for the window, which is shorter than the whole window when the rollout ends early. The loss on one window is

$$\begin{aligned} \mathcal{L}_{\mathrm{CVAE}} = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}\mid\mathbf{y},\boldsymbol{\tau})} \left[\frac{1}{|\mathcal{T}|} \sum_{t\in\mathcal{T}} \ell_{\mathrm{act}}\Big(\pi_\theta(\mathbf{o}_t, \mathbf{z}), \mathbf{u}_t^T\Big) + \lambda_{\mathrm{traj}}\, \ell_{\mathrm{traj}}(G_\xi(\mathbf{y}, \mathbf{z}), \boldsymbol{\tau}) \right] \\ + \beta\, D_{\mathrm{KL}}(q_\phi(\mathbf{z} \mid \mathbf{y}, \boldsymbol{\tau}) \,\|\, p_\psi(\mathbf{z} \mid \mathbf{y})) \,, \end{aligned} \tag{5.19}$$

where the expectation is estimated with the single sample drawn above. The action loss ℓ_{act} is a smooth L_1 loss between the student and teacher actions. The trajectory loss ℓ_{traj} is the sum of a smooth L_1 loss on the continuous channels of the descriptor and a binary cross-entropy on the contact channels. It trains the decoded descriptor to resemble the recorded motion; it does not ask the trajectory decoder to predict the action sequence. Since both distributions in the KL divergence are diagonal Gaussians, the divergence has a closed form.

Each term has its own role. The action term teaches the action decoder to act like the teacher. The trajectory term makes the latent describe the whole motion, including when each foot touches the floor. The KL term makes the prior propose latents similar to those of the posterior, since at deployment the latent comes from the prior. The weight β starts at zero and grows during training, so that the decoders learn to use the latent before the prior is fitted to it. This makes posterior collapse less likely, but does not prevent it.

5.4.4 Limitations

The contacts of the decoded trajectory are not guaranteed to be executed. The two decoders share the latent but not their outputs, so a disturbance or an error of the action decoder can make a foot touch the floor at a different time than planned. Enforcing the plan would require a tracker that follows the decoded trajectory.

The latent can only separate motions that the corpus presents as alternatives. Most requests are demonstrated by a single window, so the prior sees little evidence of several motions for the same request. With too little variation, the action decoder can ignore the latent and behave like the goal-conditioned policy; the trajectory term and the schedule on β make this less likely but do not prevent it.

The prior is a single Gaussian. It represents well-separated families of motion poorly and may place probability between them, where the latents describe none of the demonstrated motions. In addition, the whole motion is compressed into a small vector and decoded in one pass, which limits how long and how precise a motion the model can describe.

Finally, the student is only as good as the teacher and the recordings it learns from. It cannot correct a window that is physically infeasible, nor a request whose duration does not fit its endpoints. These limits motivate the next architecture, which generates an explicit trajectory with a more expressive model and executes it with a universal tracker.

5.5 Diffusion Planner and Universal Tracker

The two previous architectures ask a single network both to decide how the robot should move and to produce the actions that make it move. The third architecture separates the two decisions. A planner, a diffusion model trained offline on the corpus, receives the request and generates a complete kinematic trajectory from A to B in h steps; a universal tracker, a policy trained with reinforcement learning to follow any short kinematic trajectory, then executes the plan on the simulated robot. The same split between a kinematic generator and a physical executor is used by the systems discussed in Chapter 3, such as CLoSD [72] and TextOp [73], and it addresses two of the limits of the conditional VAE (Section 5.4.4). The first is the limited expressiveness of its generative model: the conditional VAE compresses the whole motion into a small latent vector with a single Gaussian prior and decodes it in one pass, whereas the planner describes every frame of the motion explicitly and, as a diffusion model, can represent several separate modes for the same request (Section 2.6.1). The second is that the trajectory decoded by the conditional VAE is not enforced: its action decoder shares the latent with the trajectory decoder but not its output, so the robot may not move as the decoded trajectory describes. The plan, instead, is the reference that the tracker executes. Figure 5.8 shows the whole pipeline.

5.5.1 Planner

The planner generates a whole window $\mathbf{X}$ (Equation (5.1)) at once: given the history $\mathcal{H}_A$, the future $\mathcal{F}_B$ and the duration h, it fills all the frames between A and B

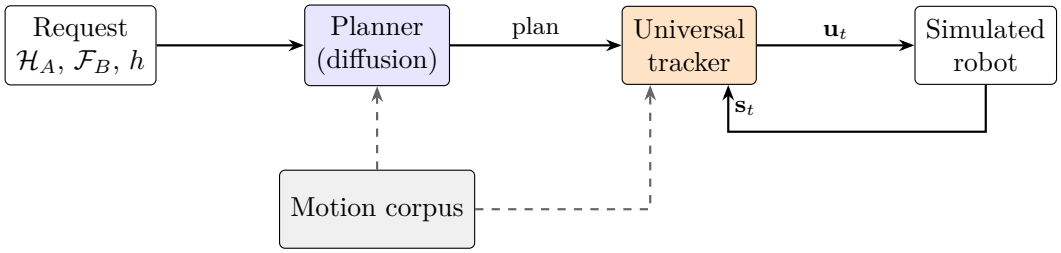


Figure 5.8. The diffusion planner and the universal tracker. At deployment (solid arrows), the planner turns the request into a kinematic plan once, at the interruption, and the tracker follows it with one action per control step. Both are trained on the motion corpus (dashed arrows).

together. The planner uses $n = 50$ frames of history and $m = 25$ frames of future, that is, 1 s before A and 0.5 s from B on, and durations from 15 to 60 control steps, from 0.3 to 1.2 s. One second of history covers a whole cycle of the gait, so the planner sees the stride, the cadence and the phase of the source motion. Half a second of future covers the step that follows B, and fixes the velocity with which the robot has to arrive there. The window is expressed in the heading frame of A (Section 5.2.5). Velocities are not part of it: they are computed from the positions after sampling, so that the positions and the velocities of a plan agree by construction.

Each frame i of the window is described by two groups of channels. The predicted channels are those the planner generates: the change of the root position since the previous frame, $\Delta\mathbf{p}_i = \mathbf{p}_i - \mathbf{p}_{i-1}$, the orientation of the root in the six-dimensional representation ρ_6 of Section 5.3.1, the change of the joint positions, $\Delta\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - \boldsymbol{\theta}_{i-1}$, and the contacts $\mathbf{c}_i$, one for each foot, equal to one when the foot is planted. The condition channels contain the root position $\mathbf{p}_i$, the joint positions $\boldsymbol{\theta}_i$ and the soles $\mathbf{f}_i$, the position and the heading of the sole of each foot, on the frames that are known, and zero on the other frames. Frame i of the window is therefore

$$\mathbf{x}_i = (\Delta\mathbf{p}_i,\ \rho_6(\mathbf{r}_i),\ \Delta\boldsymbol{\theta}_i,\ \mathbf{c}_i,\ \mathbf{p}_i,\ \boldsymbol{\theta}_i,\ \mathbf{f}_i), \tag{5.20}$$

with $11 + n_\theta$ predicted and $13 + n_\theta$ condition entries. The contacts and the soles are computed with the forward kinematics of the legs, and a foot is planted when its sole is within 5 cm of the floor and moves slower than 0.2 m/s. They are computed on the history and on the future separately, each from its own frames, so that they take the same values in training as at deployment. They tell the planner directly where the feet are and which of them supports the robot at the two ends of the window, which it would otherwise have to infer from the joint angles, and with them the phase of the gait. Positions are generated as changes, rather than directly, because the change between two frames has the same scale everywhere in the window and describes the speed of the motion, whereas the root position grows along the window and depends on where the motion goes. The plan is then rebuilt by adding up the changes from A. A plain sum, however, would not end exactly in B, since the small errors of the individual changes add up. Writing $\mathbf{q}_i = (\mathbf{p}_i, \boldsymbol{\theta}_i)$ for the positions of frame i and using the indices of Equation (5.1), where A is frame r and B is frame

$r + h$, the positions of the plan are

$$\tilde{\mathbf{q}}_i = \mathbf{q}_A + \sum_{j=r+1}^{i} \Delta\mathbf{q}_j, \qquad \mathbf{q}_i = \tilde{\mathbf{q}}_i + \frac{i-r}{h}\left(\mathbf{q}_B - \tilde{\mathbf{q}}_{r+h}\right), \qquad r < i \le r + h, \quad (5.21)$$

so the remaining gap to B is spread uniformly over the window instead of appearing as a jump at its end. The orientations are taken directly from the predicted channels. The poses are then brought back from the heading frame of A to the world frame, and the root and joint velocities are computed by central finite differences.

The known entries of the window are marked by a binary mask $\mathbf{M}$, which has one entry for each entry of $\mathbf{X}$. Every channel of the history frames is known, and so is every channel of the future frames, except the changes of B: $\Delta\mathbf{p}$ and $\Delta\boldsymbol{\theta}$ at B depend on the frame before B, which is not known. The planner fills the other entries by inpainting, that is, by completing a sample of which part is known. A diffusion model can impose known entries in two ways: by overwriting them with their values at every step of sampling, which makes them hold exactly in the result [61, 76], or by giving them to the network together with the mask, so that it learns to generate the rest consistently with them [67]. The planner does both. The condition channels serve the second purpose: they are never noised and never predicted, and they give the network the exact positions of the known frames and of their soles, which the predicted channels, made of changes, do not contain.

The denoiser is a transformer encoder [77] in which every frame of the window is a token. The entries of a frame are concatenated with the corresponding entries of the mask and projected linearly to a vector of 256 entries, to which a learned embedding of the frame index and an embedding of the noise level are added; the noise level is first encoded by sinusoidal features and then passed through a small multilayer perceptron. Four transformer layers with eight attention heads follow, and a final linear layer maps every token back to the size of a frame. Through attention every frame sees every other frame, including the history and the future at the two ends of the window, so the whole transition is denoised at once, as in MDM [56]. The duration is not a separate input: the window has a fixed length of $n + 60 + m - 1 = 134$ frames, enough for the longest duration, and the network reads h from the position of the known future frames. The frames after the last future frame repeat it and are ignored by the losses.

The diffusion process has $K = 100$ noise levels, with the cosine schedule of $\bar{\alpha}_k$ proposed by Nichol and Dhariwal [78]. As in MDM, the network predicts the clean window $\hat{\mathbf{X}}_0 = D_\vartheta(\mathbf{X}_k, \mathbf{M}, k)$ rather than the noise (Section 2.6.1), because the losses described below are defined on the clean motion. Plans are sampled with the deterministic sampler of DDIM [58] on 50 noise levels evenly spaced among the 100. Sampling starts from a window $\mathbf{X}_K$ of Gaussian noise on the predicted channels, with the known entries written in, and at each level k, followed by the next level $k' < k$, it computes

$$\hat{\mathbf{X}}_0 = D_\vartheta(\mathbf{X}_k, \mathbf{M}, k), \qquad \hat{\boldsymbol{\epsilon}} = \frac{\mathbf{X}_k - \sqrt{\bar{\alpha}_k}\,\hat{\mathbf{X}}_0}{\sqrt{1-\bar{\alpha}_k}}, \qquad \mathbf{X}_{k'} = \sqrt{\bar{\alpha}_{k'}}\,\hat{\mathbf{X}}_0 + \sqrt{1-\bar{\alpha}_{k'}}\,\hat{\boldsymbol{\epsilon}}, \quad (5.22)$$

where the known entries of $\hat{\mathbf{X}}_0$ and of $\mathbf{X}_{k'}$ are overwritten with their values. The estimate $\hat{\boldsymbol{\epsilon}}$ is the noise that $\mathbf{X}_k$ contains if its clean version is $\hat{\mathbf{X}}_0$, and the new

window combines the two at the lower noise level. The last prediction $\hat{\mathbf{X}}_0$ is the plan. Since the sampler is deterministic, the only source of randomness is the initial noise, and different noise draws give different plans for the same request. Figure 5.9 shows a plan for a switch from walking to jumping.

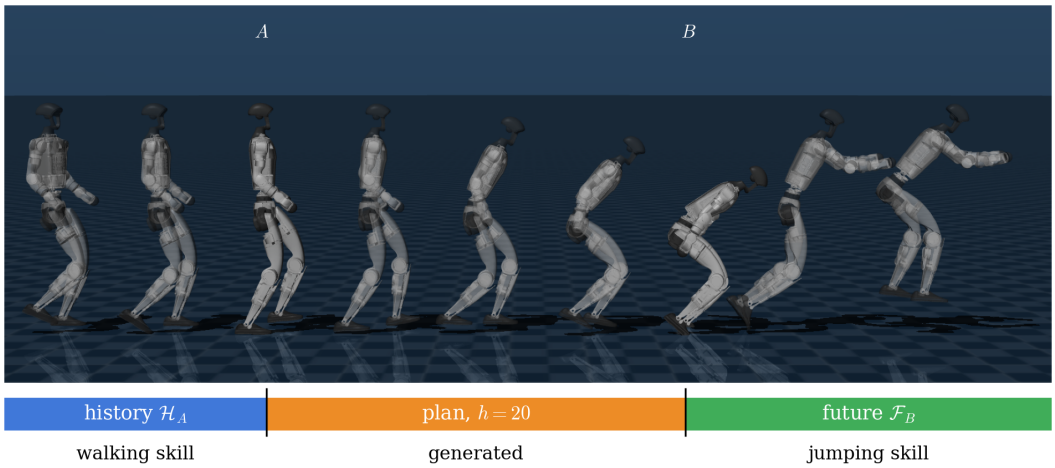


Figure 5.9. A plan of the diffusion planner for a switch from a walking skill to a jumping skill, with a duration of 0.4 s. The history is the motion of the walking skill before the switch, and the future is an entry sequence of the jumping skill, from the bottom of the crouch that prepares the take-off. The planner generates the 20 steps between A and B, which bring the walking robot into the crouch without first bringing it to rest. The states are drawn side by side, and not where the robot was.

5.5.2 Planner Training

At each iteration a batch of 256 windows is drawn from the corpus, with the target displacement, time rescaling and mirroring of Section 5.2.6, and the mask is built exactly as at deployment. In half of the windows, the joint positions of the history before A also receive Gaussian noise of 0.01 rad, which stands in for the small differences between the motion of a skill in simulation and the recorded motion; A itself is left exact, since the plan is built from it. Every channel is standardized by its mean and standard deviation, estimated once on 32 batches before training. A noise level is drawn uniformly for each window and the predicted channels are noised with Equation (2.53). The loss has five terms,

$$\mathcal{L}_{\text{plan}} = \mathcal{L}_{\text{rec}} + 0.5\,\mathcal{L}_{\text{cont}} + \mathcal{L}_{\text{end}} + \mathcal{L}_{\text{slip}} + \mathcal{L}_{\text{contact}}. \tag{5.23}$$

The reconstruction term $\mathcal{L}_{\text{rec}}$ is the mean squared error between $\hat{\mathbf{X}}_0$ and the clean window $\mathbf{X}_0$ over the unknown predicted entries, on the frames up to the last future frame. The continuity term $\mathcal{L}_{\text{cont}}$ compares the differences between consecutive frames: the window is completed with the known entries, the differences between consecutive frames of the completed prediction are compared with those of $\mathbf{X}_0$ by a mean squared error, and only the pairs of frames that contain an unknown entry are counted. A prediction can be accurate frame by frame and still jump between two

frames, especially at the seam between the known and the generated frames, and this term penalizes such jumps directly.

The endpoint term $\mathcal{L}_{\text{end}}$ asks the predicted changes to reach B on their own, before the correction of Equation (5.21) is applied:

$$\mathcal{L}_{\text{end}} = \text{mean}\left(\frac{\mathbf{q}_B - \mathbf{q}_A - \sum_{i=r+1}^{r+h} \Delta\hat{\mathbf{q}}_i}{\boldsymbol{\sigma}_\Delta \sqrt{h}}\right)^2, \tag{5.24}$$

where $\boldsymbol{\sigma}_\Delta$ contains the standard deviations of the change channels, the division and the square are taken entry by entry, and the mean is over the entries and the batch. If the errors of the single changes were independent, the error of their sum would grow with $\sqrt{h}$, so this normalization makes the term comparable across durations. Without it, the correction could hide a model whose changes do not add up to the motion between A and B, and spreading a large gap over the window would show up in the plan as a sliding motion.

The foot slip term $\mathcal{L}_{\text{slip}}$ measures the error where it matters for the tracker. The positions of the soles are computed from the rebuilt poses with the forward kinematics of the legs, written so that it can be differentiated. A foot of the recorded window is considered planted between two consecutive frames when its sole is within 5 cm of the floor in both and moves slower than 0.2 m/s; on those pairs of frames, the term penalizes the squared horizontal speed of the same foot in the prediction. The reconstruction term treats all the channels alike, but a small error on a hip angle can move the foot by several centimeters, and a foot that slides in the plan is a foot that the tracker cannot follow without slipping.

The denoiser is trained with AdamW, a learning rate of $2 \cdot 10^{-4}$ and gradients clipped to a norm of one, for 30,000 iterations. An exponential moving average of the weights, with a decay of 0.995, is kept during training and used for sampling.

