Master thesis

Adversarial Skill Embeddings applications in humanoid robotics

Philippe Gratias-Quiquandon

Master MVA – ENS Paris-Saclay / Télécom Paris

Internship Location: CNRS-AIST Joint Robotics Laboratory (JRL), Tsukuba, Japan
Supervisor: Mitsuharu Morisawa
Dates: From 07/04/2025 to 07/10/2025

October 1, 2025

Contents

Acknowledgements

I would like to express my gratitude to Stéphane Caron, my robotics teacher, for his support in helping me secure this internship in Japan. I am also grateful to Mitsuharu Morisawa for supervising my internship. My sincere thanks go to Romain Gambardella, Pierre-Alexandre Leziart, Mehdi Benallegue, Anatole Forgiel, Thomas Duvinage, Léo Moussafir, Hippolyte Leroy, Bastien Murraciolli, and Naoko Sakai for their guidance and assistance throughout my internship. Finally, I would like to thank the entire laboratory for being such welcoming colleagues and for fostering a truly supportive and collaborative team spirit.

Figure 1: The JRL team

Chapter 1

Introduction

Reinforcement learning (RL) has become a revolution in robotics, offering a powerful way to train agents to perform complex movements by interacting with their environment. Unlike traditional methods like Model Predictive Control (MPC), RL is more flexible and robust, especially in tasks like humanoid locomotion that involve many joints and unpredictable contact. RL can produce more dynamic and natural motions because it doesn't rely on simplified models.

However, this flexibility also brings challenges. RL depends on well-designed reward functions to guide learning, and creating these can be difficult, especially for complex tasks. If rewards are too weak, the agent may learn unstable or unsafe behaviors. If they are too strong or too strict, learning can become stuck or fail altogether.

To overcome this, imitation learning provides a practical alternative. Rather than relying on dozens of manually designed rewards, the agent learns directly from expert demonstrations, often obtained from human motion capture. This reduces the need for complex reward design and gives results closer to real human-like motion.

A central goal in all forms of learning is generalization. In robotics, this means adapting to different tasks, motions, or environments without needing to redesign or retrain controllers for each case. One of the hopes with RL is precisely this: that a policy trained on a wide range of motions could generalize well enough to replace hand-crafted solutions like MPC, which are typically designed for very specific tasks.

While RL can perform very well in scenarios such as locomotion over rough terrain, it is still impossible to generalize across various tasks. This limitation comes from the fact that RL policies are tightly coupled to their reward functions. If the task changes, the rewards must be redesigned, and the entire training process typically has to start over. The goal of this internship was to explore more generalizable solutions through imitation learning, which offers the potential to learn diverse behaviors without needing to redefine rewards for every new task. The resources (videos, code) associated with this work can be accessed at https://github.com/Planeurzik/jrl-internship-repo.

1.1 CNRS-AIST JRL presentation

The Centre National de la Recherche Scientifique (CNRS) is France's leading public research organization, founded in 1939. As the largest fundamental science agency in Europe, CNRS employs over 34,000 staff, including nearly 29,400 scientists and supports more than 1,100 research laboratories across France and abroad. Its mission spans all scientific disciplines, from life sciences and physics to social sciences and engineering with a strong emphasis on basic and interdisciplinary research, innovation, and knowledge sharing for societal benefit.

The National Institute of Advanced Industrial Science and Technology (AIST) was established in 2001 through the merger of several legacy research institutes and is now one of Japan's largest national R&D organizations focused on applied and interdisciplinary research. Headquartered in Tokyo but with its main research base in Tsukuba Science City (Ibaraki Prefecture), AIST employs approximately 2,300 researchers and conducts cutting-edge work in robotics, materials science, energy, and biotechnology. AIST's strong mission is to bridge innovation with industrial application, integrating multidisciplinary expertise to address socio-economic challenges.

The CNRS-AIST Joint Robotics Laboratory (JRL) was founded in 2003 as a strategic partnership between these two institutions. Located in Tsukuba, JRL is internationally renowned for its research in humanoid robotics, specializing in advanced motion control, human–robot interaction, and visual perception via image processing.

JRL's proximity to both AIST and the University of Tsukuba provides an outstanding research ecosystem. Situated about 5 km from the university campus, it benefits from close academic collaboration and high-tech infrastructure. Notably, JRL maintains more humanoid robots than are currently in existence in France, offering one of the most resource-rich environments for experimental robotics research.

Figure 1.1: Building housing the CNRS-AIST Joint Robotics Laboratory (JRL).

1.2 Reinforcement Learning background

In the reinforcement learning framework, the goal is to train an agent to interact effectively with its environment. At each time step, the agent receives the current state s_t and the reward r_t from the previous action a_{t-1}, which reflects how good that action was. Based on this information, it decides which action a_t to take next. The decision-making process is governed by a policy π, which determines the agent's actions (see Figure 1.2). To make good decisions, this policy must be trained using the rewards it receives from the environment. Therefore, the reward design is the interface where a user can specify the task that an agent should perform.

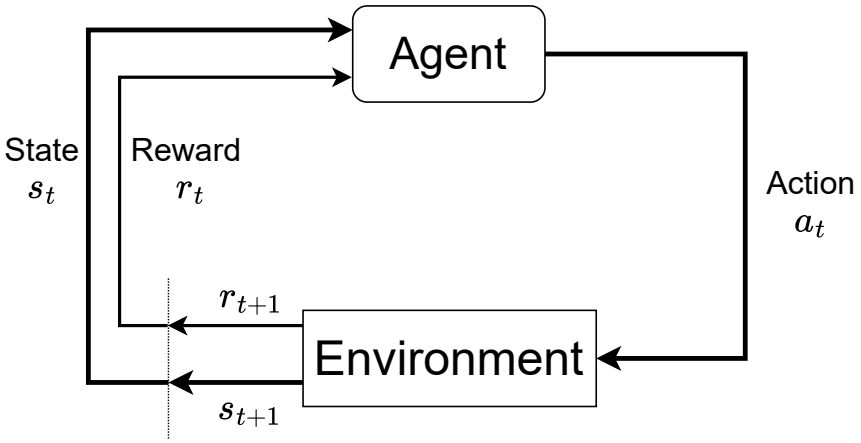

Figure 1.2: Illustration of reinforcement learning model

From a mathematical perspective, the objective in reinforcement learning is to maximize the expected discounted cumulative reward with respect to the policy:

$$\max_{\pi} \mathbb{E}_{p(\tau|\pi)} \left[\mathbf{R}(\tau)\right] \text{ with } \mathbf{R}(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t \tag{1.1}$$

where $\gamma < 1$ is the discount factor, $\tau = \{s_0, a_0, r_0, \ldots, s_{T-1}, a_{T-1}, r_{T-1}\}$ is a trajectory of states, actions and rewards, T is the time horizon and $p(\tau|\pi)$ represents the likelihood of the trajectory τ under π. We call $\mathbf{R}(\tau)$ the discounted cumulative reward.

In many environments including robotics, a distinction is made between **states** and **observations**: the agent does not have direct access to the full state of the environment. Instead, it receives observations, which may be partial or noisy representations of the true state. Below are some other key points to better understand how reinforcement learning was applied during my internship.

Proximal Policy Optimization (PPO) [1]: There are two main approaches to optimize the objective in (1.1): value-based methods and policy-based methods. Value-based methods rely on learning a value function associated with each state (or state-action pair), which estimates how good it is for the agent to be in a given state. The policy is then derived by selecting actions that lead to states with higher estimated value. On the other hand, policy-based methods directly optimize the policy parameters by using the policy-gradient theorem [2].

PPO improves training stability by preventing large, destabilizing updates to the policy. It does this by clipping the policy update in the objective function, ensuring that the new policy does not deviate too far from the old one. This strikes a good balance between exploration and exploitation, and is particularly well-suited for continuous control tasks often found in robotics. The mathematical details are provided in the appendix A.1.

Actor-Critic [3]: Actor-Critic methods combine the strengths of both value-based and policy-based approaches. They consist of two models: the actor, which proposes actions given the current state, and the critic, which evaluates how good those actions are by estimating the value function.

The critic helps reduce the variance of the policy gradient[1] by providing a more informative learning signal, while the actor updates the policy parameters in a direction suggested by the critic. This structure leads to more stable and efficient learning, especially in high-dimensional or continuous action spaces. Actor-Critic architectures are at the core of many modern reinforcement learning algorithms, including PPO.

Rollout storage: In practice, training the policy involves collecting trajectories by running the current policy π in the environment. These trajectories are stored in a rollout storage, also known as a replay buffer or experience buffer. The purpose of this storage is to maintain a record of the agent's past interactions with the environment, which allows it to continue learning without forgetting what it has already experienced. By reusing past trajectories, the agent can reinforce useful behaviors and avoid overfitting to only the most recent experiences.

At regular intervals, a batch of trajectories is randomly sampled from the rollout storage to compute gradients and update the policy. This sampling helps break temporal correlations between transitions and improves training stability. To keep the data fresh and relevant to the current policy, the oldest trajectories in the storage are gradually removed as new ones are added.

1.3 Related work

A natural approach to reproducing a wide range of motions is to train multiple specialized policies, each dedicated to a specific task or skill, and then use a higher-level network to select or blend between them based on the current context or objective. This modular strategy offers flexibility and reusability, allowing each policy to focus on mastering a particular behavior while delegating the decision of when to use which policy to a separate mechanism.

This idea has been explored in various works under different names and frameworks, such as hierarchical reinforcement learning, skill-based learning, and mixture-of-experts models.

[1] The policy-gradient theorem states that the gradient of the objective is given by $\mathbb{E}_{p(\tau|\pi_\theta)}\left[\mathbf{R}(\tau)\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})\right]$. Therefore, the optimization requires computing the term $\frac{\nabla_\theta \pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_\theta(\mathbf{a}|\mathbf{s})}$, which can vary significantly or even diverge when $\pi_\theta(\mathbf{a}|\mathbf{s})$ is very small for some action $\mathbf{a}$.