5.5.3 Universal Tracker

The tracker is a universal tracker in the sense of Section 3.3.2: a single policy that follows any reference trajectory. Unlike the teacher of the conditional VAE, which observed the recorded window only to supply corrections during distillation, it is the controller of the robot at deployment, and its reference is the plan. It is trained with PPO (Section 2.1.2) on windows of the corpus, which plans resemble by construction since the planner is trained on the same corpus.

A training episode follows one window of a clip, with a duration between 0.3 and 2.0 s, longer than the longest plan, together with the 32 frames that follow its endpoint. The whole window is rotated by a random heading and placed at the origin of the environment, and the robot is initialized in its first state with a small random perturbation: up to 2 cm on the root position, 0.05 rad on the heading, 0.1 m/s on the root velocities, 0.02 rad on the joint positions and 0.15 rad/s on the joint velocities. The episode does not end at the endpoint of the window but continues along the following frames, so that the tracker sees the motion that continues after the endpoint and learns to arrive there moving as that motion requires, rather than as if the motion ended. The episode ends after the last frame, which is treated as a time limit, or with a fall, detected as in Section 5.3.2.

Let $\hat{\mathbf{s}}_t$ denote the reference state at step t. The observation of the actor has a reference part and a proprioceptive part. The reference part describes the upcoming reference at the offsets $\delta \in \{1, 2, 3, 4, 5, 8, 12, 16, 24, 32\}$ control steps ahead: for each offset, it contains the difference between $\hat{\mathbf{s}}_{t+\delta}$ and the current state, expressed as in Equation (5.4), with position and velocity differences rotated by the heading of the robot, the relative rotation of the root in six dimensions, and the differences of the joint positions and velocities, together with the offset in seconds. The dense offsets near the present tell the tracker precisely where to go next, while the sparse ones further ahead tell it where the motion is heading, for instance that a step is being prepared. The reference part also contains the same difference for the endpoint of the window, the fraction of the window that has passed and the time left; after the endpoint, the last frame of the episode takes its place and the clock counts the remaining frames. The proprioceptive part contains the last four states of the robot, including the current one, each described by the height of the root, its linear and angular velocity in the root frame, the direction of gravity in the root frame, and the joint positions and velocities relative to the default pose, followed by the last three actions. With $n_\theta = 23$ joints, the reference part has 683 entries, the four states 224 and the actions 69, for 976 in total. Observation noise and physics randomization are applied as in Section 5.2.6, with an additional noise of 1 cm on the root height. The critic receives the same observation without noise, together with the errors of the robot with respect to the current reference and to the endpoint, normalized by the tolerances of Equation (5.7).

The action of the tracker is a correction to the reference rather than a pose of its own:

$$\boldsymbol{\theta}_t^{\mathrm{cmd}} = \hat{\boldsymbol{\theta}}_{t+1} + \mathbf{s} \odot \mathbf{u}_t, \tag{5.25}$$

where $\hat{\boldsymbol{\theta}}_{t+1}$ are the joint positions of the reference at the next step and $\mathbf{s}$ are the scales of Equation (5.5). A zero action therefore commands the joints of the reference, and the policy only learns the corrections needed to keep its balance and to make the root follow the reference, which the joint positions alone do not achieve. Actor and critic are multilayer perceptrons with hidden layers of 1024, 512 and 256 units, ELU activations and a running normalization of their inputs. The standard deviation of the actions is a single learned parameter, shared by all joints and initialized to 0.7.

The reward is

$$r_t = 10\, S_{\mathrm{track}}(\mathbf{s}_t, \hat{\mathbf{s}}_t) - 0.05\|\mathbf{u}_t - \mathbf{u}_{t-1}\|_2^2 - 0.002\|\mathbf{u}_t - 2\mathbf{u}_{t-1} + \mathbf{u}_{t-2}\|_2^2 - 10\, \ell_{\mathrm{lim}}(\mathbf{s}_t) - 20\, \mathbb{I}[\mathrm{fall}_t], \tag{5.26}$$

with the regularization terms of Equation (5.13) and all terms multiplied by the control period. The tracking score uses the error terms of Equation (5.6), but scores each joint separately instead of taking the worst one. Each of the four root errors and each joint error e is passed through the bounded kernel

$$\kappa(e; \sigma) = \frac{1}{1 + (e/\sigma)^2}, \tag{5.27}$$

the kernels of the joints are averaged within each of the four groups of Equation (5.6) (positions and velocities of the legs and waist, positions and velocities of the arms), and S_{track} is the mean of the resulting eight values. The scales are

$$\boldsymbol{\sigma} = (0.15,\ 0.15,\ 0.30,\ 0.90,\ 0.24,\ 1.0,\ 0.15,\ 1.0), \tag{5.28}$$

in the same units and order as Equation (5.7), chosen close to the errors of a trained tracker, where the kernel is steepest. This score follows the opposite principle from the score of Equation (5.10). There, the reward scored a target that had to be reached, so every error term had to keep a gradient however large it was, and the worst one dominated. Here, the reference is kinematic and part of it can be impossible to follow: retargeting sometimes places an arm inside the torso, for example. With a bounded kernel and a mean over the joints, a joint that cannot reach its reference costs at most its share of one error term, and its gradient fades once its error is a few times the scale, so the policy stops chasing it and keeps improving the rest. For the same reason, self-collisions are not penalized: about a quarter of the reference frames put an arm into the body, and a penalty would push the whole body away from the reference, while the simulated physics already stops the penetration. The tracker is not rewarded for reaching the endpoint: it only follows the reference, and the precision at B comes from a plan that ends exactly in B and from how closely the tracker follows it.

The tracker is trained in 4096 parallel environments, with 24 steps per environment in each iteration, for up to 30,000 iterations, a learning rate of 10^{-3} adapted to keep the KL divergence between successive policies near 0.01, and an entropy coefficient of 0.005.

5.5.4 Deployment

When the source skill is interrupted, the planner receives the history $\mathcal{H}_A$, ending with A, the future $\mathcal{F}_B$, starting with B, and the duration h, and generates one plan, which requires 50 evaluations of the denoiser. The plan is generated once, at the interruption, and is not revised during the transition window. The tracker then follows it from A, one action per control step. Near the end of the window its look-ahead extends beyond B, where the reference is the future $\mathcal{F}_B$, which at deployment is an entry sequence of the incoming skill, long enough to cover the look-ahead of the tracker (Section 5.1), so the tracker sees how the incoming motion continues, exactly as in training. At step h control passes to the incoming skill.

5.5.5 Limitations

The plan is generated once, at the interruption, and is never revised. If the robot falls behind the plan or drifts from it, the tracker has to recover the gap alone, while the rest of the plan still assumes that the robot is where it should be. Planning again from the state actually reached, as CLoSD does [72], would remove this assumption, but each new plan would still have to reach the same B in the remaining time, and would add the cost of sampling during the window.

The precision at B depends on how closely the tracker follows the plan. The plan ends exactly in B, but the tracker is rewarded for following the reference along the whole window, not for arriving at its end, and the scales of its reward are a few times larger than the tolerances of the handover. Any tracking error at step h is an error at the handover.

The planner is kinematic. Physics reaches it only indirectly, through the corpus and the foot slip term, and neither guarantees that a plan can be executed.

Finally, a plan costs 50 evaluations of the denoiser, computed at the interruption before the first action, and the length of the window of the planner fixes the longest duration that can be requested, 1.2 s in this thesis.

Chapter 6

Experiments

This chapter describes the experiments used to evaluate the bridge. It first states the questions the experiments are meant to answer, then describes the setup. The experiments themselves are of two kinds. The transitions of Section 6.3 isolate a single switch between two skills and repeat it under controlled conditions, so that it can be thoroughly measured. The demos place the bridge inside complete tasks which represent real-world use cases, with a high-level controller that determines a plan and the skills that execute it, and are presented in Sections 7.1.1 and 7.1.2.

6.1 Goals

The bridge is trained on a motion corpus and knows nothing about the skills it connects. One of the main questions the experiments are meant to answer is whether it can connect skills it has never seen. A switch succeeds when the bridge drives the robot in a state of the good set of the incoming skill (Equation (4.3)), from which the incoming skill carries on as it would in nominal conditions. Membership in the good set alone does not tell how well the incoming skill performs, since the quality of Equation (4.2) varies within it. The experiments therefore measure incoming-skill performance relative to nominal executions. As discussed in Section 4.2, a switch requires both selecting the target state and reaching it within the deadline. This thesis addresses the second part and uses a proxy for the first (Section 5.1). The experiments therefore vary the conditions of the switch: how far from the target the robot is when the bridge starts, how much time the bridge is given. The incoming entry point is fixed within each quantitative comparison. Finally, the experiments observe how a bridge behaves when it is used in longer tasks, in which it is called repeatedly, alongside a high-level controller that proposes a plan and requests the switches without knowing how they are made.

6.2 Experimental Setup

6.2.1 Simulation Environment

All the experiments in simulation are carried out with mjlab [16], a framework that combines the manager-based interface of Isaac Lab [79] with MuJoCo Warp [80], a

GPU-accelerated version of the MuJoCo physics engine [81]. The physics is simulated at 200 Hz, while skills and bridges act at 50 Hz, so each action is held for four physics steps. Every policy outputs joint-position targets, which are tracked by joint-level proportional-derivative controllers.

The skills, the bridge and the objects they interact with share one simulated scene. Each skill was trained in its own environment, with its own observations. In the shared scene every skill keeps reading exactly the observations it was trained with, computed from the same robot and the same objects, and only the active policy sends its actions to the robot. After a trial has been initialized, the robot is never reset or moved during the switch or the incoming execution: its motion is entirely the result of the actions of the policies.

6.2.2 Robots

Two humanoid robots are used in simulation. The first one is the Unitree G1, with 29 actuated joints: six in each leg, three in the waist, and seven in each arm, three of which are in the wrist. In all the experiments the six wrist joints are kept fixed, so 23 joints are controlled. All the transitions of Section 6.3 and the soccer demo of Section 7.1.1 use the G1, for two reasons. Many motion datasets are published already retargeted to the G1 (Section 2.3.4), and the reference motions of the jumping and kicking skills are taken from them, so these skills could be trained without retargeting their motions first. The G1 is also capable of highly dynamic motions. The second robot is the Booster T1, with 23 actuated joints: six in each leg, one in the waist, four in each arm and two in the head. The Sokoban demo of Section 7.1.2 uses the T1 as it is a dynamically simpler task and showcases the use of a complex planner as control layer.

The bridge, as formulated in Chapter 4, makes no assumption on the robot it controls. Its architecture and training are the same for both robots. What changes is the motion corpus of Section 5.2, that must be retargeted and filtered so that only the motion within the capabilities of the robot is used (Section 5.2.3). For example, the limits of the T1 are much stricter than those of the G1. On the G1, a frame is invalid if the root tilts by more than about 45° from upright, if a foot is more than 6 cm below the floor, or if both feet are more than 5 cm above it for 0.25 s or longer, so that the short flights of running strides and hops are kept and only the long flights of jumps and leaps are removed. On the T1, the root may tilt by at most about 20° and the pelvis may not drop below 0.6 m, the root may move at most at 1 m/s horizontally and 0.35 m/s vertically and turn at most at 2 rad/s, the joints may move at most at 6 rad/s, the root and the joints may accelerate at most at 6 m/s^2 and 150 rad/s^2, and no flight phase is allowed, while the limit on the feet below the floor is the same. The filter keeps 96% of the AMASS frames and 93% of the LAFAN1 frames on the G1, against 73% and 34% on the T1.

6.2.3 Skill Pool

The formulation of Chapter 4 holds for any skill, and the ones used in the experiments were deliberately obtained in different ways. They fall into the three groups of policies described in Section 2.4: skills trained with a reward that describes the

task (Section 2.4.1), skills trained to track a reference motion (Section 2.4.2), and skills that combine a reference motion with a task reward, as discussed at the end of Section 2.4.2.

Walking (G1)

This skill learns from task rewards without a reference motion. It uses the velocity-tracking task provided by mjlab: the robot is commanded a forward, lateral and turning velocity, and it is rewarded for following the command, together with terms that keep it upright, lift its feet from the ground, prevent them from slipping, and keep its actions smooth.

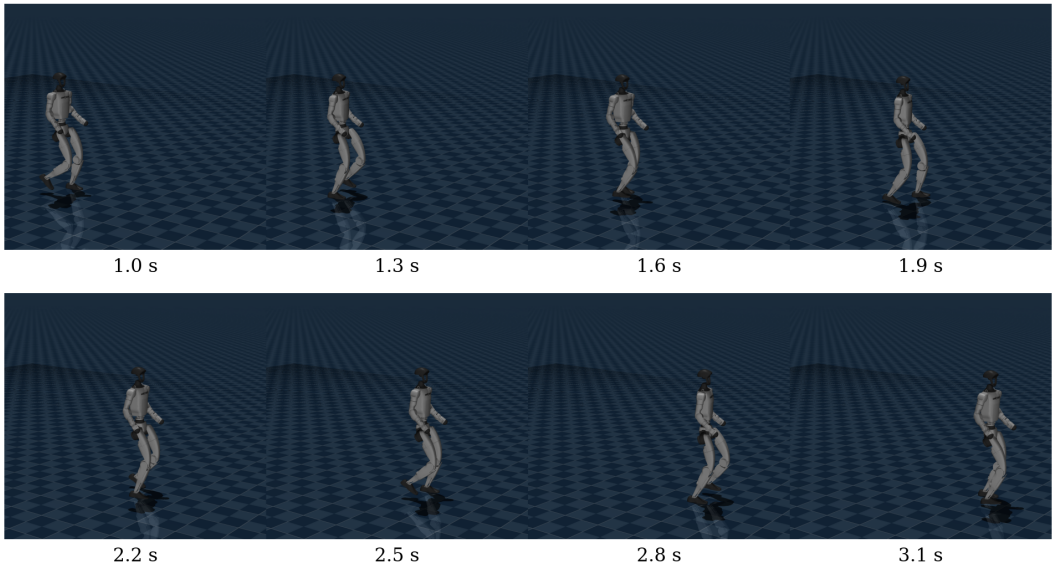


Figure 6.1. The walking skill of the G1.

Pushing (T1)

This skill also learns without a reference motion. The robot pushes a cube with a side of 1 m by a given distance. The skill reads how much of the push is left, how far the box is from its lane and a reference speed for the box, which grows from rest, stays at 0.4 m/s and decreases so that the box stops at the target. It is rewarded for walking steadily at that speed, for the speed and final position of the box and for keeping its hands on it, and it is penalized for touching the box with any other part of the body, for getting too close to it with the trunk, for leaving the ground with both feet, and for moving the box off its lane, tipping it over or turning it.

Pushing is more than walking forward with the arms stretched out. The box has a nominal mass of 8 kg and slides on the floor with a friction coefficient of 0.4. With the hands 0.76 m above the floor, it starts sliding under a push of about 31 N and tips over above about 52 N. The robot has to keep its push within this range, pushing harder to start the box and slightly less to slow it down. During training the mass of the box is randomized between about 0.6 and 1.6 times its nominal value.

The skill does not observe it: it observes the position, velocity and orientation of the box, and has to judge from how the box responds how hard to push. The same force also pushes back on the robot, which has to stay balanced against it while it walks.

Turning (T1)

The robot turns in place until it faces a commanded heading, at a turning rate of at most 0.5 rad/s. The skill is trained on the velocity-tracking task of mjlab with the linear velocity commanded to zero: it is rewarded for following the turning rate and for reaching the commanded heading, together with the terms of the walking task that keep it upright and regulate the motion of its feet.

Jumping (G1)

This skill learns by tracking a reference motion. It is rewarded frame by frame for following the reference, starts training episodes from random reference phases (Section 2.4.2), and terminates when the robot deviates too far from it. The reference is also supplied at deployment. The skill tracks a forward jump of 1.54 m taken from ASAP (Section 2.3.4). The clip is cut so that it starts two control steps before the lowest point of the crouch that prepares the take-off, so the skill always starts from this crouch.

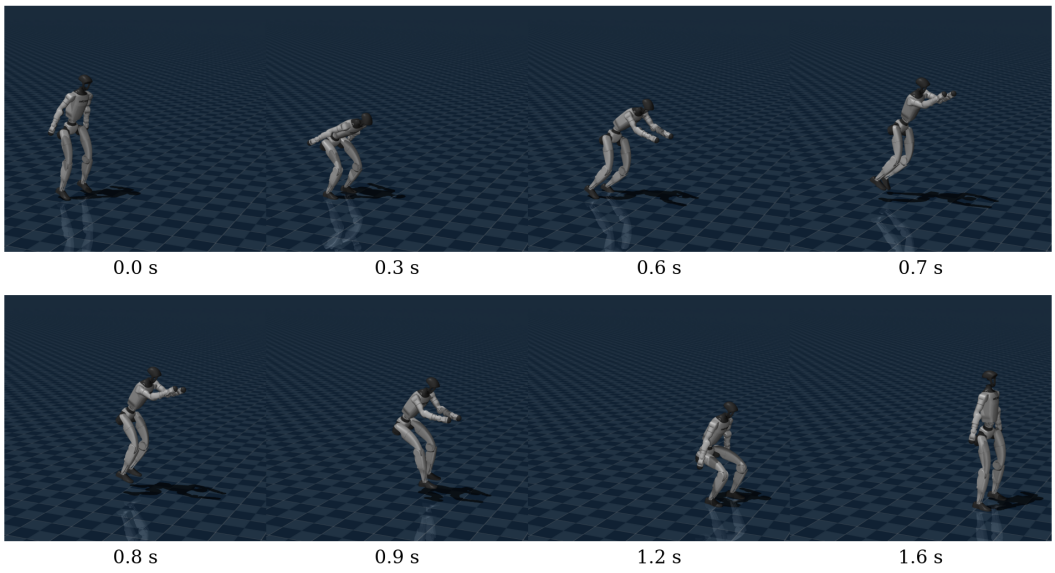


Figure 6.2. The jumping skill of the G1.