1.3.1 Adversarial Motion Prior

This work by Xue Bin Peng [4] is foundational for what follows. It adapts the principles of Generative Adversarial Imitation Learning (GAIL), which itself is inspired by Generative Adversarial Networks (GANs), and applies them to reinforcement learning. In this framework, the policy acts as the generator, producing motion trajectories. A discriminator is introduced and trained to distinguish whether a given state transition comes from real human motion data or from the policy. The policy is then trained to fool the discriminator, encouraging it to produce more human-like behaviors (see Figure 1.3).

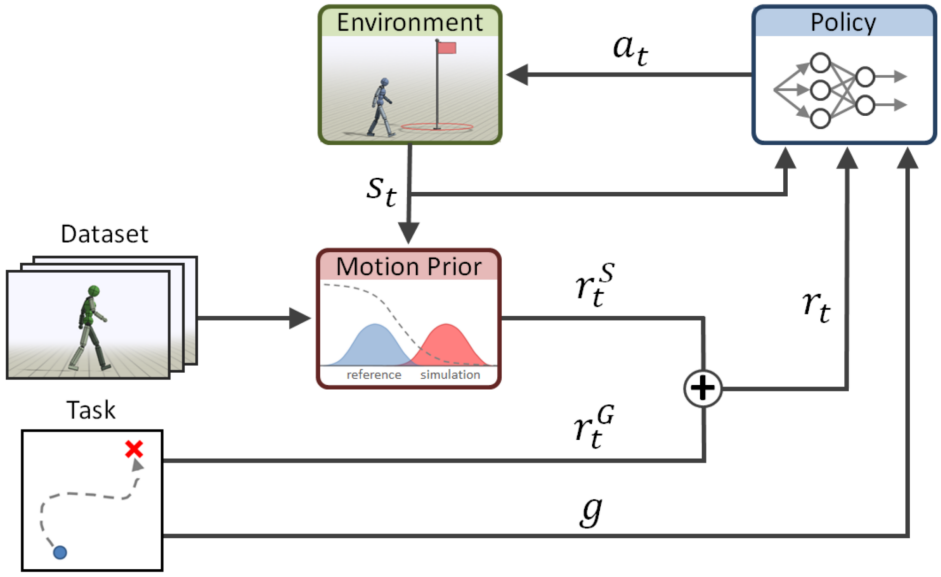

Figure 1.3: Adversarial Motion Prior (AMP) architecture.

In mathematical terms, the goal is to train the policy to fool the discriminator by encouraging it to generate transitions that resemble those from the reference motion dataset. This is done by introducing a specially designed reward, called the style reward and denoted r_t^S, which encourages the policy to follow the style of the demonstrated motions. Additionally, a goal reward r_t^G is included to ensure the policy not only moves in a realistic manner but also accomplishes the task at hand with the desired style.

The style reward is defined as:

$$r_t^S = -\log(1 - D(s_t, s_{t+1})) \tag{1.2}$$

where $D(s_t, s_{t+1})$ is the output of the discriminator, indicating how likely the transition (s_t, s_{t+1}) comes from the motion dataset.

We denote by $d^{\mathcal{M}}(s_t, s_{t+1})$ the distribution of state transitions from the motion dataset, and by $d^{\pi}(s_t, s_{t+1})$ the distribution generated by the policy. When training the discriminator, the following objective is minimized:[2]

$$\min_D -\mathbb{E}_{d^{\mathcal{M}}(s_t,s_{t+1})}\left[\log D(s_t, s_{t+1})\right] - \mathbb{E}_{d^{\pi}(s_t,s_{t+1})}\left[\log(1 - D(s_t, s_{t+1}))\right] \tag{1.3}$$

[2]In practice, we minimize the Least-Squares GAN formulation which avoids the vanishing gradients problem: $\min_D \mathbb{E}_{d^{\mathcal{M}}(s_t,s_{t+1})}\left[(D(s_t, s_{t+1}) - 1)^2\right] + \mathbb{E}_{d^{\pi}(s_t,s_{t+1})}\left[(D(s_t, s_{t+1}) + 1)^2\right]$ and the reward becomes $r_t^S = \max[0, 1 - 0.25(D(s_t, s_{t+1}) - 1)^2]$. However, this formulation is not used in Adversarial Skill Embeddings.

This adversarial setup allows the policy to learn human-like motion by encouraging it to stay close to the reference motion distribution while still achieving task-specific goals. It produces strong results across a variety of tasks, but suffers from a major limitation: the policy must be retrained from scratch whenever the task changes. While this framework excels at imitation, it lacks the ability to generalize to new or unseen tasks.

At the JRL, this framework has been applied to specific tasks that would be too complex or time-consuming to design manually. For instance, as illustrated in Figure 1.4, a researcher worked on enabling a humanoid robot to pick up a box. While the approach produced good results, it required more than just a simple goal reward. In some cases, additional signals had to be introduced to explicitly indicate when the robot should pick up the box. Here, it needed a clock signal to indicate when it should pick up the box. In other words, the policy did not truly generalize but instead relied on carefully crafted auxiliary signals to succeed.

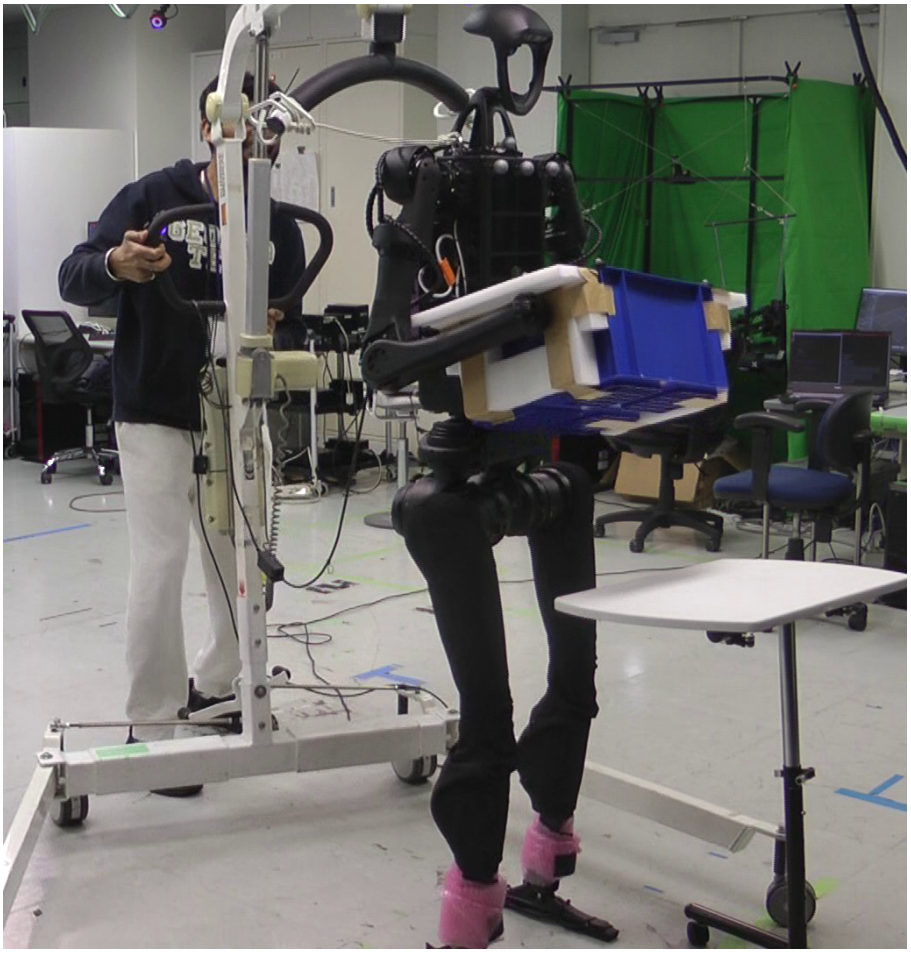

Figure 1.4: Unitree H1 robot [5] picking up a box using the AMP framework.

1.3.2 Mixture-of-Experts

Recent advances have applied the Mixture-of-Experts (MoE) framework to enable a single policy to handle multiple diverse tasks, especially in locomotion and humanoid robotics. Mixture of Residual Experts (MoRE) [6] is a good example of a complete architecture that allows generalization by including both proprioceptive and exteroceptive informa-

tion. However, like other architectures aiming for generalization with human-like gaits, it remains relatively complex and tedious to train.

The first stage of MoRE consists of training a standard reinforcement learning policy using both proprioceptive and exteroceptive inputs. We denote by $\mathbf{o}_t$ the proprioceptive observation vector at time t, which can include the robot's angular velocity, projected gravity vector, commanded velocities, joint angles, joint velocities, and previous actions. We denote by $\mathbf{I}_t$ the depth image at time t. The images are encoded through a 2D Convolutional Neural Network to produce a visual feature vector $\mathbf{f}_t^d$.

In addition, the policy receives a history of proprioceptive observations $\mathbf{o}_{t-H:t}$, which is processed by a 1D convolutional encoder to produce a proprioceptive history feature vector $\mathbf{f}_t^h$. To reduce the overall input dimensionality, the concatenated features $(\mathbf{f}_t^d, \mathbf{o}_t, \mathbf{f}_t^h)$ are projected into a latent representation $\mathbf{z}_t^o$ by the actor's hidden layers.

Training is performed using PPO with an asymmetric actor–critic setup, meaning that the critic has access to privileged information (e.g., elevation/height maps, ground-truth feet positions) that is not available to the actor. After this stage, the policy can robustly traverse stairs, gaps, slopes, and other challenging terrains using depth vision alone.

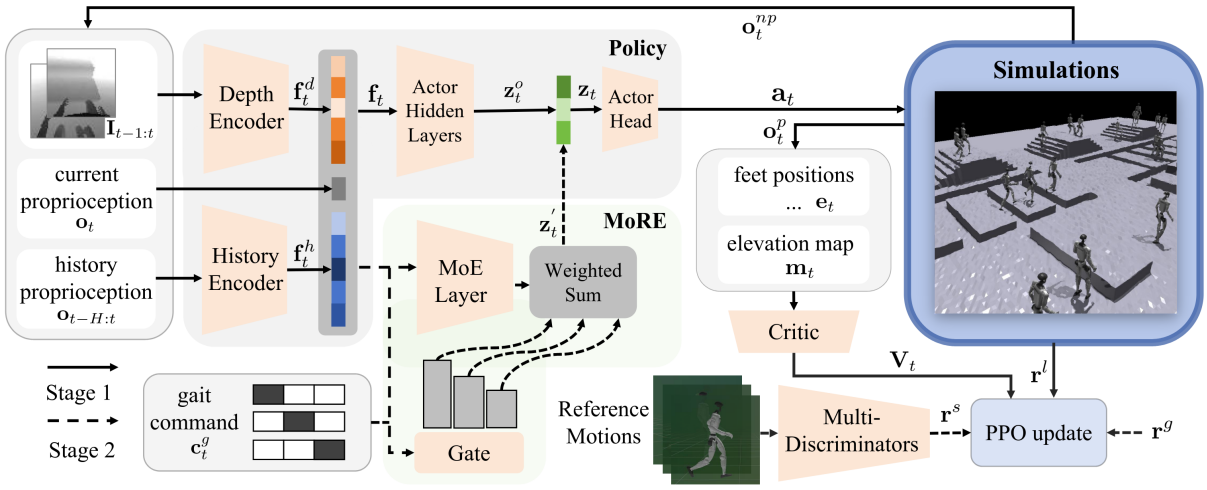

Figure 1.5: Mixture of Residual Experts architecture. The locomotion reward r_l (composed of 20 terms) is used in both Stage 1 and Stage 2 training. The style reward r_s and gait-specific reward r_g are applied only during Stage 2 to encourage human-like motion and enforce gait-specific behaviors.

Once this base policy is trained, the second stage focuses on adding human-likeness to the motion (see Figure 1.5). This is achieved by training a residual policy, meaning we add correction terms to the latent features of the base policy. These residuals are generated by a MoE module, where each expert is a small MLP that outputs a latent residual vector $\Delta\mathbf{z}_t^{(k)}$ for a specific motion style or gait. A gating network (another small MLP) takes as input the current observation features and an externally provided one-hot gait command vector c_t^g indicating the desired gait (e.g., walking, high-knees, squatting). The gating network outputs softmax weights $w_t^{(k)}$ over the experts. The final residual is then:

$$\Delta\mathbf{z}_t = \sum_{k=1}^{K} w_t^{(k)} \Delta\mathbf{z}_t^{(k)} \tag{1.4}$$

This residual is added to the base latent $\mathbf{z}_t^o$, and the sum $\mathbf{z}_t^o + \Delta\mathbf{z}_t$ is passed through

the final action head to produce the action $\mathbf{a}_t$. In parallel, for each gait g, a separate discriminator D_g is trained to distinguish between real human motion capture data for that gait and the policy's generated motion when c_t^g requests that gait and this is done using an AMP style loss. This style reward is combined with the original locomotion rewards and possible gait-specific shaping rewards:

$$R_t = r_l + \mathbf{1}_{[c_t^g = g]}(r_s^{(g)} + r_g^{(g)}) \tag{1.5}$$

and the whole pipeline (the actor-critic and the MoE residual module) is updated via PPO to maximize this combined reward.

While MoRE demonstrates strong performance in producing terrain-robust and human-like locomotion, it also comes with several limitations. First, the large number of reward terms (20 for locomotion alone) makes reward design tedious and environment-specific, limiting true generalization. Second, the framework scales poorly with the number of gaits: adding a new gait typically requires introducing an additional expert in the MoE module, designing new gait-specific rewards, and collecting labeled motion capture data for the style discriminator. As a result, although MoRE achieves high-quality results, it remains a complex and labor-intensive approach to extend and maintain.

1.4 Adversarial Skill Embeddings

We have seen through MoRE that it is possible to make general policies that can have multiple skills all-in-one, even including exteroceptive information. But the main drawback remains: MoRE requires carefully crafted rewards and a complex architecture. Instead, we would rather employ a method that does not require manually adding each gait, but can instead discover and organize skills automatically. This is the idea behind Adversarial Skill Embeddings (ASE) [7], which builds upon the AMP framework. The key innovation is to combine the adversarial imitation learning objective with the same idea as variational auto-encoders (VAEs), thereby enforcing both motion imitation and the learning of a compact, continuous skill embedding space.

1.4.1 Low-level policy

In ASE, a large and diverse set of motion clips is provided as training data, without labeling them into discrete gaits or categories. Each motion is associated with a latent skill vector $\mathbf{z}$ in a low-dimensional space, and the reference clips provide the supervision for the network to learn how to construct and organize this latent space. This skill vector is then provided to the policy alongside the usual state observations, conditioning the policy's behavior on the skill. The policy is trained, as in AMP, to fool a discriminator into classifying its motion as real, and to minimize an encoder loss that plays a role similar to the KL-divergence term in a VAE, organizing the latent space and ensuring that it follows the desired distribution.

A common issue in neural networks is the vanishing or exploding gradient: when the input norm deviates from 1, training can produce excessively small or large gradients, leading to instability. To mitigate this, the ASE framework constrains the latent skill vectors to lie on an n_{latent}-dimensional hypersphere, ensuring a constant norm of 1. As a result, a specific skill can be selected by choosing the corresponding skill vector on the hypersphere (see Figure 1.6).

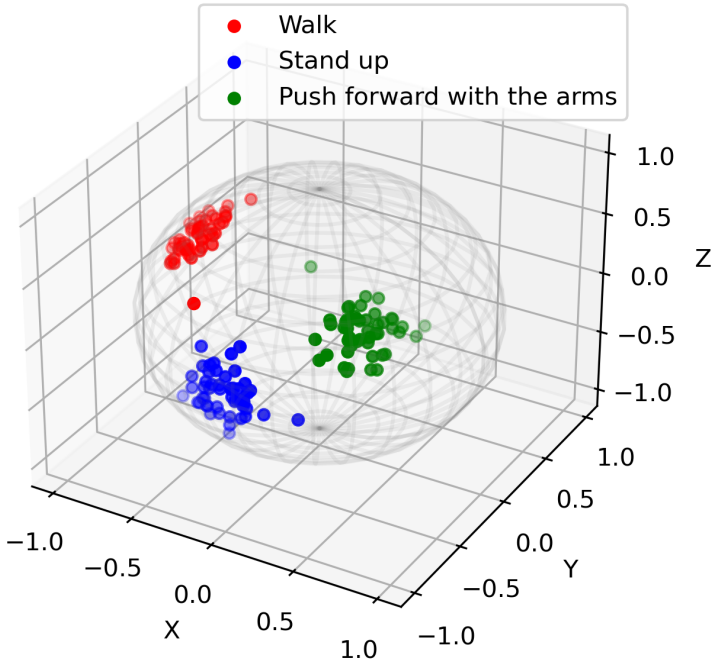

Figure 1.6: An illustration of the latent skill space in three dimensions. After training, the hyperspherical latent space exhibits clusters corresponding to different learned skills. By selecting points from these clusters, individual skills can be executed, and by interpolating between them, more complex or blended behaviors can be generated.

The first stage of training focuses on learning to cluster and imitate skills. The goal is to obtain a low-level policy capable of reproducing specific skills when conditioned on a latent vector, which acts as a command. This stage involves two main objectives: on the one hand, minimizing the discrepancy between the motions generated by the policy and those in the dataset; on the other hand, encouraging nearby latent vectors to produce similar motions. The architecture of the low-level policy is illustrated in Figure 1.7.

We denote a latent skill vector as $\mathbf{z}$. Since the latent space is constrained to an n_{latent}-dimensional hypersphere, the distribution over latent vectors corresponds to the spherical analogue of a Gaussian, the von Mises–Fisher distribution:

$$f_p(\mathbf{x}; \mu, \kappa) = C_p(\kappa) \exp(\kappa \mu^\top \mathbf{x}) \tag{1.6}$$

where $C_p(\kappa)$ is a normalization constant, μ is the mean vector and κ is the concentration parameter controlling the spread. To encourage imitation of the motion dataset, the imitation reward of equation (1.2) is used.

Additionally, an encoding reward is introduced to regularize the latent space. Given a state transition, we estimate the mean of the corresponding latent distribution (i.e. where this transition would lie on the hypersphere if properly represented). Using this estimated mean, we can evaluate whether the latent vector $\mathbf{z}$ is appropriate for the given transition, since the probability density can be computed using equation (1.6). Denoting μ_q as the encoder's predicted mean for the transition (s_t, s_{t+1}), the encoding reward is defined as the log-probability under the von Mises-Fisher distribution, which is proportional to the inner product:

$$r_t^E = \kappa \mu_q(s_t, s_{t+1})^\top \mathbf{z} \tag{1.7}$$

Therefore, this reward encourages the policy to align its behavior with the encoder's representation.