Kicking (G1)

This skill combines motion imitation with task rewards. It tracks a football kick taken from the motions published with PAiD (Section 2.3.4): the reference shows how to approach the ball, plant the supporting foot, swing and recover. A ball is placed in the scene, and two terms reward the robot for touching it and for the speed at which it leaves. The tracking terms keep their full weight during the whole

training, so that the robot kicks the way the reference does instead of finding its own way to move the ball. The reference expects the ball at a given position, where the swinging foot strikes it, and the ball is placed there. During training the ball is moved randomly around that position, by up to 8 cm at the end of a curriculum, so the policy has to read the position of the ball and adapt its swing.

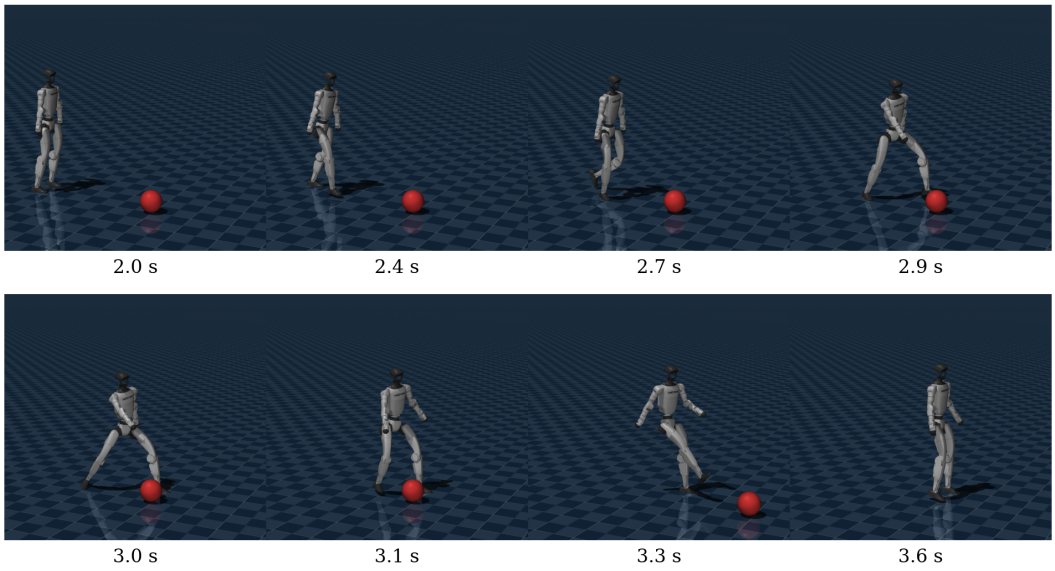


Figure 6.3. The kicking skill of the G1.

Walking (T1)

This skill learns from task rewards without a reference motion. The robot walks forward by a commanded distance, between 0.25 and 4 m, and stops at the destination. The skill reads how much of the distance is left, how far the robot is from the straight line to the destination and how far its heading is from the direction of that line. The distance is turned into a forward speed that rises to 0.5 m/s and then decreases, so that the robot comes to rest at the destination, and velocity-tracking terms reward the robot for following it. Two more terms reward the robot for staying on the line and for standing still at the destination. The episode ends if the robot overshoots the destination or strays from the line. The skill never turns or walks sideways. This is deliberate: the Sokoban demo of Section 7.1.2, in which the skill is used, moves the robot along the rows and columns of a grid, so the robot only needs to walk straight to a cell and changes direction by turning in place with the turning skill.

6.2.4 Entry Sequences

For the bridge, the target of a switch is the entry point of an entry sequence of the incoming skill (Section 5.1). Every entry sequence contains 33 states, the entry point and the 32 states that follow it, that is 0.64 s of motion, and it is kept together with the conditioning of the skill. The entry sequences of each skill are recorded once, before the experiments, in three steps:

1. the skill is executed 64 times in parallel with its conditioning, with slight initial randomization, each time long enough to cover a range of control steps and the 32 steps after its end;

2. one of the executions is chosen as representative of the skill;

3. the entry points are taken from the chosen execution at evenly spaced steps of the range and each of them, together with the following 32 states, forms an entry sequence.

The procedure is the same for every skill. Only the range and the number of entry points change, and the range is chosen according to what tells where the skill is in its motion.

Tracking skills. For the jump and the kick, the frame of the reference tells where the skill is. The range is a portion of the reference: the bottom of the crouch for the jump and the approach to the ball for the kick.

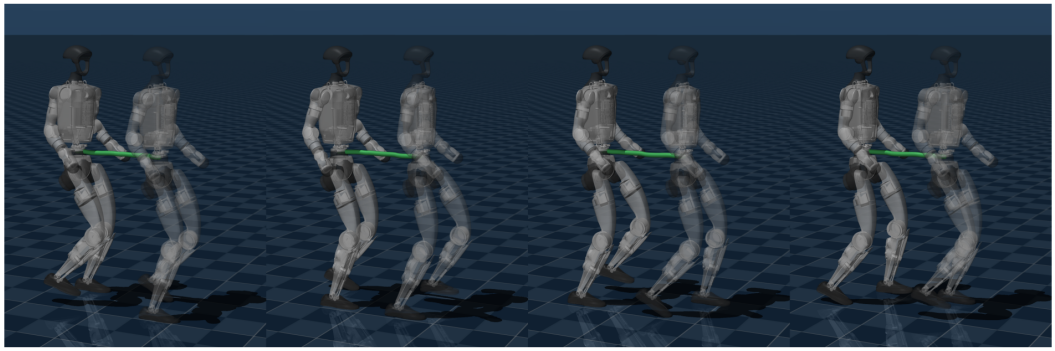
Periodic skills. Walking at a commanded velocity, as on the G1, has no start and no end. Once the gait has settled, the robot repeats the same cycle of steps, and every point of the cycle is equally valid as an entry point. The range starts once the gait has settled and covers one period of the gait, so that both feet are seen in every phase of their step.

Task skills. For the push, the turn and the walk of the T1, the progress of the task tells where the skill is: how far the box has moved, how much of the rotation is done, or how much of the distance has been walked. These skills have a real start, the situation in which the controller starts them, standing behind the box or standing still before turning or walking, and the range is a narrow one at that start, before the box, the heading or the robot moves.

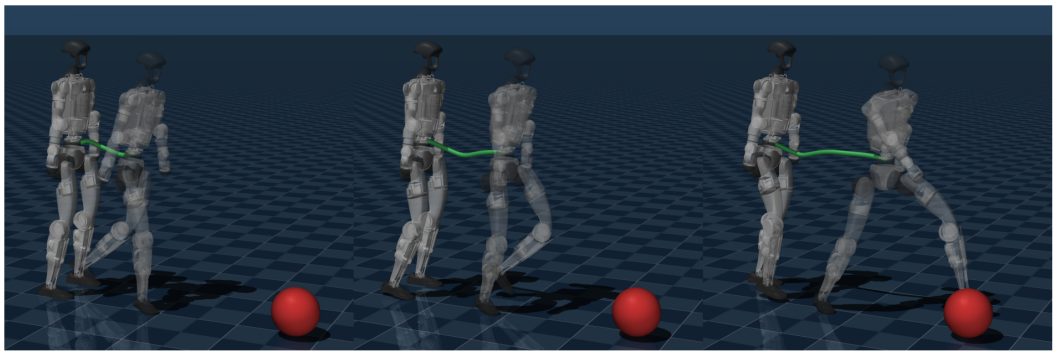
Table 6.1 lists the entry points of every skill, and Figure 6.4 shows the entry sequences of walking and kicking on the G1.

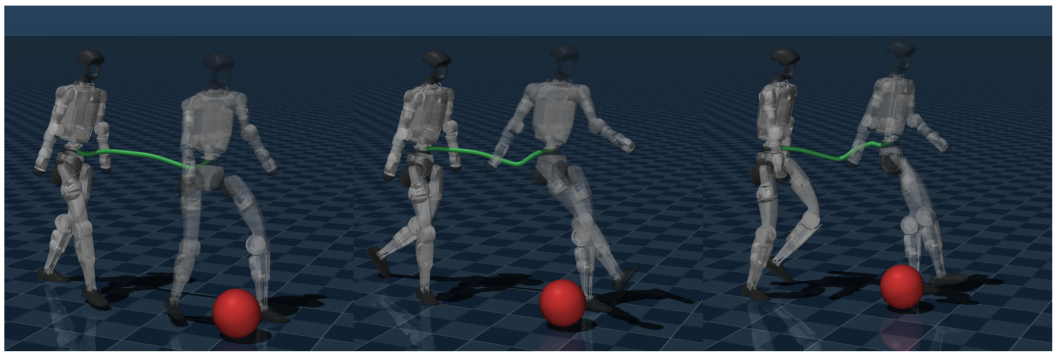
Table 6.1. Entry points of the skills. Steps are control steps at 50 Hz, counted from the first frame of the reference for a tracking skill and from the start of the execution for the other skills. Ranges include both ends.

	Range		
Skill	steps	s	Entry points (steps)
Tracking skills			
Jumping (G1)	2–3	0.04–0.06	2, 3
Kicking (G1)	95–144	1.90–2.88	99, 107, 115, 124, 132, 140
Periodic skills			
Walking (G1)	150–174	3.00–3.48	153, 159, 165, 171
Task skills			
Walking (T1)	50	1.00	50
Turning (T1)	5–9	0.10–0.18	7, turning left and right
Pushing (T1)	1–2	0.02–0.04	2



(a) Walking (G1).





(b) Kicking (G1).

Figure 6.4. The entry sequences of walking and kicking on the G1. In each panel, the robot on the left is the entry point, the robot on the right is the last state of the entry sequence, 0.64 s later, and the green line is the path of the pelvis between them. The entry points of walking cover one period of the gait. Those of kicking cover the approach to the ball, and the last of their sequences reach the swing and the strike.

The 64 executions of a skill differ slightly from one another, so at the same step the robot is not in exactly the same state in all of them. The second step therefore selects one of them as representative of the skill: the medoid, the execution closest overall to all the others. All the entry sequences of the skill are taken from it.

To compare two executions, their states are compared step by step over the range. Before the comparison, each state is expressed in its own heading frame, with its origin on the floor below the pelvis and its first axis along the direction the robot faces, so that two executions that differ only in where the robot is on the floor, or in which direction it faces, have the same states. A state, however, mixes quantities measured in different units and without a normalization there is no meaningful definition of distance.

The normalization divides each quantity by a scale with the same unit, which makes it dimensionless. The scale is chosen as the size of a difference that is considered significant for that kind of quantity. The quantities of a state are divided into six groups, each with its own scale σ_g: the height of the root, its tilt, its linear velocity, its angular velocity, the positions of the joints and their velocities. The distance between two states $\mathbf{s}$ and $\mathbf{s}'$ is then

$$d(\mathbf{s}, \mathbf{s}')^2 = \sum_{g=1}^{6} \frac{1}{n_g} \sum_{i \in g} \left(\frac{s_i - s'_i}{\sigma_g} \right)^2 , \tag{6.1}$$

where s_i is an entry of the state $\mathbf{s}$, s'_i is the corresponding entry of the state $\mathbf{s}'$ and n_g is the number of entries of group g. Each group contributes the mean of its squared normalized differences, so a group with many entries, such as the positions of the joints, does not count more than the height of the root only because it has more of them.

An execution, however, is a sequence of states, one for each step of the range. Two executions are therefore compared step by step: at every step of the range, the state of one execution is compared with the state of the other at the same step, and the squared distances are added up over the whole range. If $\mathbf{s}_t^{(k)}$ is the state of execution k at step t, the distance between executions k and j is

$$D_{kj}^2 = \sum_t d\big(\mathbf{s}_t^{(k)}, \mathbf{s}_t^{(j)}\big)^2 . \tag{6.2}$$

Two executions are close only if they are close at every step, that is, if the robot moves in the same way along the whole range. The medoid is the execution with the smallest total distance from all the others,

$$k^\star = \arg\min_k \sum_{j=1}^{64} D_{kj} . \tag{6.3}$$

The mean of the executions cannot take the place of the medoid, because it is not an execution of the skill. An entry sequence has to be on-policy, a motion the skill actually performs, and averaging states taken from different executions does not give one.

An entry sequence is recorded at some position and heading, while at test time it has to be placed in the scene, relative to whatever defines the task of the skill.

The whole sequence is moved as one, by a rotation about the vertical axis and a horizontal translation. For a tracking skill, the reference is first placed where the task requires it: the kick so that the ball expected by the reference coincides with the ball in the scene, and the jump so that its flight passes over the obstacle to be cleared. The entry sequence is then moved together with the reference, so that its relation to the reference, and therefore to the object, is preserved. The entry sequence of the push is placed in the same way relative to its box: behind the center of the box, at the distance at which it was recorded, and facing the direction of the push. Walking and turning have no object, and their entry sequences are moved so that the entry point is at the position and heading at which the controller wants the skill to start. Figure 6.5 shows an entry point of the jump placed in front of a walking robot, as in the transitions of Section 6.3.

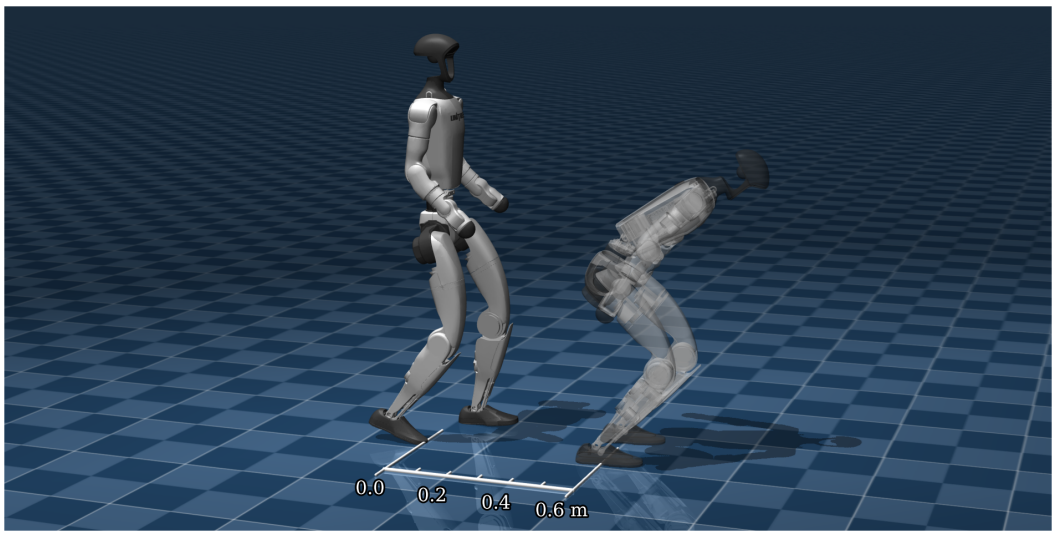


Figure 6.5. An entry point of the jump placed 0.6 m in front of a robot that walks towards it. The solid robot is a state of the walking skill. The ghost is the first entry point of the jump, at the bottom of the crouch, turned to face the direction of the walk.

6.3 Transitions

A transition is a single switch from a source skill to an incoming skill, repeated under controlled conditions so that its effectiveness can be measured. The three G1 pairs are compared using the bridge, a dedicated transition policy and an instant switch, as defined in Section 7.2.

In every transition, the entry point of the incoming skill is placed in the scene as described in Section 6.2.4 and shown in Figure 6.5, and the bridge starts when the robot is at a given distance before it, and has a given duration to complete the motion.

The demos of Sections 7.1.1 and 7.1.2 use configured entry points, distances and durations to illustrate repeated composition.

Three transitions are measured on the G1. Each of them is shown in a figure, with the distance and the duration given in its caption. The translucent robot is the

entry point the bridge has to reach and the policy in control is shown below.

6.3.1 Walking to kicking

The entry points of the kick lie in the approach to the ball, between 1.9 and 2.9 s of the reference, where the robot walks towards the ball before planting the supporting foot, so the bridge has to hand over a robot that is moving, with the correct foot on the ground and the correct speed. The ball starts at a fixed position, and the reference is placed so that its strike meets it. Neither is repositioned at handover. The incoming kick observes the ball and its reference and can correct moderate placement errors, but a larger error can produce a weak strike or a miss. Figure 6.6 shows the transition.