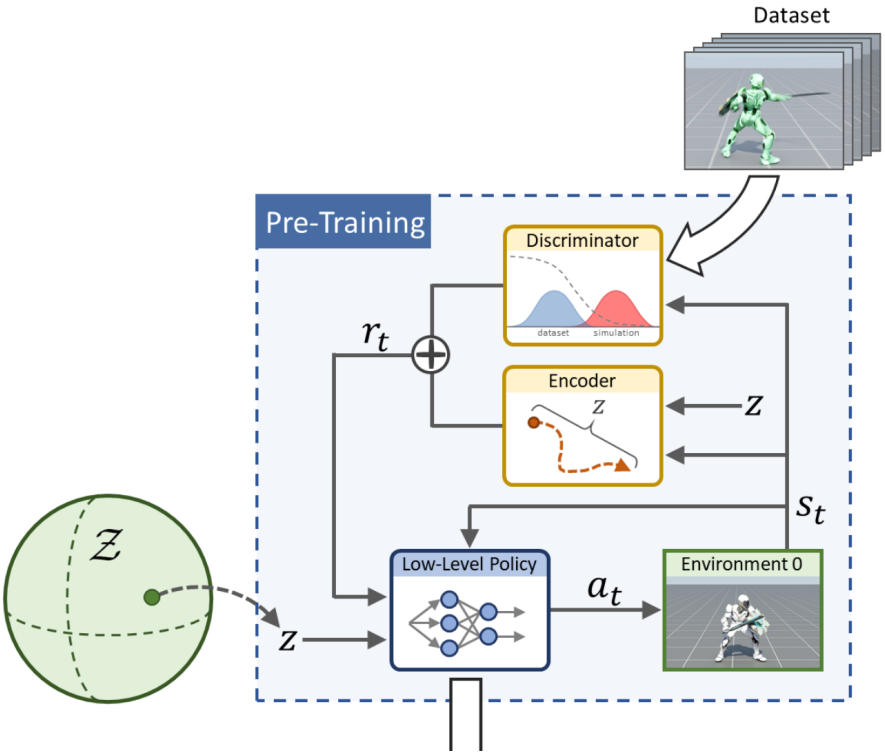

Figure 1.7: Overview of the pre-training phase for learning the low-level policy. A latent skill vector z is sampled from the latent space $\mathcal{Z}$ modeled as an hypersphere and passed to the low-level policy. The resulting state s_t is encoded into a latent representation using the encoder, and a discriminator distinguishes between trajectories from the motion dataset and the simulation.

An additional component of the objective is the diversity term. Up to this point, the problem can be formulated as the standard RL objective, where a sequence of latent vectors $\mathbf{Z} = \{\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{T-1}\}$ corresponds to a trajectory τ:

$$\max_{\pi} \mathbb{E}_{p(\mathbf{Z})} \mathbb{E}_{p(\tau|\pi,\mathbf{Z})} \left[\sum_{t=0}^{T-1} \gamma^t \left(r_t^S + \beta r_t^E \right) \right] \tag{1.8}$$

where β is a tunable hyperparameter, r_t^E is given by equation (1.7), and r_t^S by equation (1.2).

A recurrent issue with this objective is the slow responsiveness of the low-level policy: changes in the latent vector $\mathbf{z}$ often take several seconds to manifest in the agent's behavior. To address this, the authors introduce a diversity term to encourage faster and more distinct behavioral changes:

$$\begin{aligned} \max_{\pi} \; & \mathbb{E}_{p(\mathbf{Z})} \mathbb{E}_{p(\tau|\pi,\mathbf{Z})} \left[\sum_{t=0}^{T-1} \gamma^t \left(r_t^S + \beta r_t^E \right) \right] \\ & - w_{\text{div}}, \mathbb{E}_{d^\pi(\mathbf{s})}, \mathbb{E}_{\mathbf{z}_1,\mathbf{z}_2 \sim p(\mathbf{z})} \left[\left(\frac{D_{KL}(\pi(\cdot|\mathbf{s},\mathbf{z}_1), \pi(\cdot|\mathbf{s},\mathbf{z}_2))}{D_{\text{z}}(\mathbf{z}_1,\mathbf{z}_2)} - 1 \right)^2 \right] \end{aligned} \tag{1.9}$$

where $D_{\text{z}}(\mathbf{z}_1, \mathbf{z}_2) = 0.5(1 - \mathbf{z}_1^\top \mathbf{z}_2)$ is the cosine distance between latent vectors, and $d^\pi(\mathbf{s})$ is the distribution of states visited under policy π.

Intuitively, this term enforces that if two latent vectors $\mathbf{z}_1$ and $\mathbf{z}_2$ are far apart in the latent space, the corresponding action distributions $\pi(\cdot|\mathbf{s}, \mathbf{z}_1)$ and $\pi(\cdot|\mathbf{s}, \mathbf{z}_2)$ should differ equally. This encourages the policy to react more quickly and distinctly when the latent changes, improving responsiveness.

The discriminator in the low-level policy is trained according to Equation (1.3), augmented with a gradient penalty term:

$$\begin{aligned}\min_D & -\mathbb{E}_{d^{\mathcal{M}}(s_t,s_{t+1})}\left[\log D(s_t, s_{t+1})\right] - \mathbb{E}_{d^{\pi}(s_t,s_{t+1})}\left[\log(1 - D(s_t, s_{t+1}))\right] \\ & + w_{\text{gp}}\mathbb{E}_{d^{\mathcal{M}}(s_t,s_{t+1})}\left[\|\nabla_\phi D(\phi)|_{\phi=(s_t,s_{t+1})} - 1\|^2\right] \end{aligned} \tag{1.10}$$

where w_{gp} is a manually specified coefficient. The last term regularizes the discriminator via a gradient penalty, encouraging smoother decision boundaries.

The encoder is optimized to maximize the compatibility between the sampled latent vector $\mathbf{z}$ and the encoder's output $\mu_q(s_t, s_{t+1})$ for the observed state transition. This can be expressed as:

$$\max_q \mathbb{E}_{p(\mathbf{z})}\mathbb{E}_{d^{\pi}(s_t,s_{t+1}|\mathbf{z})}\left[\kappa\mu_q(s_t, s_{t+1})^\top \mathbf{z}\right] \tag{1.11}$$

As discussed later in Section 1.4.3, this formulation is related to maximizing the log-likelihood of latent samples generated by the policy π, thereby encouraging the encoder to produce latent representations that are well aligned with the policy's behavior.

1.4.2 High-level policy

Once we have a pre-trained low-level policy that meets all the desired requirements, we can compose complex motions by selecting and sequencing skills. The high-level policy outputs a latent vector $\mathbf{z}_t$, which is passed to the low-level policy. The low-level policy then produces the corresponding action a_t at time step t (see Figure 1.8).

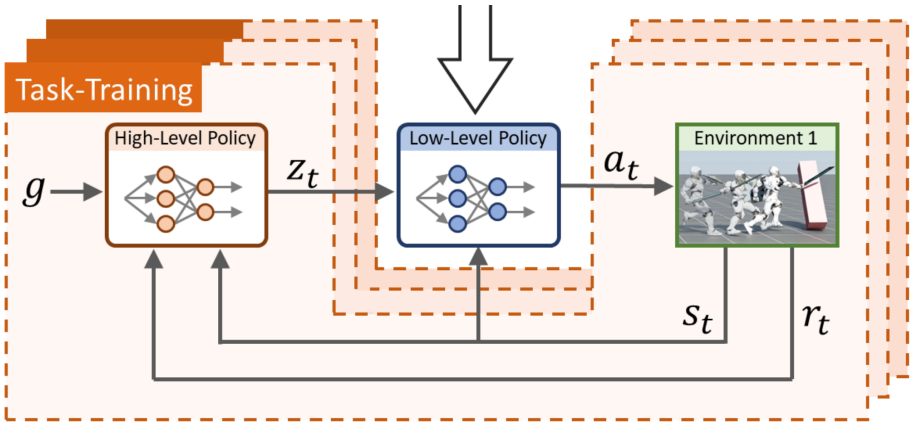

Figure 1.8: Overview of the task-training phase. The high-level policy receives the task goal g, which is defined through the design of the reward r_t, and outputs a latent skill vector z_t. This latent vector is then passed to the pre-trained low-level policy, which generates the action a_t for the environment.

In practice, the high-level policy $\omega(\mathbf{z}|\mathbf{s}, g)$ samples a latent vector $\bar{\mathbf{z}} \sim \mathcal{N}(\mu_\omega(\mathbf{s}, g), \Sigma_\omega)$, where $\mu_\omega(\mathbf{s}, g)$ is produced by a neural network and Σ_ω is a constant covariance matrix. The latent is then normalized as $\mathbf{z} = \bar{\mathbf{z}}/\|\bar{\mathbf{z}}\|$, ensuring that it lies on the hypersphere and remains specific in its selection (see Figure 1.9).

1.4.3 Mathematical Formulation of the Low-Level Policy

Let us dive into the mathematical details to better understand the foundations of the low-level policy and its underlying mechanisms. The two objectives mentioned earlier (learning to cluster and imitate skill) are equivalent to the following optimization problem:

$$\max_{\pi} -D_{\text{JS}}\left(d^{\pi}(s_t, s_{t+1}) \| d^{\mathcal{M}}(s_t, s_{t+1})\right) + \beta I\left(s_t, s_{t+1}; \mathbf{z} | \pi\right) \tag{1.12}$$

The first term minimizes the Jensen-Shannon divergence between the state transition distribution induced by the policy and that of the dataset—this corresponds to the imitation objective. The second term maximizes the mutual information between the state transitions and the latent variable $\mathbf{z}$, conditioned on the policy π.

$$I\left(s_t, s_{t+1}; \mathbf{z} | \pi\right) = \mathcal{H}\left(s_t, s_{t+1} | \pi\right) - \mathcal{H}\left(s_t, s_{t+1} | \mathbf{z}, \pi\right) \tag{1.13}$$

where $\mathcal{H}$ denotes entropy. Thus, maximizing the mutual information encourages two behaviors simultaneously: increasing the overall entropy of state transitions (promoting diverse behaviors) and reducing the entropy of transitions conditioned on $\mathbf{z}$ (making the transitions predictable given the latent command). Therefore, the second term essentially ensures that nearby latent vectors produce similar motions.