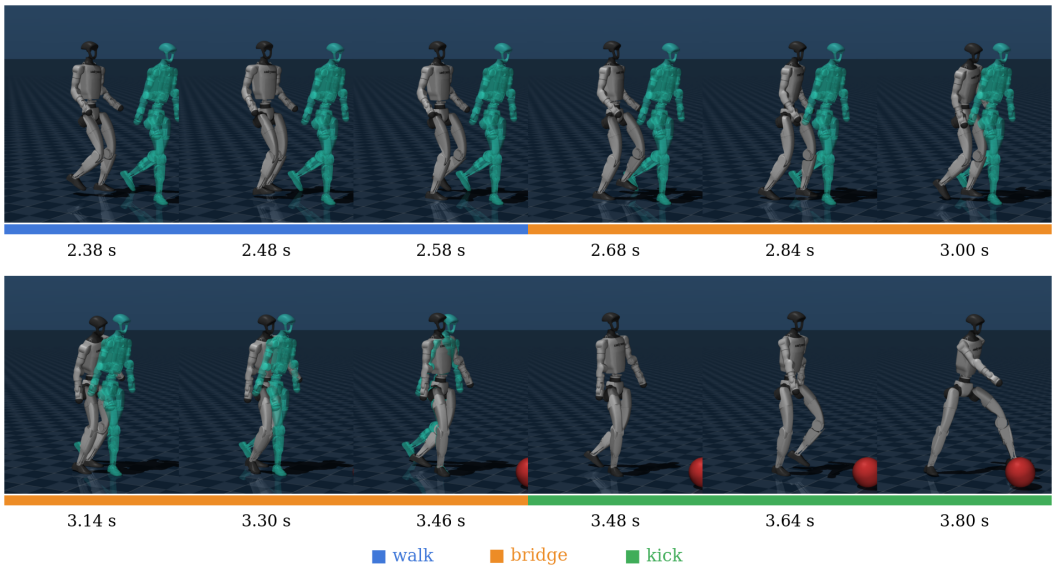


Figure 6.6. The switch from walking to kicking. The bridge is called 0.4 m before the entry point and lasts 0.8 s, and it hands over a robot that is still walking, in the approach to the ball.

6.3.2 Walking to jumping

The jump starts from a crouch, and its entry points are at the bottom of it, just before the legs extend for the take-off. Since its reference starts from this crouch, the jumping skill has never observed the robot in walking conditions. The bridge has to bring a walking robot into a crouch, and to stop its forward motion at a precise distance from the take-off. Figure 6.7 shows the switch towards the first entry point, and Figure 5.9 the kinematic plan that the planner generates for it, between the walking robot and the entry point.

6.3.3 Jumping to walking

This transition goes the other way, from the jump back to walking. The source skill is the jump, which runs from the start of its reference. The entry point of the walk

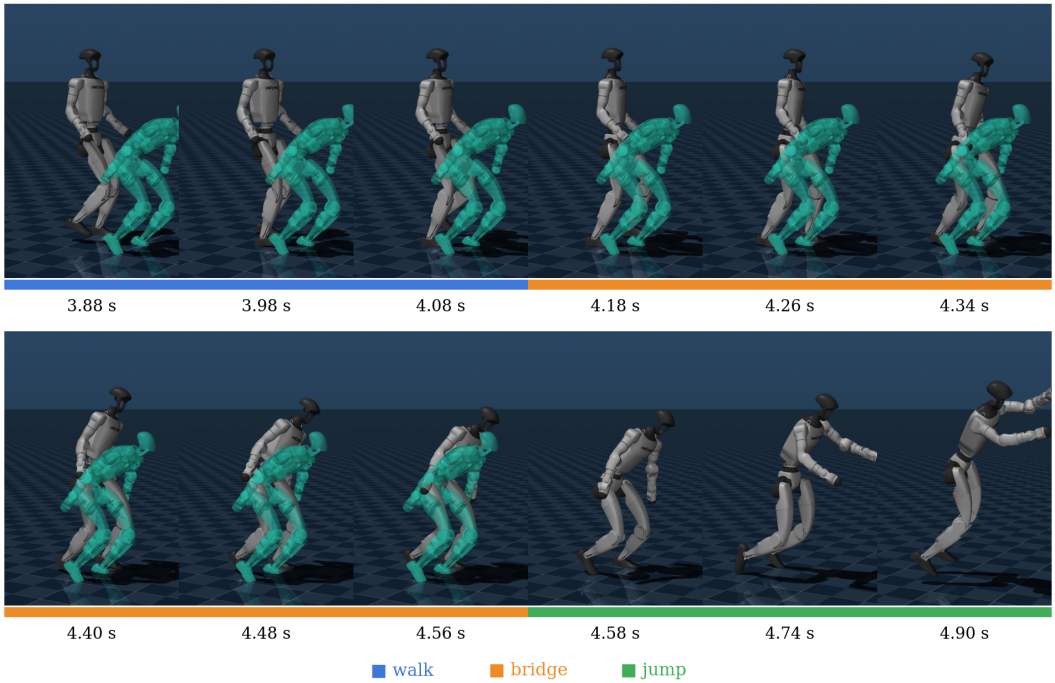


Figure 6.7. The switch from walking to jumping towards the first entry point, at the bottom of the crouch. The bridge is called 0.2 m before the entry point and lasts 0.4 s, and it brings the walking robot into the crouch from which the jump takes off.

is placed 0.1 m beyond the point where the reference of the jump ends, facing the direction of the jump. Figure 6.8 shows the switch.

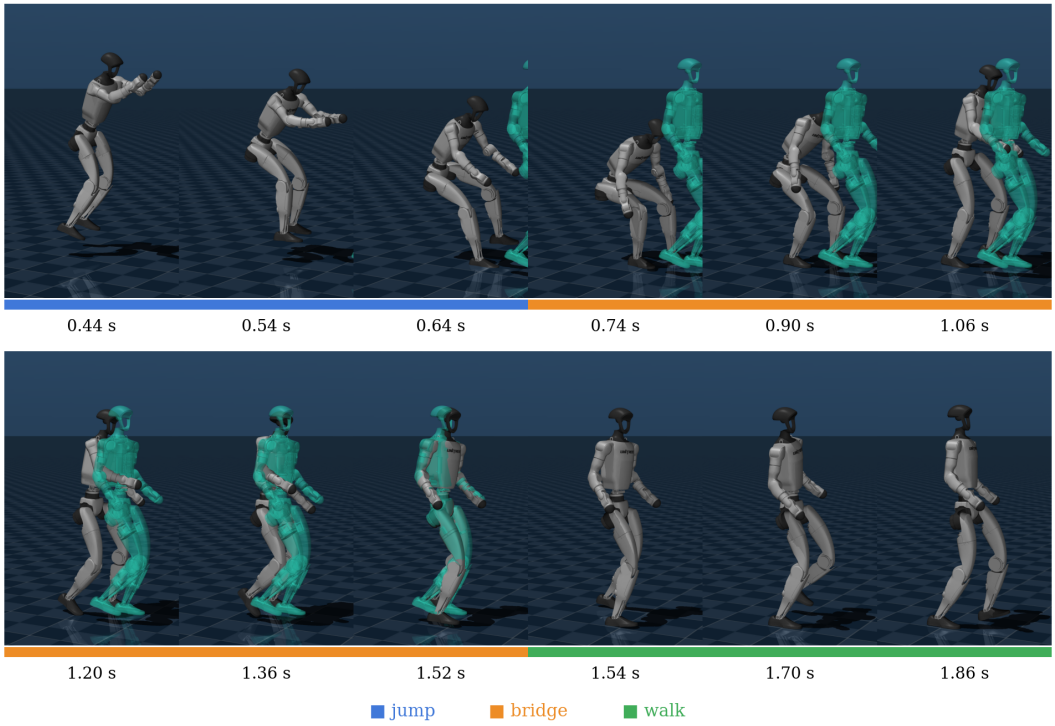


Figure 6.8. The switch from jumping to walking towards the first entry point of the walk. The bridge starts 0.6 m before the entry point, while the robot is still absorbing the landing, and lasts 0.8 s.

Chapter 7

Results

This chapter evaluates the diffusion planner and universal tracker of Section 5.5. It first describes what can be observed in individual switches and in two extended demos. It then introduces two baselines, a dedicated transition policy and an instant switch, and defines a common evaluation criterion: how well the incoming skill performs after taking control. The quantitative study measures nominal skill performance, examines the bridge's distance and duration parameters, and compares all three methods relative to the same nominal reference. Finally, it measures how abruptly the commanded joint targets change when control passes from one policy to another, which the skill measurements do not capture.

7.1 Qualitative Results

The main observation from the G1 transitions is that one bridge can connect motions with different postures, velocities and contact patterns. In representative executions of walking to jumping, the robot leaves its gait and lowers its pelvis into the crouch from which the jumping skill takes off. In walking to kicking, it enters the approach to the ball while still moving. After a jump, it reorganizes its posture and foot placement to enter a walking cycle. These are the transitions illustrated in Figures 6.6 to 6.8.

The same planner and tracker connect all three pairs. Visual continuity alone does not imply that the target state has been reached exactly: the robot can finish a transition close to the requested configuration while retaining an error in trunk orientation, joint velocity or foot contact. The incoming policy must absorb this error. A kick may then make a weaker contact with the ball, a jump may track its reference poorly or terminate early, and a walk may need several steps to settle. These effects motivate measuring the incoming execution.

The bridge needs time and space to adjust the robot's motion and reach its target, so the request distance and duration affect the transition. Asking for a large displacement in a short interval leaves little room to change posture and velocity; allowing more time changes the pace and contact sequence of the generated transition. The planner produces a kinematic path, and the tracker must realize it through the robot's dynamics. A plausible plan can therefore still lead to an inaccurate handover. The observations support the usefulness of a shared bridge, but do not establish that

it can connect arbitrary skills: its training corpus must cover the motions needed by the request, and the incoming skill must tolerate the state actually reached.

In both demos a high-level controller decides which skill runs next and requests the switches, which the bridge carries out.[1] Since the bridge only needs the request of Section 4.3, the controller can be as simple or as elaborate as the task requires, without knowing how a switch is made. The two demos are opposite cases. The soccer demo pairs a simple controller, which follows a fixed sequence, with dynamic skills: walking, jumping and kicking, connected while the robot is moving. The Sokoban demo does the opposite: a classical planner solves a puzzle, and simpler skills, walking, turning and pushing, carry out the plan.

As in the transitions, the demos do not select target states: every switch aims at an entry point of the incoming skill, placed according to the geometry of the task, and the bridge is called at a given distance before it, with a given duration. Each demo is presented through one representative run. It shows that frozen skills can be composed into one physical execution, but not how reliably: this is measured on isolated transitions, in the controlled comparison of Section 7.4.

The same mechanism also serves reactive behaviors, in which the switch is not planned in advance. To react to an event in the scene as soon as possible, the entry point of the incoming skill can be placed directly in front of the robot, at a suitable distance, and the bridge called right away with a suitable duration.

7.1.1 Soccer Demo

The soccer demo runs on the G1 and takes its setting from the matches between humanoid robots at RoboCup [82]. During a match, robots fall often. A fallen robot becomes an obstacle on the field: robots that walk into it often stumble over it and fall in turn. The demo shows that a bridge can be used to avoid such an obstacle actively, by switching to a skill that clears it. The jump was chosen for this because it is a difficult skill to enter, and because it requires the bridge to bring a walking robot into a crouch. In a real match, jumping over a fallen robot is not necessarily the best choice.

The pitch is 12 m long and 8 m wide (Figure 7.1). The robot starts at one end, facing a goal 8 m ahead, which is 2 m wide and 1.2 m high. A fallen robot lies on the pitch 3 m in front of it, and a ball rests 6 m in front of it. The task is to jump over the fallen robot, walk to the ball and kick it into the goal, which requires three switches: from walking to jumping, from jumping back to walking, and from walking to kicking.

The controller follows the fixed sequence of the task, and places every switch with simple geometric rules derived from the scene.

Jump. The take-off and the landing of the reference are known, so the distance the jump covers on the ground is known too. The reference is placed along the line of approach so that the point halfway between the take-off and the landing coincides with the fallen robot. The entry point is placed exactly where the reference takes off.

[1] Videos of the demos are available at https://ggldnl.github.io/Skill-Bridging.



Figure 7.1. The scene of the soccer demo. The robot starts at one end of the pitch, facing the goal. The fallen robot lies 3 m in front of it, and the ball rests 6 m in front of it.

Walk after the jump. One of the entry sequences of the walking skill, recorded with a speed of 0.8 m/s as conditioning, is placed so that its entry point is where the reference of the jump ends, facing the direction of the jump.

Kick. The ball has to travel in a straight line towards the goal. The direction in which the reference sends the ball is known, so the reference is rotated until this direction points from the ball to the center of the goal, and then translated until the position at which it expects the ball coincides with the actual ball. This fixes where the kick starts. The entry point used is about 2.5 s after its start, and it is where this placement puts it. Unlike the crouch of the jump, it is not a still posture: it lies in the middle of the approach, so the bridge hands over a robot that is already moving towards the ball, with the momentum the kick has built up in its approach.

While walking, the robot is commanded to walk at 0.8 m/s towards the next entry point. The bridge is called when the robot is at a given distance before the entry point. In the run shown here, every switch uses the setting with the highest score in the transitions (Table 7.6): the bridge into the jump is called 0.2 m before its entry point and lasts 0.6 s, the bridge back to walking is called after landing, 1.0 m before its entry point, and lasts 1.2 s, and the bridge into the kick is called 0.6 m before its entry point and lasts 0.8 s. Every decision of the controller is a rule derived from the position of objects in the scene.

The demo succeeds when the ball crosses the goal line between the posts and below the crossbar after being kicked. The jump counts as clearing the fallen robot

when, after a phase in which both feet are off the ground, the robot lands with its root at least 0.45 m beyond the fallen robot along the direction of the jump. The demo fails if the root of the robot drops below 0.4 m, if a phase takes more than 15 s or the kick more than 8 s, if the robot passes the entry point of a switch before being aligned with it, if the jump does not clear the fallen robot, or if a bridge does not hand over on time.

Figure 7.2 shows a successful run. The robot clears the fallen robot, resumes walking and sends the ball into the goal, and the same bridge connects all three stages while the robot remains in motion. Figures 7.3 and 7.4 show the switches into the jump and into the kick. Into the jump, the bridge stops the walking robot in the crouch from which the jump takes off. Into the kick, it preserves the approach to the ball rather than interrupting it with a standing phase.

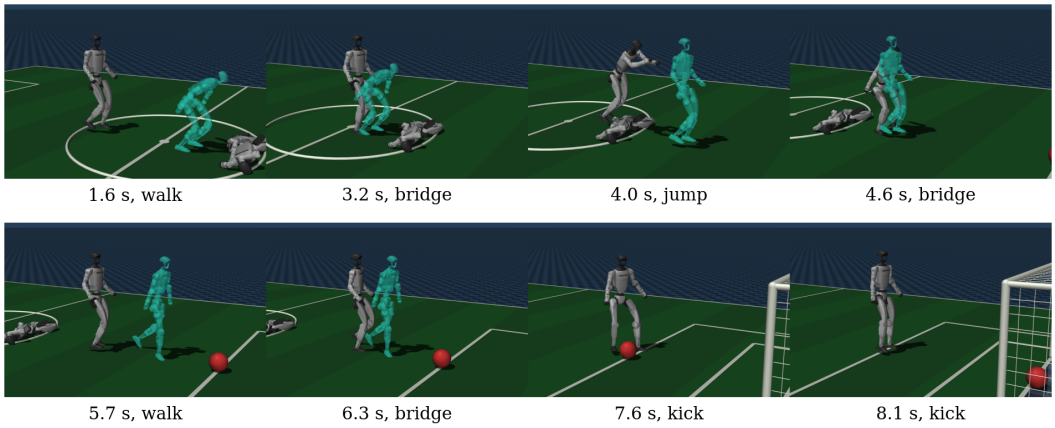


Figure 7.2. A successful run of the soccer demo. The robot walks towards the fallen robot, a bridge brings it into the crouch of the jump, the jump clears the fallen robot, a second bridge brings the robot back to walking, and a third one brings it into the approach of the kick, which sends the ball into the goal.

7.1.2 Sokoban Demo

Sokoban is a puzzle played on a grid, in which a player has to push boxes onto goal cells. The player can only push a box, never pull it, and only one box at a time. A careless push can leave a box in a corner from which it can never be moved. Solving a puzzle requires planning the whole sequence of moves in advance. In this demo the player is the T1. The grid is made of square cells with a side of 1 m, and every box is a cube as large as a cell, which the robot moves with the pushing skill of Section 6.2.3. The robot walks between the cells with the walking skill, and changes direction with the turning skill. A bridge trained on the T1 motion corpus is called between these skills to prepare the next one.