To maximize the mutual information, we use the same principle as the Evidence Lower Bound (ELBO). First, we rewrite equation (1.13). Since mutual information is symmetric, we have:

$$\begin{aligned} I\left(s_t, s_{t+1}; \mathbf{z} | \pi\right) &= I\left(\mathbf{z}; s_t, s_{t+1} | \pi\right) \\ &= \mathcal{H}(\mathbf{z}|\pi) - \mathcal{H}(\mathbf{z}|s_t, s_{t+1}, \pi) \\ &= \mathcal{H}(\mathbf{z}) - \mathcal{H}(\mathbf{z}|s_t, s_{t+1}, \pi) \quad \text{(since } \mathbf{z} \text{ is independent of } \pi\text{)} \end{aligned}$$

The first term, $\mathcal{H}(\mathbf{z})$, is constant with respect to π, so we focus on finding a lower bound for the second term. Using Gibbs' inequality, for any two distributions p and q, we have:

$$-\mathbb{E}_p[\log p] \leq -\mathbb{E}_p[\log q]$$

with equality if and only if $p = q$. Recognizing the entropy on the left-hand side, we apply this to $-\mathcal{H}(\mathbf{z}|s_t, s_{t+1}, \pi)$, yielding:

$$I\left(s_t, s_{t+1}; \mathbf{z} | \pi\right) \geq \mathcal{H}(\mathbf{z}) + \max_{q} \mathbb{E}_{p(\mathbf{z})} \mathbb{E}_{p(s_t, s_{t+1} | \pi, \mathbf{z})}\left[\log q(\mathbf{z}|s_t, s_{t+1})\right] \tag{1.14}$$

We recover the appearance of the encoder q, whose role is now clearer: it estimates a distribution over $\mathbf{z}$ given the state transition (s_t, s_{t+1}), in order to maximize the mutual information as described in equation (1.13).

Therefore, the Jensen–Shannon divergence term is implemented in practice through an imitation reward, which serves the same purpose of aligning the policy's motion distribution with that of the dataset. Likewise, the term inside the expectation in (1.14) corresponds to the encoder objective, which encourages the policy to produce motions that are consistent with the latent vector $\mathbf{z}$, as inferred by the encoder.

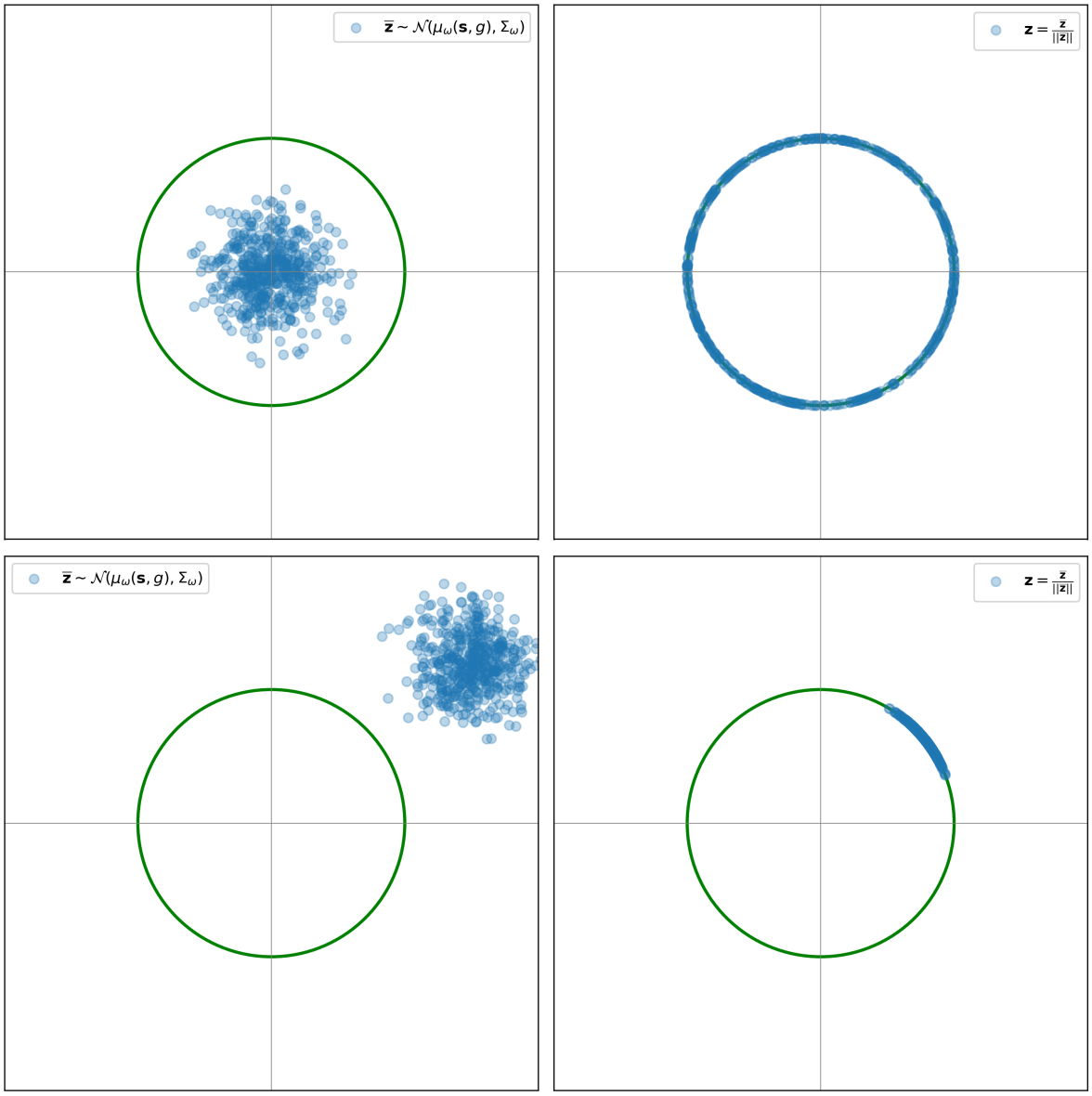

Figure 1.9: Examples of latent vector sampling and normalization in the high-level policy framework. The left column shows latent vectors sampled from a Gaussian distribution, while the right column illustrates the normalized vectors lying on a hypersphere. Changing $\mu_\omega(\mathbf{s}, g)$ can lead to either selecting a specific skill or performing random sampling over the entire latent space.

Chapter 2

Adversarial Skill Embeddings applications

Following discussions with my supervisor, we decided to explore the ASE framework to enable a robot to both walk and open a door, with smooth transitions between the two skills. My plan was to first train a low-level policy for walking. For this purpose, I used the Unitree H1 robot, illustrated in Figure 1.4.

The original ASE paper reports a training setup requiring 10 days with 4096 simulated environments running in parallel on a single NVIDIA V100 GPU, using a 30-minute motion dataset of everyday locomotion behaviors. While we had access to more powerful GPUs at the JRL (NVIDIA GeForce RTX 4090), the training time and dataset requirements were impractical. The motion data used by the authors came from NVIDIA's paid datasets, where each clip (10–20 seconds) costs around $20. This made it clear that our first step was to scale down the problem by reducing dataset size, training time, and computational resources, while still making the framework functional.

The ASE implementation available on GitHub was originally built for Isaac Gym. However, in December 2024, a new simulator, Genesis [8], was released, claiming to be at least 10 times faster than Isaac Gym. While indeed very fast, Genesis had incomplete documentation and required digging into the source code to implement specific functionalities. Nevertheless, I chose to reimplement ASE on Genesis, starting from the rsl_rl framework developed by ETH Zürich and NVIDIA [9].

In the first part of this chapter, I describe how I reimplemented ASE from scratch, including the preprocessing of the motion dataset and the addition of a sanity check to deepen the understanding of the latent space. In the second part, I detail the efforts made to analyze and adapt ASE, first on more complex tasks and later on real-world robotics, where addressing the challenges of the sim-to-real gap was a key focus.

2.1 Progressive Development: From AMP to ASE

Initially, I implemented the AMP framework in isolation, which consists solely of a discriminator providing an imitation reward. This allowed me to validate the core reinforcement learning and reward integration pipeline in a simplified setting.

Subsequently, I integrated the encoder module and conducted a series of controlled experiments designed to verify the correctness of the implementation. These experiments served as sanity checks, ensuring that the system produced the expected behavior before moving to more complex configurations.

2.1.1 Preprocessing Motion Data for Imitation Learning

Initially, I worked with the LAFAN1 Unitree dataset [10], a motion capture dataset already adapted and retargeted for the Unitree H1 humanoid. Typically, motion capture sequences involve a human performer. To make these motions executable on a robot, a mapping must be established between the human's degrees of freedom (DoF) and those of the robot. In our case, the human model has 45 DoF, whereas the H1 robot has only 19 DoF, making a direct one-to-one mapping insufficient. This necessitates a motion retargeting process.

Using the raw LAFAN1 Unitree data directly can produce inconsistent behavior. Instead, we extract specific clips containing a single, well-defined motion such as walking. Each clip contains the position and orientation of the robot's base in the world frame, along with the angles of all its DoFs. From this information, we give to the discriminator:

- The linear and angular velocities of the base, expressed in the robot's local frame.
- The angular positions and velocities of all DoFs

In the initial implementation, the reward is defined as $r_t = r_t^S$ (Equation 1.2), meaning only the discriminator's signal is used. The discriminator itself is updated according to Algorithm 1.