The puzzle is solved before the simulation starts, by a classical planner run through the Unified Planning library [83]. The puzzle is described with two actions, moving the player to a free adjacent cell and pushing a box by one cell, and the planner looks for a plan that puts every box on a goal with the smallest number of pushes. The plan is translated into a sequence of walking and pushing actions,

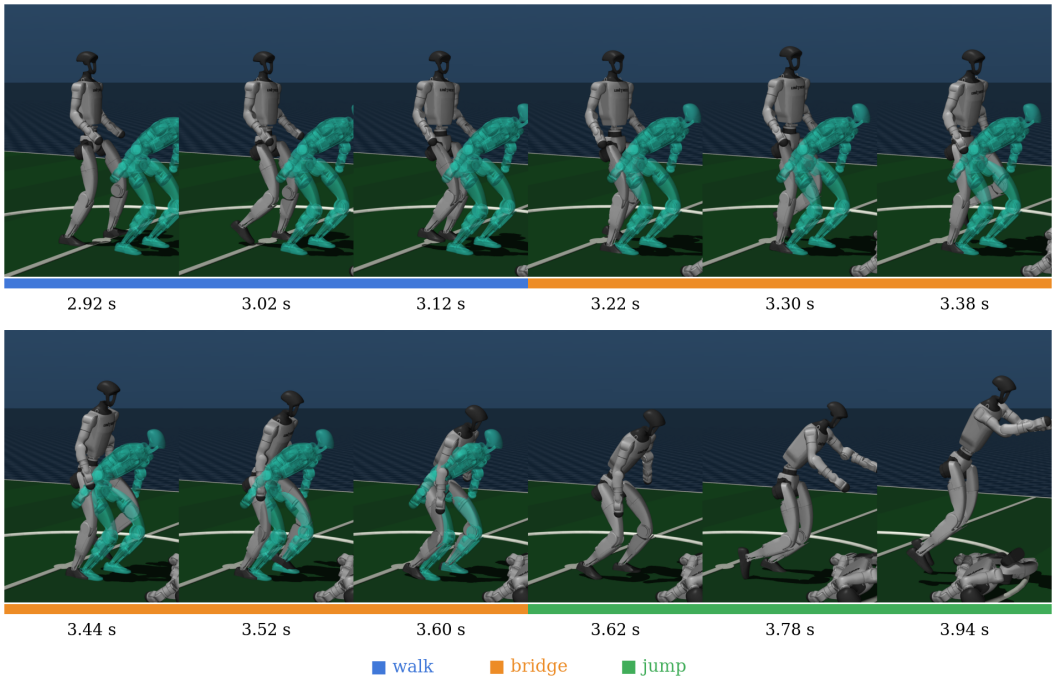


Figure 7.3. The switch from walking to jumping in the run of Figure 7.2. The bridge is called 0.2 m before the entry point, lasts 0.6 s, and brings the walking robot into the crouch from which the jump starts.

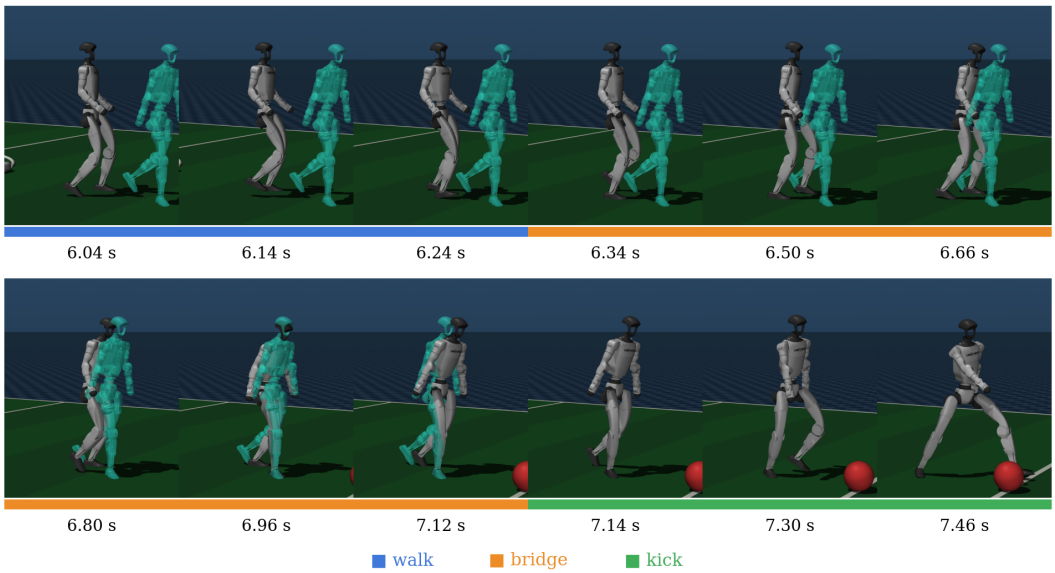


Figure 7.4. The switch from walking to kicking in the run of Figure 7.2. The bridge is called 0.6 m before the entry point, lasts 0.8 s, and hands over a robot that is still walking, in the approach to the ball.

each with a direction and a number of cells, and it is checked cell by cell against the map: no action may cross a wall or a box, every push must have a box in front of the robot and a free cell behind the box, and at the end every box must lie on a goal. Consecutive pushes of the same box in the same direction are merged into one push over several cells, which the pushing skill performs in a single run. The plan is therefore known to solve the puzzle before the robot moves.

The controller executes the plan one action at a time. A walking action is carried out as one straight walk: the walking skill is commanded the distance to the center of the last cell of the action, along the direction of travel, and the action ends when the skill has brought the robot to rest there, within 4 cm of the center of the cell and 0.05 rad of the direction of travel. When the robot does not face the direction of travel, it first switches to the turning skill, turns in place, and switches back to walking. Before a push, the robot walks in the same way to the cell behind the box, facing the direction of the push. The controller then checks that every box is in the cell where the plan expects it, gives the pushing skill its command, and requests the switch. The push ends when the box has been at rest in its destination cell, within 4 cm of its center and 0.04 rad of its heading, for 0.2 s, and a bridge then brings the robot back to walking.

Every switch of the demo, between walking, turning and pushing, is made towards an entry point of the incoming skill (Table 6.1), placed where the incoming skill has to start. The entry point of the push, recorded at its start, is placed 1 m behind the center of the box, where the hands of the robot nearly touch its face, facing the direction of the push. The entry points of walking and turning are placed where the robot is when the switch is requested, facing where it faces. The entry sequence of walking is recorded standing still, with a distance of zero as conditioning, and the walking skill receives the distance to walk only after the handover. Turning has one entry sequence for each direction of the rotation, left or right, recorded with that direction as conditioning. Every switch is requested with a duration of 1 s. The bridge hands over as soon as the robot is within the endpoint box of the entry point and, before a push, within 8 cm and 0.1 rad of it, since the hands of the robot have to meet the face of the box. The controller never corrects the scene by hand and never replans: if a bridge does not bring the robot to its entry point within 10 s, or if an action takes more than 60 s, for example because a box is not where the plan expects it, the execution stops and the failure is reported.

The puzzle used in the demo is shown in Figure 7.5. It is a room of 5 by 7 cells, with one box and one goal, divided at the top by a wall three cells long that stands between the box and the goal. The box therefore cannot be pushed straight onto the goal: it is pushed down by one cell, then by three cells along the passage below the wall, and finally up by one cell onto the goal. The solution consists of 11 moves of one cell, 5 of which are pushes, grouped into six walking actions and three pushes. Every push goes in a different direction, and between two pushes the robot has to walk around the box to reach the cell behind it, so the robot turns in place before almost every action, and the execution of the plan requires more than twenty switches between walking, turning and pushing.

Figure 7.6 shows the scene of this puzzle at the start of a run, and Figure 7.7 a complete run that solves it in about 89 s, through 23 switches. Each panel of Figure 7.7 is taken halfway through a stretch in which one skill is in control, while

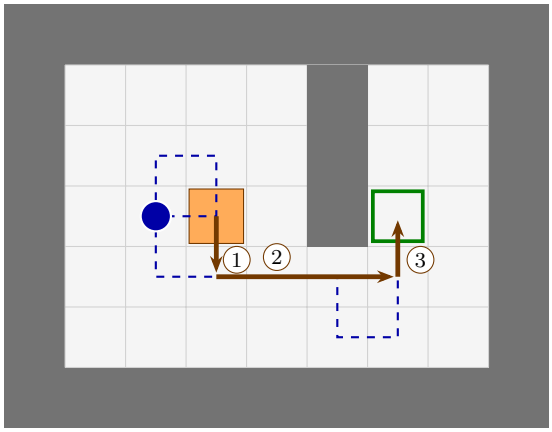


Figure 7.5. The puzzle of the Sokoban demo, seen from above. Walls are dark, the box is orange, the goal is the cell outlined in green, and the robot is the blue circle at its start. Every cell is 1 m wide. The plan pushes the box three times, in the order of the numbers, and the dashed line is the path of the robot, which follows the box during each push.

the bridges between the stretches are not shown. The run follows the plan action by action, and shows how a symbolic plan can be carried out by frozen physical policies. Before every walking action the robot turns in place towards the direction of travel, and before every push it walks to the cell behind the box, so that turning, walking and pushing alternate through the whole run. The three pushes move the box down by one cell, along the passage below the wall by three cells, and finally up onto the goal.

Here the role of the bridge is less dramatic than in the soccer task. Walking ends close to rest, turning is performed in place, and pushing starts from a nearly stationary robot behind a box. The skills therefore have more overlap in their acceptable starting conditions. The bridge can adjust the robot's pose before a turn or align it with the start of a push, but it also adds execution time and another opportunity for error. Over ten runs of the puzzle, the bridge solved it nine times, in about 91 s, while switching directly between the skills solved all thirteen runs, in about 71.5 s. The one failure came at the first push: the robot fell 1.6 s after the bridge handed it over to the pushing skill. Similar initiation postures therefore do not eliminate transition errors, and placement still matters because the hands have to meet the box at the start of the push.

The value of this demo is the connection between planning and execution. The planner chooses the sequence of actions, the controller supplies their conditioning and entry requests, and the bridge realizes the changes between the corresponding motions. Together, the two demos show both a use case with substantial changes of motion and one in which the additional transition mechanism may offer a smaller benefit.

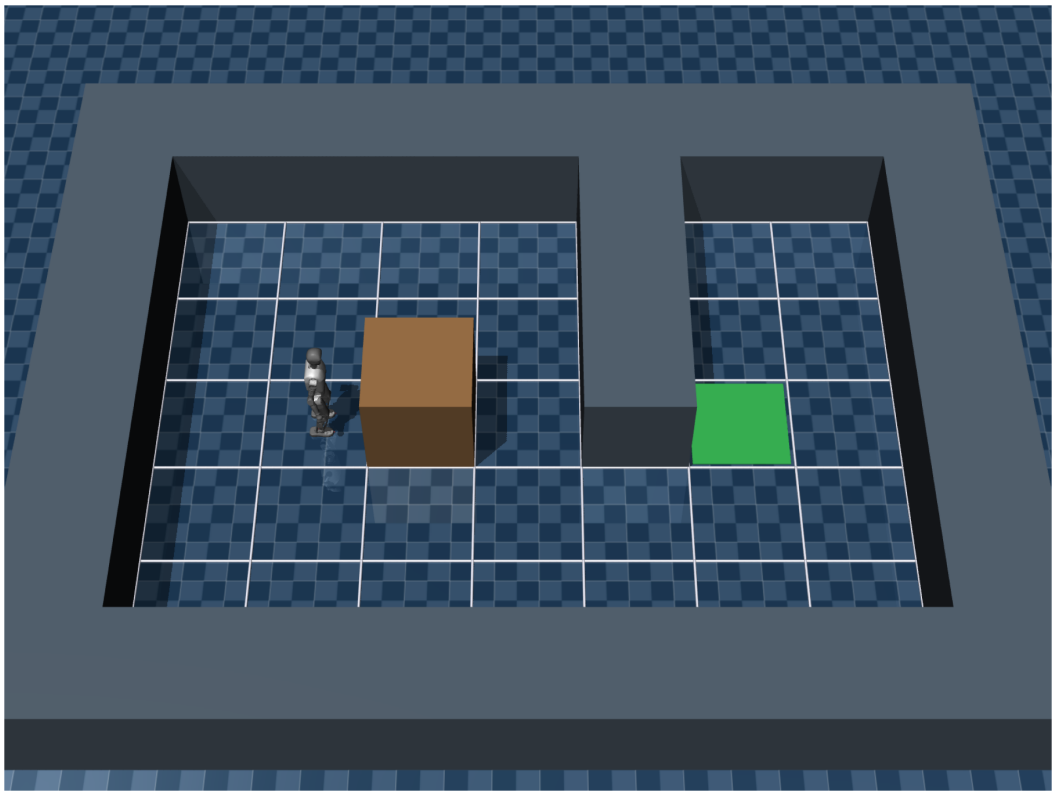


Figure 7.6. The Sokoban demo at the start of a run, with the map of Figure 7.5. The robot stands next to the box, and the goal is the green cell beyond the wall.

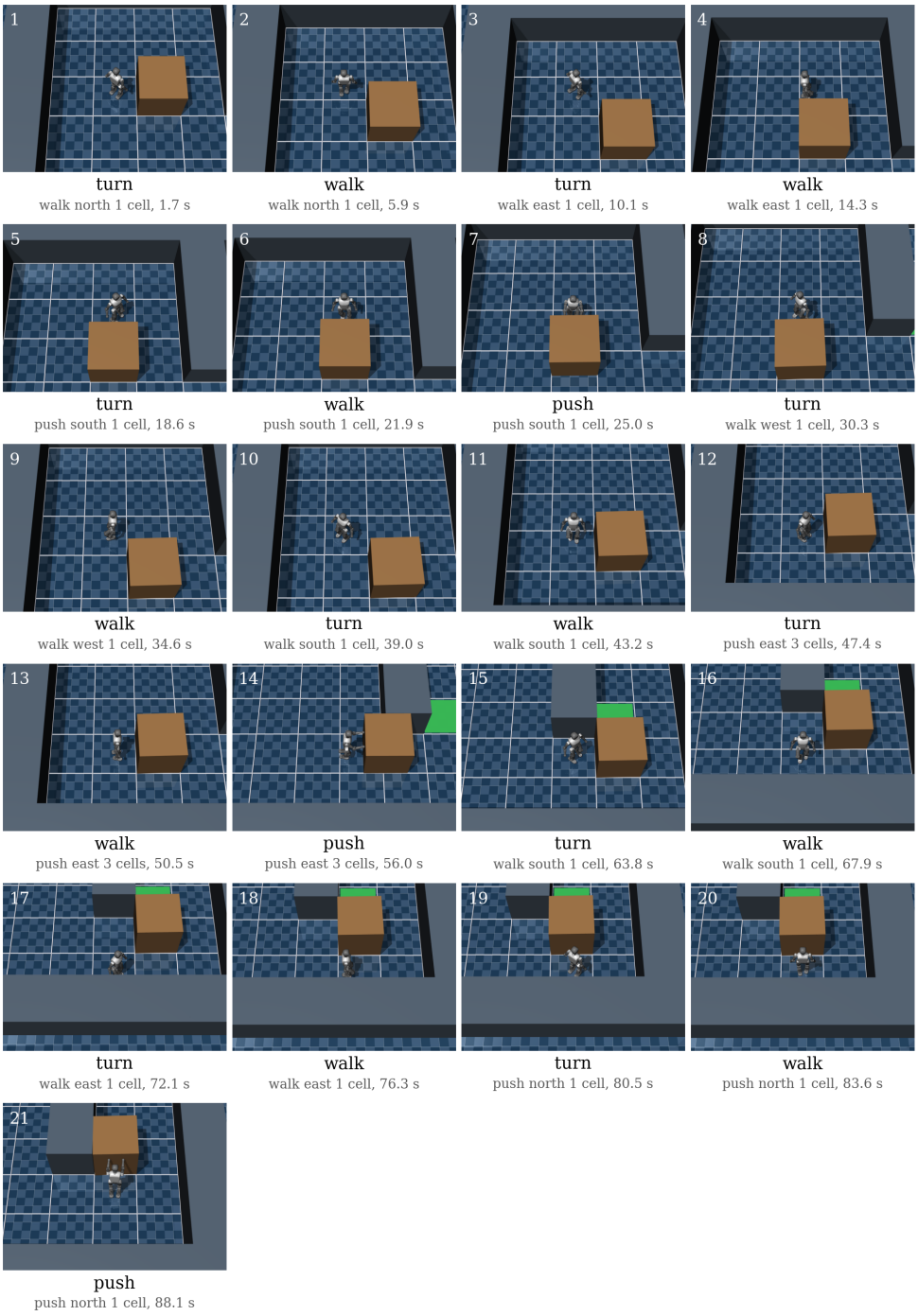


Figure 7.7. A run of the Sokoban demo.

7.2 Baselines

The comparison uses two baselines. The transition policies of Lee et al., *Composing Complex Skills by Learning Transition Policies* [13], leave the primitive skills frozen and insert a learned controller between them, trained on executions of the skills it connects. This provides a comparison with a dedicated transition mechanism. The instant switch removes that mechanism entirely and tests how the incoming skill behaves when it takes over directly from the source.

7.2.1 Transition Policy Architecture and Training

The first baseline reimplements the architecture of Lee et al.[2] It combines a transition policy trained with proximal policy optimization and a proximity predictor associated with the incoming skill. The predictor estimates how favorable the current state is for initiating that skill.