Algorithm 1 AMP Training

input $\mathcal{M}$: dataset of reference motions
$D \leftarrow$ initialize discriminator
$\pi \leftarrow$ initialize policy
$V \leftarrow$ initialize value function
while not done **do**
    $\mathcal{B} \leftarrow \emptyset$ ▷ initialize data buffer
    for trajectory $i = 1, \ldots, m$ **do**
        $\tau^i \leftarrow \{s_0, a_0, s_1, \ldots, s_T\}$ collect trajectory with π
        for time step $t = 0, \ldots, T-1$ **do**
            $r_t \leftarrow -\log\left(1 - D(s_t, s_{t+1})\right)$
            record r_t in τ^i
        end for
        store τ^i in $\mathcal{B}$
    end for

    Update discriminator:
    for update step $= 1, \ldots, n$ **do**
        $b^{\mathcal{M}} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^{K}$ from $\mathcal{M}$
        $b^{\pi} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^{K}$ from $\mathcal{B}$
        update D according to Equation 1.10 using $b^{\mathcal{M}}$ and b^{π}
    end for

    update V and π using data from $\mathcal{B}$
end while

The training procedure yields a human-like gait (see video AMP_walk_lafan.mp4) after approximately 30 minutes. The process can be divided into two distinct phases. In the

first phase, the agent rapidly learns to stand upright, which is achieved within a few minutes of training. The second phase, corresponding to the acquisition of locomotion, is considerably longer. This is due to the fact that maintaining a standing posture is more immediately rewarding than attempting a step and risking a fall. Notably, at the beginning of the video, there is a short interval of about two seconds where the robot consistently struggles to initiate its first step, which it always performs with the right foot.

To improve generalization, an additional data processing step is introduced. For each motion sequence in the dataset, we generate a mirrored counterpart by reflecting the movement across the sagittal plane. This effectively doubles the dataset while ensuring that both left and right-sided variations of the motion are represented.

2.1.2 Retargeting

If the objective is to train the robot to imitate a human motion, it becomes necessary to record a custom dataset, which in turn requires learning how to perform motion retargeting. To this end, I employed Pyroki [11], a modular toolkit for robot kinematic optimization, to implement an optimization-based retargeting procedure. The approach consists of optimizing over the robot's joint angles[1] and base position, with an objective function designed to minimize the discrepancy between the robot's motion and the reference dataset motion.

At a given time step t, let c_i denote the angle of the robot's i-th joint, and let w_i represent the global position (x, y, z) of this joint in the world frame. We further define $\delta_{ij} = w_i - w_j$ as the relative position of joint i with respect to joint j.

The most straightforward objective is to ensure that the robot's joint positions in the global frame are as close as possible to those in the reference data. This is captured by the global alignment residual. In addition, the robot's joint angles must remain within their feasible ranges; thus, a joint limit residual is introduced to penalize any violations of joint boundaries.

Beyond global alignment, we also require the relative positions of each joint with respect to its child joints to match those observed in the dataset. For instance, this ensures that the robot's legs remain aligned with those of the actor, up to a scaling factor. An illustration of this idea is provided in Figure 2.1. Since these scale factors may differ between joints and are not known a priori, we introduce variables s_{ij} representing the scaling between each pair of corresponding joints in the human and robot models. These scale factors are optimized jointly with the retargeting process.

In addition to optimizing the relative positions of consecutive joints, we also enforce consistency at the angular level. Specifically, the joint angles of the robot should match those from the dataset, which means that the angle difference between the two should be zero. Denoting this angular difference by θ_{ij}, we can express the condition as

$$\theta_{ij} = 0 \iff 1 - \cos(\theta_{ij}) = 0$$

where

$$\cos(\theta_{ij}) = \frac{\delta_{ij} \cdot \delta_{ij}^{\text{data}}}{\|\delta_{ij}\| \|\delta_{ij}^{\text{data}}\|}$$

Finally, two additional residuals are introduced to improve realism and ensure feasible behaviors. The rest residual penalizes joint configurations that deviate too far from a

[1] A joint is equivalent to a degree of freedom.

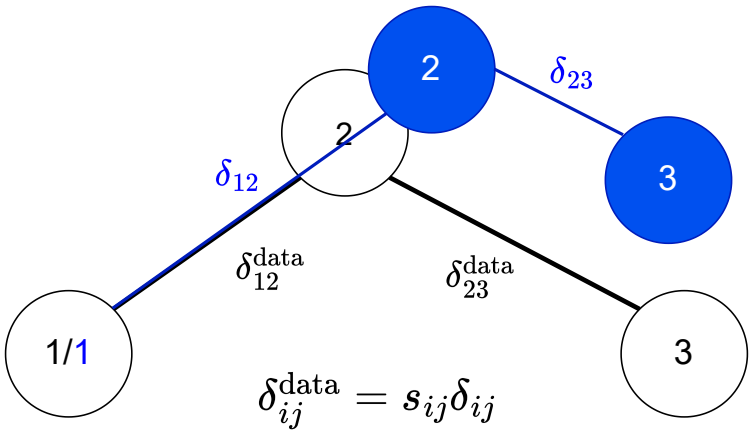

Figure 2.1: Alignment between the motion capture skeleton (black) and the robot model (blue). Due to structural differences, an offset exists between corresponding nodes (e.g., node 2 of the skeleton and the matching robot joint). To achieve optimal retargeting, scaling factors must be computed for each pair of joints to minimize these discrepancies.

predefined rest pose, thereby preventing unnatural positions. The smoothness residual penalizes abrupt variations in motion, ensuring temporal consistency and coherence.

All residual terms are summarized in Table 2.1. These terms are inspired by the cost functions described in the Pyroki paper [11].

Residual Name	**Formula**
Global Alignment	$\sum_i w_i - w_i^{\text{data}}$
Local Alignment Position	$\sum_{(i,j)} \left(\delta_{ij}^{\text{data}} - s_{ij}\delta_{ij}\right)$
Local Alignment Angle	$\sum_{(i,j)} 1 - \cos(\theta_{ij})$
Joint Limit	$\sum_i \max(0, c_i - c_{i,\max}) + \max(0, c_{i,\min} - c_i)$
Rest	$\sum_i c_i - c_{i,\text{rest}}$
Smoothness	$\sum_i c_{i,t} - c_{i,t-1}$, where $c_{i,t}$ and $c_{i,t-1}$ are the i-th joint angle at time t and $t-1$

Table 2.1: Residual terms and their mathematical formulations.

By denoting c the set of all joint angles and s the set of scale factors, the retargeting objective at a given time step t can be expressed as the minimization of the squared sum of all residuals defined in Table 2.1:

$$\min_{c,,s} \sum_{r \in \mathcal{R}} r(c,s)^2 \tag{2.1}$$

where $\mathcal{R}$ denotes the set of residual terms (global alignment, local alignment, joint limits, rest pose and smoothness). It finally gives us the result shown in Figure 2.2 and in the video retargeting.mp4.

Through the processes detailed above (motion retargeting and dataset symmetrization), we achieve a high-fidelity imitation of human motion. This result not only validates

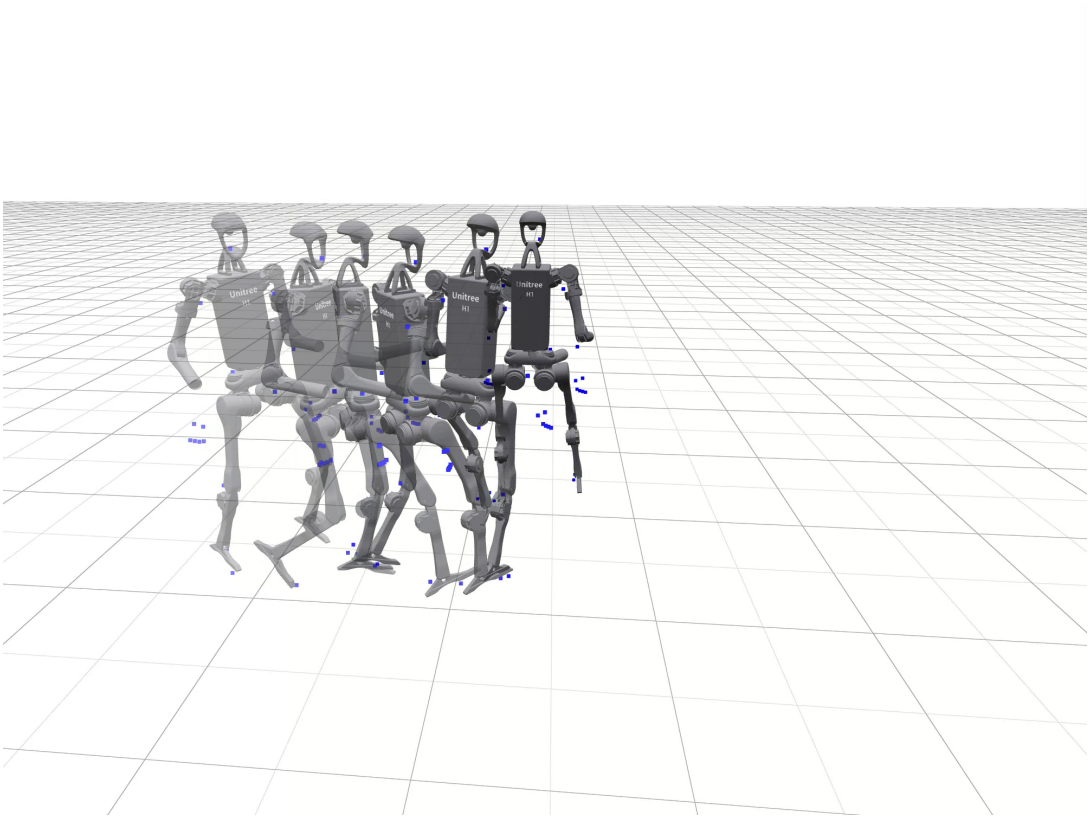

Figure 2.2: Chronophotographic sequence of the Unitree H1 robot demonstrating motion retargeting: while the robot accurately replicates human-like locomotion (with knee flexion) based on the motion capture dataset (blue dots), the elbows remain fixed in a pre-set position to avoid self-collision

our approach but also paves the way for reproducing ASE.