Training alternates between executing transitions and updating the two networks. After each handover, the frozen incoming skill is executed and its outcome labels the preceding transition as successful or failed. The predictor learns from these traces, with successful examples becoming more favorable towards handover; nominal primitive executions supply its initial examples. The transition policy learns from increases in predicted proximity and a terminal proximity bonus. The observed outcome therefore influences training through the predictor, without adding the continuous evaluation score to the policy's reward.

At deployment, the transition policy acts until its learned stopping decision hands control to the incoming skill, subject to a maximum length. The proximity predictor is used during training and is not needed to execute the trained controller.

A separate controller is trained for each of the three tested skill pairs. Each is specialized by the executions on which it is trained even though the architecture itself is only conditioned on the destination primitive, not on the source identity. During training, the point where the incoming skill should start is fixed arbitrarily in the scene. From walking, it lies ahead of the robot in the direction it faces at reset: the start of the jump is placed 3 m ahead, and the kick is placed so that its strike meets a ball 3 m ahead. From jumping, it lies 0.1 m beyond the end of the jump reference. The switch is requested when the robot is 0.2, 0.4 or 0.6 m before this point for walking to jumping and jumping to walking, and 0.4 m before it for walking to kicking, and the transition policy then has at most 2 s, or 100 control steps, to hand over. The policy observes neither the point nor the distance, which only set the starting conditions of each episode.

7.2.2 How a Transition Policy is used

The bridge receives a target state, the expected motion after that state, and a duration. When the switch is requested, the entry sequence of the incoming skill is placed a distance d in front of the robot, along its heading, and the bridge has a duration T to complete the motion. The distance and the duration are consequently

[2]The reference implementation is available at https://github.com/youngwoon/transition; the port follows commit `79ab8e4`.

parameters of its use. The tracker follows the generated transition until the scheduled handover, and the incoming skill continues from the resulting physical state.

The transition policy has no corresponding target-state, distance or duration input. When the switch is requested it takes over from the source skill and acts until it chooses to hand over, subject to its maximum length. Its training determines which states are favorable for the incoming skill, so where and when the incoming skill starts is decided implicitly by the policy. It need not reproduce the bridge's requested configuration and there is no search over the recorded entry points at deployment.

This distinction does not remove the conditioning required by the incoming G1 skill. The jump and the kick still need a reference phase and a placement of that reference in the scene, and the kick also needs the ball. From walking to jumping and from jumping to walking nothing is placed in advance: when the policy hands over, the reference of the incoming jump is placed on the robot, at its entry phase, and the walk has no reference at all. From walking to kicking this is not possible, because the kick has to end in a contact with a ball that is part of the scene. The evaluation then reproduces what happens during training. The ball is fixed 3 m ahead of the robot's starting pose, and the kick reference is placed so that its own ball coincides with the real one. The reference is frozen on its first frame while the walk and the transition policy run, so the kick's clock does not advance. The walk heads for the entry pose of the kick, and the transition policy takes over when the robot is within 0.4 m of it, the gap it was trained to fill. The policy can hand over before reaching the entry pose; the kick can then correct a moderate error through its own feedback.

7.2.3 Instant Switch (No-op)

The no-op baseline switches policies immediately when the request arrives. The last action before the request comes from the source skill, and the next control action comes from the incoming skill. There is no intermediate policy, waiting interval, interpolation or zero-action step. The robot retains its physical state and previous-action history; the incoming skill therefore observes the state actually left by the source.

At the request, the reference of the incoming jump or kick is placed on the robot at its entry phase, and the kick's ball is placed where that reference expects it. From the request onwards the robot continues under the incoming skill's actions. Measurement starts with its first action, so the transition duration is exactly zero.

This baseline distinguishes the benefit of preparing the robot from the incoming skill's own ability to recover from an unfamiliar state. A favorable score would show that direct takeover is sufficient under the tested condition; a poor score would identify a condition in which the source state is unsuitable for the incoming execution.

7.3 Metrics

The two learned architectures solve different intermediate problems. The bridge is asked to reach a specified target within a specified time; the transition policy

learns how to prepare the robot and when to relinquish control. The no-op baseline bypasses intermediate control altogether. Error with respect to the bridge's target would therefore measure an objective that only the bridge receives. Instead, the comparison measures what all three methods are intended to enable: the performance of the incoming skill after handover.

7.3.1 Measurement after the Handover

A trial starts with the source skill and ends after the incoming skill has been evaluated. From walking, the robot starts at rest and walks straight ahead at 0.8 m/s. The switch is requested after 3 s of walking plus a random interval between 0 and 1.25 s, about one gait cycle, drawn for each trial, so that the request falls at a different point of the gait in each trial. From jumping, the switch is requested at the first landing. The 50 requests of a pair are recorded once and replayed for the instant switch, for every bridge setting and for the transition policy, with one exception: from walking to kicking the transition policy is requested within 0.4 m of the fixed entry point, as described in Section 7.2.2. The robot is never moved after the request.

Measurements begin when the incoming skill first takes control, immediately at the request for the no-op baseline. The bridge or transition interval is excluded from the incoming measurement, but failure during that interval still invalidates the trial. Walking is evaluated for two seconds. Jumping and kicking are evaluated over the full remainder of their reference. The same physical measurement is obtained in nominal conditions by executing the incoming skill from its recorded entry point with matching conditioning and initial perturbations.

A handover far from the recorded entry point can receive a good score if the incoming skill performs well. Conversely, reaching the bridge's target closely can still receive a poor score if the following execution is unstable or ineffective. This follows the role of incoming-skill quality in Equation (4.2), using measured nominal performance as the reference.

7.3.2 Measures for the G1 Skill Pool

The quantitative study uses the G1, for two reasons. The G1 is much more capable than the T1 and can perform highly dynamic motions such as jumps and kicks. Secondly, more of the motion data is within its reach: the kinematic filter keeps 96% of the AMASS frames and 93% of the LAFAN1 frames on the G1, against 73% and 34% on the T1 (Section 5.2.3), so its bridge is trained on a larger and more varied corpus.

The three G1 skills require different kinds of control: walking follows a velocity command through a periodic gait, jumping tracks a dynamic motion beginning in a crouch, and kicking combines reference tracking with an external contact. Their performance therefore needs task-specific measurements. All three skills remain frozen, and the same bridge connects walking to jumping, walking to kicking and jumping to walking.

The T1 is considered in the qualitative Sokoban study. Its walking, turning and pushing skills begin from nearly stationary conditions, with more overlap in their acceptable starting states. They show repeated composition within a plan, while the

G1 pairs provide the controlled comparison of larger changes in posture, velocity and contact timing. Results measured on the G1 are not assumed to transfer numerically to the T1.

Walking. The command-following measurement has two components. With H observed control steps after the handover, the planar velocity and yaw-rate errors are

$$J_v = \frac{1}{H}\sum_{t=1}^{H}\left\|\mathbf{v}_{xy}^{\mathrm{body}}(t) - \mathbf{v}_{xy}^{\mathrm{cmd}}\right\|_2, \tag{7.1}$$

$$J_\omega = \frac{1}{H}\sum_{t=1}^{H}\left|\omega_z^{\mathrm{body}}(t) - \omega_z^{\mathrm{cmd}}\right|. \tag{7.2}$$

They are expressed in m/s and rad/s. The test commands forward walking with zero lateral and yaw velocity, using the same forward command in nominal and post-handover executions. The two components are kept separate in physical units and combined only after normalization.

Jumping. The measure is the mean Euclidean body-position error over the tracked bodies and observed control steps:

$$J_{\mathrm{jump}} = \frac{1}{H\,n_{\mathrm{b}}}\sum_{t=1}^{H}\sum_{b=1}^{n_{\mathrm{b}}}\left\|\mathbf{p}_b(t) - \widehat{\mathbf{p}}_b(t)\right\|_2. \tag{7.3}$$

Here n_{b} is the number of tracked bodies and $\widehat{\mathbf{p}}_b$ is the reference position used by the jumping task. Its planar position and heading are aligned with the robot's torso anchor, while its height comes from the reference. The measurement is in metres and smaller is better. The skill's native termination checks still reject excessive deviation from the placed reference, and the failure penalty prevents an early termination from receiving an artificially favorable tracking score.

Kicking. The kick is assessed by the peak forward speed of the ball after registered foot contact, within the incoming execution:

$$J_{\mathrm{kick}} = \max\Big(0,\ \max_{t\in\mathcal{T}_{\mathrm{contact}}}\mathbf{v}_{\mathrm{ball},xy}(t)\cdot\mathbf{u}_{\mathrm{kick},xy}\Big). \tag{7.4}$$

The set $\mathcal{T}_{\mathrm{contact}}$ contains the observed steps from the first contact onwards, and $\mathbf{u}_{\mathrm{kick},xy}$ is the direction of the kick reference placed in the scene. The measurement is in m/s and larger is better. It is zero if there is no contact. The skill has no commanded target ball velocity, so a velocity-error metric would introduce a target absent from its task.

A trial fails if its source fails, the transition fails or never hands over, the robot falls, or the incoming skill triggers a native failure condition. A jump must land. A kick must make foot–ball contact after handover and exceed 0.5 m/s of forward ball speed.

For an error measurement, where smaller is better, a failed execution receives at least an explicit penalty P. If a partial measurement J_i is available before failure, a larger observed error is retained:

$$\widetilde{J}_i = \begin{cases} J_i & \text{if trial } i \text{ is valid,} \\ \max(J_i, P) & \text{if it fails and } J_i \text{ is available,} \\ P & \text{if it fails before a measurement is available.} \end{cases} \tag{7.5}$$

The penalties are 2 m/s for walking velocity error, 2 rad/s for walking yaw-rate error, and 1 m for jump tracking error. For kicking, where larger ball speed is better, a failed trial receives $\widetilde{J}_i = 0$ even if a partial execution moved the ball. These values also apply to failed nominal executions. The penalties are part of the evaluation protocol and are kept unchanged across all three methods.

Every reported absolute mean is computed over $N = 50$ trials:

$$\overline{J} = \frac{1}{N}\sum_{i=1}^{N} \widetilde{J}_i, \qquad s_J = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\widetilde{J}_i - \overline{J})^2}. \tag{7.6}$$

The tables distinguish the mean and sample standard deviation where indicated.

7.3.3 Parameter Selection

The measurements above are also used to choose the parameters of the bridge. At the switch request the entry sequence is placed at a distance d in front of the robot and the bridge is given a duration T, and both change the performance of the incoming skill. Both are therefore swept: d from 0.2 to 1.0 m in steps of 0.1 m and T from 0.4 to 1.2 s in steps of 0.2 s. Each of the 45 combinations is scored over $N = 50$ trials for each pair. The selected setting (d^*, T^*) of a pair is the one with the best mean $\overline{J}$: the smallest body-position error for jumping and the largest ball speed for kicking. Walking has two measurements in different units, which are first normalized and then combined as in Equation (7.8). The selected settings are the ones used in the demos.

The sweep is reported in Section 7.4.2, after the nominal performance that the normalization requires. The parameters are selected on the same trials on which they are reported, so the selected score is the best value observed in the sweep, not an estimate on new trials.

7.3.4 Normalization

For measurement m, let $\overline{J}_{m,\text{nom}}$ and $\overline{J}_{m,A}$ be the nominal and post-handover means, including failure penalties. Relative performance is

$$R_{m,A} = \begin{cases} \overline{J}_{m,\text{nom}}/\overline{J}_{m,A} & \text{for an error measurement,} \\ \overline{J}_{m,A}/\overline{J}_{m,\text{nom}} & \text{for ball speed,} \end{cases} \qquad S_{m,A} = \min(1, R_{m,A}). \tag{7.7}$$

Jumping and kicking each have one component. Walking combines its normalized errors equally:

$$S_{\text{walk},A} = \frac{S_{v,A} + S_{\omega,A}}{2}. \tag{7.8}$$

7.3.5 Action Discontinuity

The measurements above judge the outcome of the incoming skill. They do not show how control passes from one policy to the next. Every policy commands joint-position targets, which the proportional-derivative controllers turn into torques. When the controlling policy changes, the target changes within one control step by as much as the two policies disagree about what to command in the same state, and the torque changes with it.

Let $\boldsymbol{\theta}_t^{\mathrm{cmd}}$ be the target commanded at control step t, after clipping, where $t = 0$ is the first step after the request. The change at step t is the root mean square over the $n_\theta = 23$ joints of the G1,

$$\Delta_t = \sqrt{\frac{1}{n_\theta} \sum_{j=1}^{n_\theta} \left(\theta_{t,j}^{\mathrm{cmd}} - \theta_{t-1,j}^{\mathrm{cmd}}\right)^2}, \tag{7.9}$$

in radians per control step of 0.02 s. A *seam* is a step at which the controlling policy changes. The bridge and the transition policy have two: the request, at $t = 0$, where they take over from the source skill, and the handover, where the incoming skill takes over. The instant switch has only one, at $t = 0$. Each trial records four values: Δ_t at the request, Δ_t at the handover, the largest change of a single joint at the handover, and the peak Δ_t over the whole trial, from the request to the end of the incoming measurement. Compared with the peak of the nominal skill, it shows whether the switch produces a larger change than the skill makes on its own: a seam as large as the fastest steps of the skill adds nothing unusual, while a seam well above them could be a problem.

All policies drive the same actuators with the same gains, so equal changes of the target are equal changes of the torque, whichever policy commands them. Every trial that reaches a step contributes to its mean, including trials that fail later. These values are not normalized and do not enter the scores of Section 7.3.4.

7.4 Results

This section reports how the three incoming skills perform in nominal conditions, which bridge parameters give the best handover, how well each skill performs after the baselines and after the bridge, how the three methods compare, using the definitions of Section 7.3, and how much training each of them needs. Across all three pairs the study contains 7,200 scored trials: 6,750 bridge trials, 150 transition-policy trials, 150 instant switches and 150 nominal executions. Every setting includes all 50 trials and their failure penalties. The action discontinuity of Section 7.4.7 repeats 600 of these trials, those of the nominal skills, of the two baselines and of the bridge at its selected setting, recording the commanded targets.

7.4.1 Nominal Performance

Table 7.1 reports the three incoming skills running in nominal conditions. Walking and jumping are measured by an error and kicking by a ball speed. Walking keeps a velocity error of 0.089 m/s and a yaw-rate error of 0.075 rad/s over its gait, the

jump follows its reference with a mean body-position error of 4.2 cm, and the kick sends the ball forward at 5.89 m/s.

Table 7.1. Nominal performance over 50 runs. All runs succeed. Values are mean ± sample standard deviation.

Skill	Measurement	Nominal performance
Walk	Planar velocity error	0.0888 ± 0.0015 m/s
Walk	Yaw-rate error	0.0752 ± 0.0041 rad/s
Jump	Body-position error	0.0423 ± 0.0003 m
Kick	Forward ball speed	5.890 ± 0.061 m/s

7.4.2 Bridge Parameter Selection

Tables 7.2 to 7.4 report the post-handover measurements for all 45 bridge settings of each pair, as described in Section 7.3.3. The selected setting is marked in bold. All the settings are evaluated keeping the entry point fixed (Table 6.1): the first one for walking and jumping, the fourth for kicking.

From walking to jumping, the bridge has to stop a walking robot in a crouch at a precise point in front of it. Close to the robot most durations work: at 0.2 m every duration of at least 0.6 s succeeds in all 50 trials, with an error between 0.056 and 0.058 m, while $T = 0.4$ s fails in 4 trials. With a larger distance, short durations fail. At 0.4 m, 30 of the 50 trials fail with $T = 0.4$ s and 7 with $T = 0.6$ s, while $T = 0.8$ and $T = 1.0$ s fail in one. From 0.5 m onwards $T = 1.2$ s is the best duration, with a success rate between 88% and 98% up to 0.8 m, 72% at 0.9 m and 56% at 1.0 m. Most failures are jumps that terminate on trunk orientation or do not land. The selected setting is $d^* = 0.2$ m and $T^* = 0.6$ s, with an error of 0.056 m against the nominal 0.042 m.

From walking to kicking, the ball speed stays close to nominal over most of the grid. With $T \geq 0.8$ s every trial succeeds at every distance, and the speed is at least 5.67 m/s. Only short durations at large distances degrade the kick: at 1.0 m, 7 trials fail with $T = 0.6$ s and 25 with $T = 0.4$ s, where the mean speed drops to 1.98 m/s. Most of these failures are missed balls. The selected setting is $d^* = 0.6$ m and $T^* = 0.8$ s, with 5.90 m/s, slightly above the nominal 5.89 m/s.

From jumping to walking, 3 of the 2,250 trials fail, each by a fall before the handover and all with $T = 1.2$ s. Both errors generally decrease with the duration, and the velocity error also decreases with the distance. With $T = 0.4$ s the velocity error is between 0.21 and 0.25 m/s at every distance. The six best settings all have a mean speed d/T between 0.67 and 1.0 m/s, close to the commanded walking speed of 0.8 m/s. This suggests that the walk performs best when the bridge hands it a robot already moving at about the speed it is asked to keep. The selected setting is $d^* = 1.0$ m and $T^* = 1.2$ s, with errors of 0.098 m/s and 0.083 rad/s against the nominal 0.089 m/s and 0.075 rad/s.

Table 7.2. Walking to jumping: mean body-position error in metres, over 50 runs per cell including failures. Smaller is better. Rows give the distance d and columns the duration T. The best score is in bold; several cells round to 0.056 m, and the bold one is the smallest before rounding, 0.0556 m.

$d\backslash T$	0.4 s	0.6 s	0.8 s	1.0 s	1.2 s
0.2 m	0.139	**0.056**	0.056	0.056	0.058
0.3 m	0.391	0.158	0.077	0.056	0.056
0.4 m	0.635	0.199	0.080	0.078	0.115
0.5 m	0.800	0.279	0.121	0.079	0.078
0.6 m	0.910	0.484	0.200	0.118	0.099
0.7 m	1.000	0.688	0.426	0.103	0.101
0.8 m	1.000	0.727	0.631	0.219	0.179
0.9 m	1.000	0.836	0.835	0.443	0.332
1.0 m	1.000	0.854	0.926	0.649	0.481

Table 7.3. Walking to kicking: mean peak forward ball speed in m/s, over 50 runs per cell including failures. Larger is better. Rows give the distance d and columns the duration T. The best score is in bold; 1.0 m with 1.2 s also rounds to 5.90 m/s, but is slightly lower before rounding.

$d\backslash T$	0.4 s	0.6 s	0.8 s	1.0 s	1.2 s
0.2 m	5.87	5.85	5.81	5.83	5.86
0.3 m	5.77	5.88	5.86	5.84	5.87
0.4 m	5.80	5.89	5.87	5.86	5.87
0.5 m	5.73	5.87	5.88	5.89	5.87
0.6 m	5.35	5.76	**5.90**	5.88	5.89
0.7 m	5.08	5.69	5.87	5.89	5.87
0.8 m	4.46	5.40	5.78	5.89	5.85
0.9 m	3.57	5.15	5.76	5.87	5.87
1.0 m	1.98	4.12	5.67	5.85	5.90

Table 7.4. Jumping to walking: each cell reports the mean planar velocity error in m/s followed by the mean yaw-rate error in rad/s, separated by a slash. Each mean includes all 50 runs and their failure penalties. Smaller is better for both components. Rows give the distance d and columns the duration T. The best score is in bold.

$d\backslash T$	0.4 s	0.6 s	0.8 s	1.0 s	1.2 s
0.2 m	0.252/0.206	0.184/0.107	0.161/0.116	0.156/0.107	0.196/0.144
0.3 m	0.228/0.141	0.167/0.096	0.149/0.108	0.144/0.103	0.144/0.103
0.4 m	0.213/0.141	0.159/0.097	0.136/0.104	0.131/0.104	0.131/0.103
0.5 m	0.228/0.164	0.161/0.131	0.131/0.101	0.123/0.099	0.158/0.140
0.6 m	0.236/0.177	0.155/0.156	0.122/0.104	0.115/0.097	0.110/0.096
0.7 m	0.234/0.186	0.160/0.182	0.120/0.105	0.123/0.098	0.139/0.130
0.8 m	0.235/0.204	0.159/0.190	0.110/0.100	0.102/0.095	0.099/0.090
0.9 m	0.233/0.220	0.156/0.194	0.111/0.105	0.102/0.090	0.098/0.088
1.0 m	0.230/0.218	0.154/0.196	0.110/0.109	0.105/0.092	**0.098/0.083**

7.4.3 Baselines after the Handover

Table 7.5 reports the post-handover measurements of the instant switch and of the transition policies. Neither baseline has a distance or a duration, and each is evaluated over 50 trials.

The transition policies stop very early. With their learned stopping rule they hand over after 0.10 to 0.18 s, or 5 to 9 control steps, from walking to jumping, and after 0.04 s, or 2 control steps, in the other two pairs, in every trial and well before the 2 s limit. A learned transition is therefore close to an instant switch preceded by a short corrective motion.

From walking to jumping, those few steps matter. The instant switch succeeds in 38 of the 50 trials. In the other 12 the jump terminates within 0.7 to 1.7 s of the switch, on trunk orientation or because the robot drifts too far from the reference. Even the successful jumps track with an error of 0.081 m on average, nearly twice the nominal 0.042 m. The transition policy succeeds in every trial, with an error of 0.062 m.

From walking to kicking, both baselines succeed in every trial. The instant switch sends the ball at 5.46 m/s and the transition policy at 5.66 m/s, against the nominal 5.89 m/s. The two are not measured from the same request. The instant switch takes over during the same timed walk as the bridge, with the reference and the ball placed on the robot at the switch, while the transition policy is asked 0.4 m before the fixed entry point, in the scene it was trained in (Section 7.2.2).

From jumping to walking, every trial succeeds for both baselines. The instant switch leaves a velocity error of 0.23 m/s and a yaw-rate error of 0.22 rad/s, between two and a half and three times the nominal values. The two steps of the transition policy after the landing increase them further, to 0.31 m/s and 0.33 rad/s.

Table 7.5. Post-handover measurements of the instant switch and the transition policy, as mean ± sample standard deviation over 50 trials including failures, with the success rate in percent in parentheses. The last column gives the time the transition policy takes before it hands over.

Transition	Measurement	No-op	Transition policy	Handover
Walk to jump	Body-position error (m)	0.301 ± 0.397 (76)	0.062 ± 0.006 (100)	0.10–0.18 s
Walk to kick	Ball speed (m/s)	5.46 ± 0.25 (100)	5.66 ± 0.21 (100)	0.04 s
Jump to walk	Velocity error (m/s)	0.229 ± 0.013 (100)	0.315 ± 0.025 (100)	0.04 s
	Yaw-rate error (rad/s)	0.222 ± 0.019	0.326 ± 0.025	

7.4.4 Bridge after the Handover

Table 7.6 reports the bridge at its selected parameters. All 150 trials succeed. The kick reaches a ball speed slightly above nominal. The jump error is 13 mm larger than nominal, and both walking errors are about 10% larger than nominal.

The distance and the duration are choices of the controller that calls the bridge; the baselines have neither. To show how much the bridge depends on the distance, Table 7.7 reports, for each distance, the best duration T_d^* and its measurements. From walking to kicking every trial succeeds at every distance, with at least 5.87 m/s. From jumping to walking the velocity error generally decreases with the distance,

Table 7.6. Bridge at the selected parameters (d^*, T^*). Each mean and sample standard deviation includes 50 trials, all successful.

Transition	d^*	T^*	Measurement	Mean ± SD
Walk to jump	0.2 m	0.6 s	Body-position error	0.056 ± 0.008 m
Walk to kick	0.6 m	0.8 s	Forward ball speed	5.90 ± 0.10 m/s
Jump to walk	1.0 m	1.2 s	Planar velocity error	0.098 ± 0.009 m/s
			Yaw-rate error	0.083 ± 0.010 rad/s

from 0.156 m/s at 0.2 m to 0.098 m/s at 1.0 m. From walking to jumping the error stays below 0.06 m up to 0.3 m and then grows, to 0.481 m at 1.0 m, where even the longest duration fails in 22 of the 50 trials.

Table 7.7. Bridge with the best duration T_d^* at each distance d. Measurements are means over 50 trials including failures, in the same units as Table 7.5, with the success rate in percent in parentheses.

	Walk to jump		Walk to kick		Jump to walk	
d	T_d^*	Error	T_d^*	Ball speed	T_d^*	Errors
0.2 m	0.6 s	0.056 (100)	0.4 s	5.87 (100)	1.0 s	0.156/0.107 (100)
0.3 m	1.0 s	0.056 (100)	0.6 s	5.88 (100)	1.2 s	0.144/0.103 (100)
0.4 m	1.0 s	0.078 (98)	0.6 s	5.89 (100)	1.2 s	0.131/0.103 (100)
0.5 m	1.2 s	0.078 (98)	1.0 s	5.89 (100)	1.0 s	0.123/0.099 (100)
0.6 m	1.2 s	0.099 (96)	0.8 s	5.90 (100)	1.2 s	0.110/0.096 (100)
0.7 m	1.2 s	0.101 (96)	1.0 s	5.89 (100)	1.0 s	0.123/0.098 (100)
0.8 m	1.2 s	0.179 (88)	1.0 s	5.89 (100)	1.2 s	0.099/0.090 (100)
0.9 m	1.2 s	0.332 (72)	1.2 s	5.87 (100)	1.2 s	0.098/0.088 (100)
1.0 m	1.2 s	0.481 (56)	1.2 s	5.90 (100)	1.2 s	0.098/0.083 (100)

7.4.5 Comparison

The comparison normalizes the measurements of Tables 7.5 and 7.7 with the nominal performance, as in Equations (7.7) and (7.8). Each baseline has a single score per pair. Table 7.8 reports the score of the bridge with its best duration at every distance, next to the scores of the two baselines. Table 7.9 summarizes the bridge in two ways: at its selected setting (d^*, T^*), and as a mean over the nine distances, which does not depend on the choice of d^*.

At its selected setting the bridge has the highest score on all three pairs. The difference is largest from jumping to walking, where neither baseline prepares the walk: the bridge can bring the robot towards the commanded speed before the handover, and its score is 0.91 against 0.36 for the instant switch and 0.26 for the transition policy. This does not depend on the tuning, since 43 of the 45 bridge settings score above the instant switch and all 45 above the transition policy.

From walking to jumping, the transition policy is a strong baseline. It scores 0.69 against 0.76 for the bridge at its selected setting, and only 6 of the 45 bridge settings are better, all with the entry point at most 0.3 m ahead and at least 0.6 s to reach it. With its best duration the bridge scores 0.54 at 0.4 and 0.5 m and less beyond, so its mean over the distances, 0.43, is below the transition policy. The

Table 7.8. Normalized scores of the bridge with its best duration T_d^* at each distance d, and of the two baselines, which have no distance. All scores include failures. Higher is better, up to one.

	Walk to jump	Walk to kick	Jump to walk
Bridge, 0.2 m	0.760	0.997	0.635
Bridge, 0.3 m	0.754	0.999	0.674
Bridge, 0.4 m	0.541	1.000	0.705
Bridge, 0.5 m	0.542	1.000	0.742
Bridge, 0.6 m	0.426	1.000	0.798
Bridge, 0.7 m	0.419	1.000	0.745
Bridge, 0.8 m	0.236	1.000	0.867
Bridge, 0.9 m	0.128	0.997	0.882
Bridge, 1.0 m	0.088	1.000	0.906
No-op	0.140	0.927	0.363
Transition policy	0.685	0.961	0.257

Table 7.9. Normalized performance of the three methods. Higher is better, up to one. The bridge is reported at its selected setting (d^*, T^*) and as the mean of its scores in Table 7.8 over the nine distances.

			Bridge		
Transition	No-op	Transition	(d^*, T^*)	At (d^*, T^*)	Mean over distances
Walk to jump	0.140	0.685	(0.2 m, 0.6 s)	0.760	0.433
Walk to kick	0.927	0.961	(0.6 m, 0.8 s)	1.000	0.999
Jump to walk	0.363	0.257	(1.0 m, 1.2 s)	0.906	0.773

instant switch completes 76% of the jumps but scores only 0.14, below 24 of the 45 bridge settings.

From walking to kicking, all three methods kick well. The instant switch scores 0.93 and the transition policy 0.96, while the bridge reaches the nominal ball speed at every distance with its best duration, scoring at least 0.997. Only short durations at large distances fall below both baselines, 0.4 s from 0.6 m and 0.6 s from 0.8 m, in 8 of the 45 settings.

The bridge pays for this with time. At the selected settings it takes 0.6 to 1.2 s before the handover, while the transition policies take at most 0.18 s and the instant switch none. The scores also have the limitation stated in Section 7.3.3: d^*, T^* and T_d^* are selected on the same trials on which they are reported, which favors the bridge. The baselines have no parameter to select, apart from the training of the transition policy. Finally, the walk-to-kick transition policy starts from its own request, in the scene in which it was trained, rather than from the timed walk shared by the other methods (Section 7.3.1).

7.4.6 Training Cost

The three methods also differ in how much training they need (Table 7.10). The instant switch needs none. A transition policy is trained for one pair of skills, on

executions of those skills, so every pair needs its own. The bridge is trained once, on the motion corpus, and the same planner and tracker serve all three pairs.

Table 7.10. Training needed by each method. Steps are control steps at 50 Hz, summed over the parallel environments; the planner is trained without simulation. Times are wall-clock hours, all on one NVIDIA GeForce RTX 5070 Ti.

Method		Steps (millions)	Time (h)
Instant switch	No training	–	–
Transition policy	Walk to jump	19.3	4.3
	Walk to kick	16.7	3.8
	Jump to walk	16.3	3.0
	Total, three pairs	52.3	11.1
Bridge	Planner	–	1.9
	Tracker	1966	16.1
	Total, any pair	1966	18.0

On the three pairs tested here, the bridge takes longer to train than the three transition policies together, 18.0 h against 11.1 h, and many more steps, since its tracker learns to follow the whole corpus with 4096 parallel environments. Its cost, however, is paid once. Every new pair needs a new transition policy, about 3.7 h on average, and the two costs are equal at about five pairs: the three G1 skills of this chapter already form six ordered pairs, of which only three are tested. A transition policy also has to be trained again whenever one of its skills changes, while the bridge never sees the skills.

7.4.7 Action Discontinuity

Table 7.11 reports the changes of the commanded targets defined in Section 7.3.5, for the nominal skills, the two baselines and the bridge at its selected setting.

At the handover the bridge commands the smallest change on all three pairs: 0.235, 0.129 and 0.158 rad from walking to jumping, walking to kicking and jumping to walking, against 0.350, 0.333 and 0.506 rad for the instant switch and 0.566, 0.577 and 0.537 rad for the transition policy. The largest single joint shows the same order. After the bridge it moves by 0.40 to 0.57 rad on average, while after either baseline it moves by 0.92 to 1.53 rad within 0.02 s.

The bridge has a second seam at the request, where the tracker takes over from the source skill. From walking to jumping and from walking to kicking it is close to the handover, 0.268 and 0.179 rad. From jumping to walking it is the largest change of the trial, 0.423 rad against 0.158 rad at the handover. Even so, each of the two seams of the bridge is smaller than the single seam of the instant switch on every pair.

From walking to kicking, the peak of a bridge trial, 0.197 rad, is close to the largest step of the nominal kick, 0.175 rad, so the bridge adds no step clearly larger than those of the kick itself. The same holds from walking to jumping, 0.295 against 0.265 rad.

Overall, the bridge passes control with the smallest changes of the three methods

Table 7.11. Change of the commanded joint-position targets in one control step, root mean square over the joints in radians (Equation (7.9)), as the mean over 50 trials. Request and handover are the seams at which the controlling policy changes; the instant switch has one, reported under handover, and the nominal skills have none. Largest joint is the largest change of a single joint at the handover, and peak is the largest change from the request to the end of the trial.

Transition	Method	Request	Handover	Largest joint	Peak
Walk to jump	Nominal	–	–	–	0.265
	No-op	–	0.350	1.009	0.452
	Transition policy	0.455	0.566	1.336	0.567
	Bridge (0.2 m, 0.6 s)	0.268	0.235	0.574	0.295
Walk to kick	Nominal	–	–	–	0.175
	No-op	–	0.333	0.919	0.335
	Transition policy	1.062	0.577	1.483	1.062
	Bridge (0.6 m, 0.8 s)	0.179	0.129	0.402	0.197
Jump to walk	Nominal	–	–	–	0.107
	No-op	–	0.506	1.532	0.506
	Transition policy	0.893	0.537	1.505	0.893
	Bridge (1.0 m, 1.2 s)	0.423	0.158	0.444	0.433

at both seams and on all three pairs. As in the original method, the reward of the transition policies has no term on the change of the action (Section 7.2.1).

Chapter 8

Conclusions

8.1 Summary

This thesis studied the sequential composition of whole-body skills of a humanoid robot, and asked whether a single policy, learned without access to the skills it connects, can bring the robot within a given time from the state in which one frozen skill ends to a state from which another frozen skill can continue. The switch was formulated in the options framework as a request made of the recent motion of the robot, the motion that the incoming skill is expected to perform from the target, and a deadline (Chapter 4). The expected motion is not a path for the bridge to follow, it rather conveys what a single target state cannot. The bridge was learned from a corpus of human motion retargeted to the robot rather than from the skills themselves (Section 5.2). The selection of the target was separated from reaching it, and replaced by a proxy based on entry points recorded from the incoming skill (Section 5.1).

Three architectures were developed on the same corpus (Chapter 5): a goal-conditioned policy trained with reinforcement learning, a conditional variational autoencoder distilled from a tracking teacher, and a diffusion planner whose plans are executed by a universal tracker, each one addressing the limits of the previous one.

Only the last architecture was evaluated quantitatively, on three switches between frozen skills of the Unitree G1, against an instant switch and a transition policy trained for each pair, and inside two complete tasks (Chapters 6 and 7). The same bridge, trained once and only on retargeted human motion, connected all three pairs.

These results have clear limits. The score of the bridge depends on the distance and the duration of the request: from walking to jumping it beats the transition policy only when the entry point is at most 0.3 m ahead, and its mean over the distances, 0.43, is below it. The selected settings were chosen on the same trials on which they are reported. The bridge also needs 0.6 to 1.2 s before the handover, while the transition policies need at most 0.18 s. A bridge learned only from human motion can therefore connect frozen skills it has never seen, at least for the skills and conditions tested. How to choose the distance and the duration of each request, which here were hand-picked, remains the most important question left open (Section 8.2).