2.1.3 Designing and validating the latent space

Now that imitation is working, we can shift our focus to implementing the encoder and designing the latent space. In the original paper, the latent dimension is set to 64, a relatively large number, which makes it difficult to directly assess the structure and behavior of the latent space. To gain intuition, I first reduced the latent dimensionality and built a simple sanity check before integrating the full pipeline.

The goal is to verify two key properties of the latent space: (i) clustering of skills (i.e., whether a given latent variable $\mathbf{z}$ consistently reproduces a specific motion), and (ii) generalization of skills. For this, I selected a short motion clip where the actor first walks forward and then backward. Ideally, the latent space should separate into two clusters corresponding to forward and backward walking, while also generalizing the motion by adapting the walking speed.

Since the simplest hypersphere that can accommodate two clusters is a circle, I defined the latent space as a circle for visualization purposes. In theory, one point on the circle should correspond to forward walking (as in the dataset), while the opposite point should correspond to backward walking. The intermediate points along the circle should interpolate smoothly, producing slower forward walking, a stationary stance, and then slower backward walking.

The low-level policy is trained using Algorithm 2.

Algorithm 2 ASE Pre-Training

input $\mathcal{M}$: dataset of reference motions
$D \leftarrow$ initialize discriminator
$q \leftarrow$ initialize encoder
$\pi \leftarrow$ initialize policy
$V \leftarrow$ initialize value function
while not done **do**
    $\mathcal{B} \leftarrow \emptyset$ ▷ initialize data buffer
    for trajectory $i = 1, \ldots, m$ **do**
        $\mathbf{Z} \leftarrow$ sample sequence of latents $\{\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{T-1}\}$ from $p(\mathbf{z})$
        $\tau^i \leftarrow \{s_0, a_0, s_1, \ldots, s_T\}$ collect trajectory with π and $\mathbf{Z}$
        record $\mathbf{Z}$ in τ^i
        for time step $t = 0, \ldots, T-1$ **do**
            $r_t \leftarrow -\log\left(1 - D(s_t, s_{t+1})\right) + \beta \log q(\mathbf{z}_t | s_t, s_{t+1})$
            record r_t in τ^i
        end for
        store τ^i in $\mathcal{B}$
    end for

    Update encoder:
    for update step $= 1, \ldots, n$ **do**
        $b^\pi \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from $\mathcal{B}$
        update q according to Equation 1.11 using b^π
    end for

    Update discriminator:
    for update step $= 1, \ldots, n$ **do**
        $b^\mathcal{M} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from $\mathcal{M}$
        $b^\pi \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from $\mathcal{B}$
        update D according to Equation 1.10 using $b^\mathcal{M}$ and b^π
    end for

    update V and π according to Equation 1.8 using data from $\mathcal{B}$
end while

Removing the diversity loss: In practice, I found that the diversity term in the objective (equation 1.9) creates serious convergence issues. To compute this term, two latent variables $\mathbf{z}_1$ and $\mathbf{z}_2$ are sampled, the KL-divergence between the corresponding policies is calculated, and the result is divided by their cosine distance.

The problem arises when $\mathbf{z}_1$ and $\mathbf{z}_2$ are very close. In that case, the cosine distance approaches zero, causing the diversity loss to blow up. This matches my observations: the loss typically stagnates around zero, then randomly spikes to magnitudes as large as 10^6, before collapsing back down again.

So why did the authors include this term? Their setting involved high-dimensional latent spaces, where the probability of sampling two latent variables that are close to each other is extremely low. In fact, this probability approaches zero as the dimensionality of

the space tends to infinity.

Concretely, during training, latent samples are drawn as $\overline{\mathbf{z}} \sim \mathcal{N}(0, I_d)$, where d is the latent dimension. The latent variable is then normalized as $\mathbf{z} = \overline{\mathbf{z}}/\|\overline{\mathbf{z}}\|$. We will now show that as d increases, the expected dot product between two such latent variables converges to zero.

A random vector X is said to be *isotropic* if its covariance matrix satisfies

$$\Sigma(X) = \mathbb{E}[XX^\top] = \lambda I_d$$

In our case, the latent variable $\mathbf{z}$ is centered and isotropic, since the distribution $p(\mathbf{z})$ is orthogonally invariant. Indeed, for any orthogonal matrix Q,

$$Q\mathbf{z} = \frac{Q\overline{\mathbf{z}}}{\|Q\overline{\mathbf{z}}\|} \overset{\text{distr}}{=} \frac{\overline{\mathbf{z}}}{\|\overline{\mathbf{z}}\|} = \mathbf{z}$$

which implies that $\mathbb{E}[\mathbf{z}\mathbf{z}^\top] = \lambda I_d$.

Taking the trace gives

$$\operatorname{tr}(\mathbb{E}[\mathbf{z}\mathbf{z}^\top]) = \lambda d = \mathbb{E}[\|\mathbf{z}\|^2] = 1$$

so that $\lambda = \frac{1}{d}$.

Now consider two independent samples $\mathbf{z}_1, \mathbf{z}_2 \sim p(\mathbf{z})$. We are interested in

$$\mathbb{E}[\langle \mathbf{z}_1, \mathbf{z}_2 \rangle^2] = \mathbb{E}_{\mathbf{z}_2}\Big[\mathbb{E}_{\mathbf{z}_1}\Big[\langle \mathbf{z}_1, \mathbf{z}_2 \rangle^2 \mid \mathbf{z}_2\Big]\Big]$$

Fixing $\mathbf{z}_2$ and computing the inner expectation:

$$\begin{aligned}
\mathbb{E}_{\mathbf{z}_1}[\langle \mathbf{z}_1, \mathbf{z}_2 \rangle^2] &= \mathbb{E}_{\mathbf{z}_1}\left[\mathbf{z}_2^\top \mathbf{z}_1 \mathbf{z}_1^\top \mathbf{z}_2\right] \\
&= \mathbf{z}_2^\top \, \mathbb{E}_{\mathbf{z}_1}[\mathbf{z}_1 \mathbf{z}_1^\top] \, \mathbf{z}_2 \\
&= \lambda \|\mathbf{z}_2\|^2 \\
&= \frac{1}{d} \quad \text{since } \|\mathbf{z}_2\| = 1
\end{aligned}$$

Thus the expectation is constant, and we obtain:

$$\mathbb{E}[\langle \mathbf{z}_1, \mathbf{z}_2 \rangle^2] = \frac{1}{d}$$

where d is the latent dimensionality.

This result shows that as d increases, independent latent vectors become nearly orthogonal. Consequently, in high-dimensional settings (e.g., $d = 64$ as in the original paper), the probability of drawing two nearly aligned latent vectors becomes negligible, which explains why the authors did not encounter convergence issues or exploding losses.

The near-orthogonality of independent random vectors is discussed in the book High Dimensional Probability [12], in subsection 3.2.4. However, the author does not provide a direct proof for this specific result, so I decided to derive it independently.

In the original paper, the authors explained that they added the diversity term to improve the responsiveness of the low-level policy with respect to changes in $\mathbf{z}$. In my view, the issues they experienced were likely due to the large differences between certain skills. For example, the low-level policy was trained to switch between striking an object with a sword and adopting a sneaking position while walking. The significant difference

between these skills may have caused the policy to sometimes respond poorly to changes in the latent variable, which the diversity term was intended to mitigate.

With the small modification I introduced, the low-level policy now produces results that are easier to interpret. In particular, I can represent skills on a circle during simulation: since the latent space is circular, each skill corresponds to an angle, which I map to a color, as shown in Figure 2.3. In the accompanying video 2D_walk_forward_backward.mp4, the selected skill is visualized by its associated color, directly reflecting the angle θ.

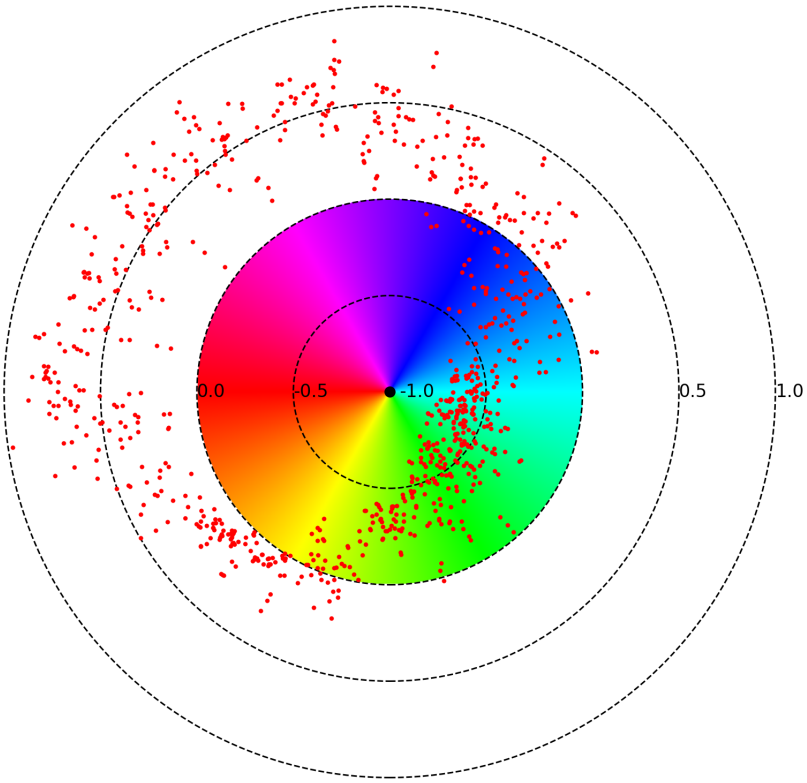

Figure 2.3: Color wheel representation of the latent space. Each hue corresponds to a latent direction (angle θ), while the red points indicate measured velocities for sampled latent vectors. The concentric dashed circles denote velocity magnitudes in m/s, with the innermost point corresponding to -1 m/s, the unit circle to 0 m/s, and the outer circles to positive speeds. The organization of the latent space shows that green-cyan-blue regions correspond to backward walking, yellow-marine blue to standing, and red-magenta to forward walking.