8.2 Future Work

The bridge can connect a new skill without training only if the corpus contains motion similar to the states of that skill, a condition that this thesis states only informally. If the corpus contains no crouch, for example, nothing guarantees that the bridge can reach the crouch that precedes a jump. A measure of how well the corpus covers a request would tell in advance whether a switch can be trusted, and when the corpus has to be extended for a new skill. Together with a measure of the training effort as the pool grows, it would also make precise when a single bridge is cheaper than a policy for each pair of skills.

The corpus itself could be extended to the motions discarded during selection (Section 5.2.1), such as fighting, dancing, falling and getting up, that were excluded because a switch rarely requires them and they could make the bridge less reliable on ordinary transitions. A curriculum that adds them gradually, once the bridge is reliable on the ordinary transitions, could widen the set of skills it can connect without degrading the performances.

The diffusion planner addresses the averaging of the first architecture, since it can represent several motions for the same request, but its plans are only kinematic: nothing ensures that the tracker executes them exactly, and any error of the tracker at the deadline is an error at the handover (Section 5.5.5). This error could be reduced by closing the loop between the two parts and training them together, so that the planner learns which motions the tracker can execute and the tracker learns to follow the plans the planner actually generates. Scoring the bridge during training by how well the incoming skill performs after the handover, as it is scored at test time, would bring training closer to deployment as well.

The selection of the target, replaced here by a proxy, is the other half of the problem. A learned estimate of the good set of each skill, or of the success of the bridge for a given request, would let the controller choose the entry point and the duration of each switch, and how far before the entry point to call the bridge, since the robot needs some room to change its velocity and posture, instead of taking them from a grid of tested values.

Finally, all the experiments were carried out in simulation. Deploying the bridge and the skills on a real robot is a natural next step, and would show how the bridge behaves on real hardware.

Bibliography

[1] Ziwen Zhuang, Shenzhe Yao, and Hang Zhao. "Humanoid Parkour Learning". In: *Conference on Robot Learning (CoRL)*. 2024.

[2] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning Quadrupedal Locomotion over Challenging Terrain". In: *Science Robotics* 5.47 (2020), eabc5986. DOI: 10.1126/scirobotics.abc5986.

[3] Zhiyuan Xu, Mingyo Seo, Daniel Lee, Hao Fu, Jiaheng Hu, Jiajun Cui, Yifeng Jiang, Zhe Wang, Adam Brund, Joydeep Biswas, and Peter Stone. "Learning Agile Striker Skills for Humanoid Soccer Robots from Noisy Sensory Input". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2026. arXiv: 2512.06571.

[4] Russ Tedrake, Ian R. Manchester, Mark Tobenkin, and John W. Roberts. "LQR-Trees: Feedback Motion Planning via Sums-of-Squares Verification". In: *The International Journal of Robotics Research* 29.8 (2010), pp. 1038–1052. DOI: 10.1177/0278364910369189.

[5] Anirudha Majumdar and Russ Tedrake. "Funnel Libraries for Real-Time Robust Feedback Motion Planning". In: *The International Journal of Robotics Research* 36.8 (2017), pp. 947–982. DOI: 10.1177/0278364917712421.

[6] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2021, pp. 91–100.

[7] Yuxuan Kuang, Haoran Geng, Amine Elhafsi, Tan-Dzung Do, Pieter Abbeel, Jitendra Malik, Marco Pavone, and Yue Wang. "SkillBlender: Towards Versatile Humanoid Whole-Body Loco-Manipulation via Skill Blending". In: *arXiv preprint arXiv:2506.09366* (2025). arXiv: 2506.09366.

[8] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112.1–2 (1999), pp. 181–211. DOI: 10.1016/S0004-3702(99)00052-1.

[9] Youngwoon Lee, Joseph J. Lim, Anima Anandkumar, and Yuke Zhu. "Adversarial Skill Chaining for Long-Horizon Robot Manipulation via Terminal State Regularization". In: *Conference on Robot Learning (CoRL)*. 2021. arXiv: 2111.07999.

[10] Zixuan Chen, Ze Ji, Jing Huo, and Yang Gao. "SCaR: Refining Skill Chaining for Long-Horizon Robotic Manipulation via Dual Regularization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 37. 2024.

[11] Tao Huang, Kai Chen, Wang Wei, Jianan Li, Yonghao Long, and Qi Dou. "Value-Informed Skill Chaining for Policy Learning of Long-Horizon Tasks with Surgical Robot". In: *arXiv preprint arXiv:2307.16503* (2023). arXiv: 2307.16503.

[12] Yucheng Xin, Jiacheng Bao, Yubo Dong, Xueqian Wang, Bin Zhao, Xuelong Li, Junbo Tan, and Dong Wang. "RPG: Robust Policy Gating for Smooth Multi-Skill Transitions in Humanoid Fighting". In: *arXiv preprint arXiv:2604.21355* (2026). arXiv: 2604.21355.

[13] Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S. Hu, and Joseph J. Lim. "Composing Complex Skills by Learning Transition Policies". In: *International Conference on Learning Representations (ICLR)*. 2019.

[14] Ju-Seung Byun and Andrew Perrault. "Training Transition Policies via Distribution Matching for Complex Tasks". In: *International Conference on Learning Representations (ICLR)*. 2022.

[15] Brendan Tidd, Nicolas Hudson, Akansel Cosgun, and Jurgen Leitner. "Learning Setup Policies: Reliable Transition Between Locomotion Behaviours". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11958–11965. DOI: 10.1109/LRA.2022.3207567.

[16] Kevin Zakka, Qiayuan Liao, Brent Yi, Louis Le Lay, Koushil Sreenath, and Pieter Abbeel. "mjlab: A Lightweight Framework for GPU-Accelerated Robot Learning". In: *arXiv preprint arXiv:2601.22074* (2026). arXiv: 2601.22074.

[17] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: MIT Press, 2018.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-Level Control Through Deep Reinforcement Learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[19] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: *arXiv preprint arXiv:1703.03864* (2017). arXiv: 1703.03864.

[20] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 12. 1999, pp. 1057–1063.

[21] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8.3–4 (1992), pp. 229–256. DOI: 10.1007/BF00992696.

[22] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. "Trust Region Policy Optimization". In: *International Conference on Machine Learning (ICML)*. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 1889–1897.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347* (2017). arXiv: 1707.06347.

[24] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *International Conference on Learning Representations (ICLR)*. 2016. arXiv: 1506.02438.

[25] Leslie Pack Kaelbling. "Learning to Achieve Goals". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 1993, pp. 1094–1099.

[26] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *International Conference on Machine Learning (ICML)*. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 1312–1320.

[27] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. 2017.

[28] Andrew Y. Ng, Daishi Harada, and Stuart Russell. "Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping". In: *International Conference on Machine Learning (ICML)*. 1999, pp. 278–287.

[29] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. "Time Limits in Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4045–4054.

[30] Peter Dayan and Geoffrey E. Hinton. "Feudal Reinforcement Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 5. 1993.

[31] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. "Data-Efficient Hierarchical Reinforcement Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31. 2018.

[32] David Abel, Nate Umbanhowar, Khimya Khetarpal, Dilip Arumugam, Doina Precup, and Michael Littman. "Value Preserving State-Action Abstractions". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1639–1650.

[33] João Pedro Araújo, Yanjie Ze, Pei Xu, Jiajun Wu, and C. Karen Liu. "Retargeting Matters: General Motion Retargeting for Humanoid Motion Tracking". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2026, pp. 6097–6104. arXiv: 2510.02252.

[34] Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. “Robust Motion In-Betweening”. In: *ACM Transactions on Graphics* 39.4 (2020). DOI: 10.1145/3386569.3392480.

[35] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. “AMASS: Archive of Motion Capture as Surface Shapes”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[36] Carnegie Mellon University Graphics Lab. *CMU Graphics Lab Motion Capture Database*. http://mocap.cs.cmu.edu. 2003.

[37] Christian Mandery, Ömer Terlemez, Martin Do, Nikolaus Vahrenkamp, and Tamim Asfour. “The KIT Whole-Body Human Motion Database”. In: *International Conference on Advanced Robotics (ICAR)*. 2015, pp. 329–336.

[38] Advanced Computing Center for the Arts and Design. *ACCAD MoCap Dataset*. https://accad.osu.edu/research/motion-lab/mocap-system-and-data. 2018.

[39] Nikolaus F. Troje. “Decomposing Biological Motion: A Framework for Analysis and Synthesis of Human Gait Patterns”. In: *Journal of Vision* 2.5 (2002), pp. 371–387.

[40] Abhinanda R. Punnakkal, Arjun Chandrasekaran, Nikos Athanasiou, Alejandra Quirós-Ramírez, and Michael J. Black. “BABEL: Bodies, Action and Behavior with English Labels”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[41] Tairan He, Jiawei Gao, Wenli Xiao, Yuanhang Zhang, Zi Wang, Jiashun Wang, Zhengyi Luo, Guanqi He, Nikhil Sobanbabu, Chaoyi Pan, Zeji Yi, Guannan Qu, Kris Kitani, Jessica Hodgins, Linxi "Jim" Fan, Yuke Zhu, Changliu Liu, and Guanya Shi. “ASAP: Aligning Simulation and Real-World Physics for Learning Agile Humanoid Whole-Body Skills”. In: *Robotics: Science and Systems (RSS)*. 2025.

[42] Kong et al. “Learning Soccer Skills for Humanoid Robots: A Progressive Perception-Action Framework”. In: *arXiv preprint arXiv:2602.05310* (2026). arXiv: 2602.05310.

[43] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. “DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills”. In: *ACM Transactions on Graphics* 37.4 (2018), 143:1–143:18. DOI: 10.1145/3197517.3201311.

[44] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. “AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control”. In: *ACM Transactions on Graphics* 40.4 (2021). DOI: 10.1145/3450626.3459670.

[45] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. “ASE: Large-Scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters”. In: *ACM Transactions on Graphics* 41.4 (2022). DOI: 10.1145/3528223.3530110.

[46] Zhengyi Luo, Jinkun Cao, Alexander Winkler, Kris Kitani, and Weipeng Xu. "Perpetual Humanoid Control for Real-time Simulated Avatars". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023. arXiv: 2305.06456.

[47] Zhengyi Luo, Jinkun Cao, Josh Merel, Alexander Winkler, Jing Huang, Kris Kitani, and Weipeng Xu. "Universal Humanoid Motion Representations for Physics-Based Control". In: *International Conference on Learning Representations (ICLR)*. 2024. arXiv: 2310.04582.

[48] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "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*. Vol. 15. Proceedings of Machine Learning Research. PMLR, 2011, pp. 627–635. URL: https://proceedings.mlr.press/v15/ross11a.html.

[49] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507.

[50] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2014.

[51] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations (ICLR)*. 2017.

[52] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. "Generating Sentences from a Continuous Space". In: *SIGNLL Conference on Computational Natural Language Learning (CoNLL)*. 2016, pp. 10–21.

[53] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 28. 2015.

[54] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *International Conference on Machine Learning (ICML)*. Vol. 37. Proceedings of Machine Learning Research. PMLR, 2015, pp. 2256–2265.

[55] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020.

[56] Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Daniel Cohen-Or, and Amit H. Bermano. "Human Motion Diffusion Model". In: *International Conference on Learning Representations (ICLR)*. 2023.

[57] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations (ICLR)*. 2021.

[58] Jiaming Song, Chenlin Meng, and Stefano Ermon. "Denoising Diffusion Implicit Models". In: *International Conference on Learning Representations (ICLR)*. 2021.

[59] Prafulla Dhariwal and Alexander Nichol. "Diffusion Models Beat GANs on Image Synthesis". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. 2021.

[60] Jonathan Ho and Tim Salimans. "Classifier-Free Diffusion Guidance". In: *NeurIPS Workshop on Deep Generative Models and Downstream Applications*. 2021.

[61] Michael Janner, Yilun Du, Joshua B. Tenenbaum, and Sergey Levine. "Planning with Diffusion for Flexible Behavior Synthesis". In: *International Conference on Machine Learning (ICML)*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022.

[62] Robert R. Burridge, Alfred A. Rizzi, and Daniel E. Koditschek. "Sequential Composition of Dynamically Dexterous Robot Behaviors". In: *The International Journal of Robotics Research* 18.6 (1999), pp. 534–555. DOI: 10.1177/02783649922066385.

[63] George Konidaris and Andrew G. Barto. "Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 22. 2009.

[64] Akhil Bagaria and George Konidaris. "Option Discovery using Deep Skill Chaining". In: *International Conference on Learning Representations (ICLR)*. 2020.

[65] Yuanpei Chen, Chen Wang, Li Fei-Fei, and C. Karen Liu. "Sequential Dexterity: Chaining Dexterous Policies for Long-Horizon Manipulation". In: *Conference on Robot Learning (CoRL)*. 2023.

[66] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. "Motion Graphs". In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 2002, pp. 473–482.

[67] Setareh Cohan, Guy Tevet, Daniele Reda, Xue Bin Peng, and Michiel van de Panne. "Flexible Motion In-betweening with Diffusion Models". In: *ACM SIGGRAPH Conference Papers*. 2024. arXiv: 2405.11126.

[68] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, Tommi S. Jaakkola, and Pulkit Agrawal. "Is Conditional Generative Modeling all you need for Decision-Making?" In: *International Conference on Learning Representations (ICLR)*. 2023. arXiv: 2211.15657.

[69] Qiayuan Liao, Takara E. Truong, Xiaoyu Huang, Yuman Gao, Guy Tevet, Koushil Sreenath, and C. Karen Liu. "BeyondMimic: From Motion Tracking to Versatile Humanoid Control via Guided Diffusion". In: *Science Robotics* 11.117 (2026), eadx8924. DOI: 10.1126/scirobotics.adx8924. arXiv: 2508.08241.

[70] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Richard Forbes. "DReCon: Data-Driven Responsive Control of Physics-Based Characters". In: *ACM Transactions on Graphics* 38.6 (2019). DOI: 10.1145/3355089.3356536.

[71] Ye Yuan, Jiaming Song, Umar Iqbal, Arash Vahdat, and Jan Kautz. "PhysDiff: Physics-Guided Human Motion Diffusion Model". In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023. arXiv: 2212.02500.

[72] Guy Tevet, Sigal Raab, Setareh Cohan, Daniele Reda, Zhengyi Luo, Xue Bin Peng, Amit H. Bermano, and Michiel van de Panne. "CLoSD: Closing the Loop between Simulation and Diffusion for Multi-Task Character Control". In: *International Conference on Learning Representations (ICLR)*. 2025. arXiv: 2410.03441.

[73] Weiji Xie, Jiakun Zheng, Jinrui Han, Jiyuan Shi, Weinan Zhang, Chenjia Bai, and Xuelong Li. "TextOp: Real-time Interactive Text-Driven Humanoid Robot Motion Generation and Control". In: *arXiv preprint arXiv:2602.07439* (2026). arXiv: 2602.07439.

[74] Chen Tessler, Yunrong Guo, Ofir Nabati, Gal Chechik, and Xue Bin Peng. "MaskedMimic: Unified Physics-Based Character Control Through Masked Motion Inpainting". In: *ACM Transactions on Graphics* 43.6 (2024), 209:1–209:21. DOI: 10.1145/3687951. arXiv: 2409.14393.

[75] Bruno C. da Silva, George Konidaris, and Andrew G. Barto. "Learning Parameterized Skills". In: *Proceedings of the 29th International Conference on Machine Learning (ICML)*. 2012.

[76] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. "RePaint: Inpainting using Denoising Diffusion Probabilistic Models". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 11461–11471.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. 2017.

[78] Alexander Quinn Nichol and Prafulla Dhariwal. "Improved Denoising Diffusion Probabilistic Models". In: *International Conference on Machine Learning (ICML)*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 8162–8171.

[79] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. "Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments". In: *IEEE Robotics and Automation Letters* 8.6 (2023), pp. 3740–3747. DOI: 10.1109/LRA.2023.3270034.

[80] Google DeepMind. *MuJoCo Warp: A GPU-Accelerated Version of MuJoCo*. https://github.com/google-deepmind/mujoco_warp. 2025.

[81] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A Physics Engine for Model-Based Control". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[82] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. "RoboCup: The Robot World Cup Initiative". In: *International Conference on Autonomous Agents (AGENTS)*. ACM, 1997, pp. 340–347. DOI: 10.1145/267658.267738.

[83] Andrea Micheli, Arthur Bit-Monnot, Gabriele Röger, Enrico Scala, Alessandro Valentini, Luca Framba, Alberto Rovetta, Alessandro Trapasso, Luigi Bonassi, Alfonso Emilio Gerevini, Luca Iocchi, Felix Ingrand, Uwe Köckemann, Fabio Patrizi, Alessandro Saetti, Ivan Serina, and Sebastian Stock. "Unified Planning: Modeling, Manipulating and Solving AI Planning Problems in Python". In: *SoftwareX* 29 (2025), p. 102012.