Fortunately, the high-level policy is highly effective at selecting appropriate skills, even when the latent space is not perfectly continuous or contains out-of-distribution variables. For instance, in Figure 2.3, certain points in the red-magenta region correspond to nearly zero velocity. However, the high-level policy consistently avoids these regions, instead favoring the blue or yellow zones.

This demonstrates that the latent space does generalize, providing valuable insight

into how ASE operates. In this toy example, the mechanism is relatively easy to interpret, but the question remains: what happens when the problem is made more complex?

2.2 Scaling ASE to complex robotic tasks

One of the most challenging aspects of my internship was identifying an innovative yet feasible application of ASE to robotics. Compared to the simple forward–backward walking policy, any policy requiring the coordination of multiple skills already introduces greater complexity. My initial goal was to gain a clear understanding of the latent space and how it generalizes to more complex motions. Even the straightforward walking-and-turning policy presented a meaningful challenge.

I therefore began training a low-level policy using motion clips from both the LAFAN1 dataset and our own dataset, recorded with a motion capture (Mocap) system at JRL. The objective was to gain a deeper understanding of how the latent space is structured and, in particular, to determine the most appropriate latent dimension for a given task.

The Mocap data was collected at JRL using an array of infrared cameras and reflective markers. The process begins with camera calibration, performed using a square frame with attached markers. The subject then dons a suit fitted with markers and adopts a T-pose, allowing the system to register the marker positions as a human skeleton. This skeleton representation encodes the joints of the human body, providing joint angles and a root position (typically the pelvis) rather than raw 3D point clouds. In addition, extra markers can be defined to represent objects in the scene, for instance, a door, as shown in the video OptiTrack_record.mp4.

This technology often suffers from uncontrolled glitches, such as the hands unnaturally flipping over themselves, producing motions that would require the wrist to rotate by 180 degrees. To simplify the movement and avoid such issues, we chose not to use the hands in our setup. Instead, the elbows were fixed at a 90-degree angle, thereby reducing retargeting problems.

Developing a walking-and-turning policy is not new in itself, but ASE offers distinct advantages, particularly in producing smoother transitions when commands change. In traditional RL, a common issue arises when a robot trained to walk is instructed to stand still: upon receiving a zero command, the robot often continues to step in place as it compensates for residual base velocities. ASE addresses this limitation by enabling a switch between a walking skill and a standing skill. The pipeline consists of first training a low-level policy on motion clips, followed by training the high-level policy using these rewards:

$$\begin{cases} r_t^{\text{lin}} = \exp\left(-\frac{(\dot{x}_{\text{command}}-\dot{x})^2+(\dot{y}_{\text{command}}-\dot{y})^2}{\sigma_{\text{tracking}}}\right) \\ r_t^{\text{ang}} = \exp\left(-\frac{(\dot{\theta}_{\text{z,command}}-\dot{\theta}_{\text{z}})^2}{\sigma_{\text{tracking}}}\right) \end{cases} \tag{2.2}$$

We also vary the commands regularly during high-level policy training, encouraging the policy to select the appropriate skills while maintaining balance.

2.2.1 Selecting the latent dimension

A key limitation of ASE lies in the need to manually choose the latent dimension, alongside other hyperparameters. Theoretically, the smallest latent space capable of representing two distinct skills is a sphere, since it provides two degrees of freedom: one for

linear velocity and one for angular velocity. This configuration is illustrated in Figure 2.4, which shows both the idealized and the learned latent space. In practice, however, the outcome proved more complex than expected.

The learned latent space for linear velocity roughly follows the anticipated structure: one side of the sphere corresponds to high forward velocity, while the opposite side corresponds to slower motion or standing. Angular velocity, on the other hand, does not align with the expected distribution, as detailed in the figure caption. This discrepancy arises because ASE provides no direct mechanism to enforce generalization along a specific motion component. For example, in the walking clips, the actor moves their arms in varying ways. The pipeline may have interpreted these arm motions as the generalizable feature, thereby limiting the low-level policy's ability to capture angular velocity properly.

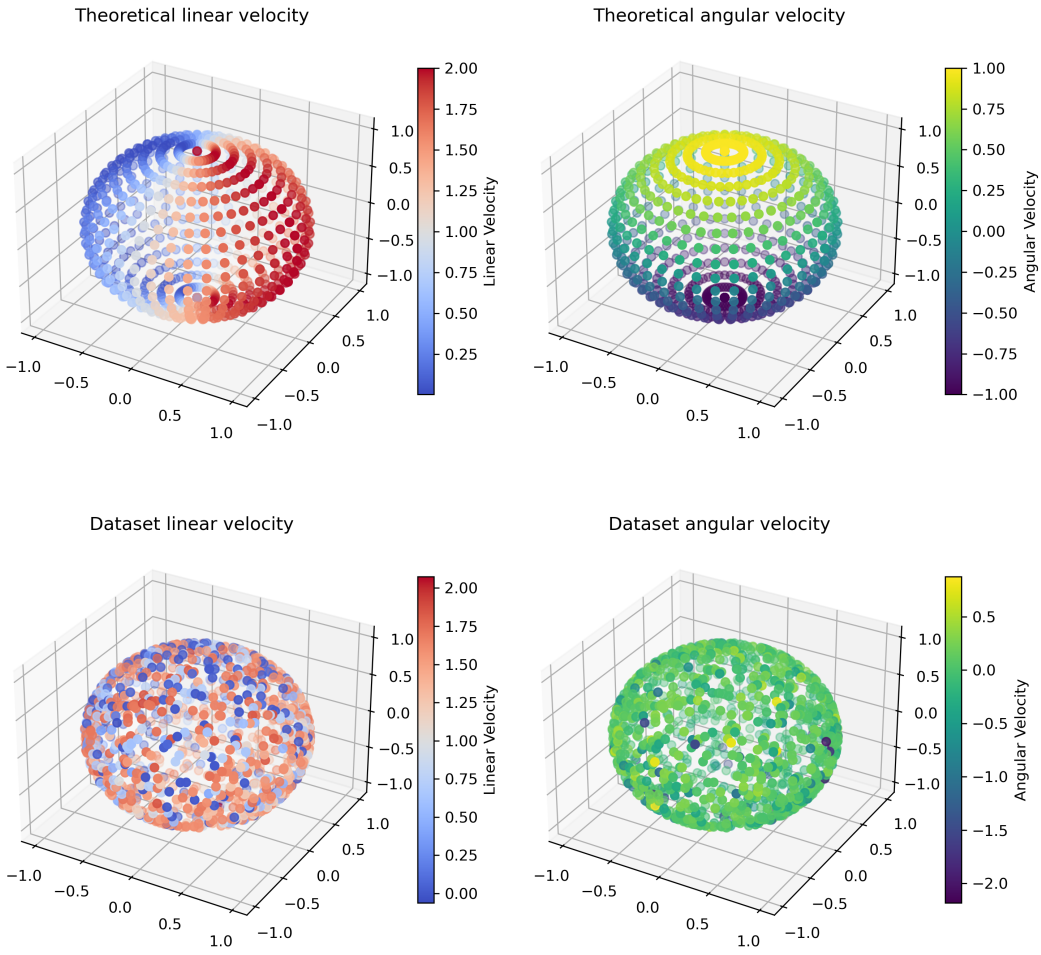

Figure 2.4: **Top left:** theoretical linear velocity, which is expected to vary smoothly along a circular angle inside the sphere, with one side corresponding to high speed and the opposite side to slower motion or standing still. **Bottom left:** real linear velocity, showing a similar pattern with two distinct regions (red for higher speed and blue for lower speed), though with some outliers. **Top right:** theoretical angular velocity, which can take a range of values for each possible linear velocity. **Bottom right:** real angular velocity, which does not follow the theoretical distribution.

This simple experiment highlights how difficult it is to constrain or fully interpret the latent space once a certain level of complexity is reached. Remarkably, the model even generalized to running (see video 3D_walk_lafan.mp4), despite the absence of any running clips in the dataset, meaning the encoder effectively inferred it on its own. When training was pushed further, unusual behaviors began to emerge, particularly in the arms (see video 3D_lafan_arms_torso_generalization.mp4). This confirmed my intuition that the model was generalizing walking motions with varying postures of the arms and torso, visible in the shifting shoulder positions while maintaining the same gait. Despite these artifacts, the transitions between skills remained impressively smooth and fluid.

We can conclude that too small a latent dimension does not allow for a proper walking-and-turning policy. To address this, I increased the latent dimension to six, which enabled a comparison of the two latent spaces and how the skills are distributed within them. The most immediate consequence of increasing the latent dimension in practice is longer training time, since the policy must now explore a larger space and assign a distribution to each of its regions.

To visualize how well the skills were clustered, I applied Uniform Manifold Approximation and Projection (UMAP) [13], as shown in Figure 2.5. This technique embeds the high-dimensional latent points into a 2D space, revealing the relationship between different regions of the latent space and the corresponding linear and angular velocities. Additional details are provided in Appendix A.2.

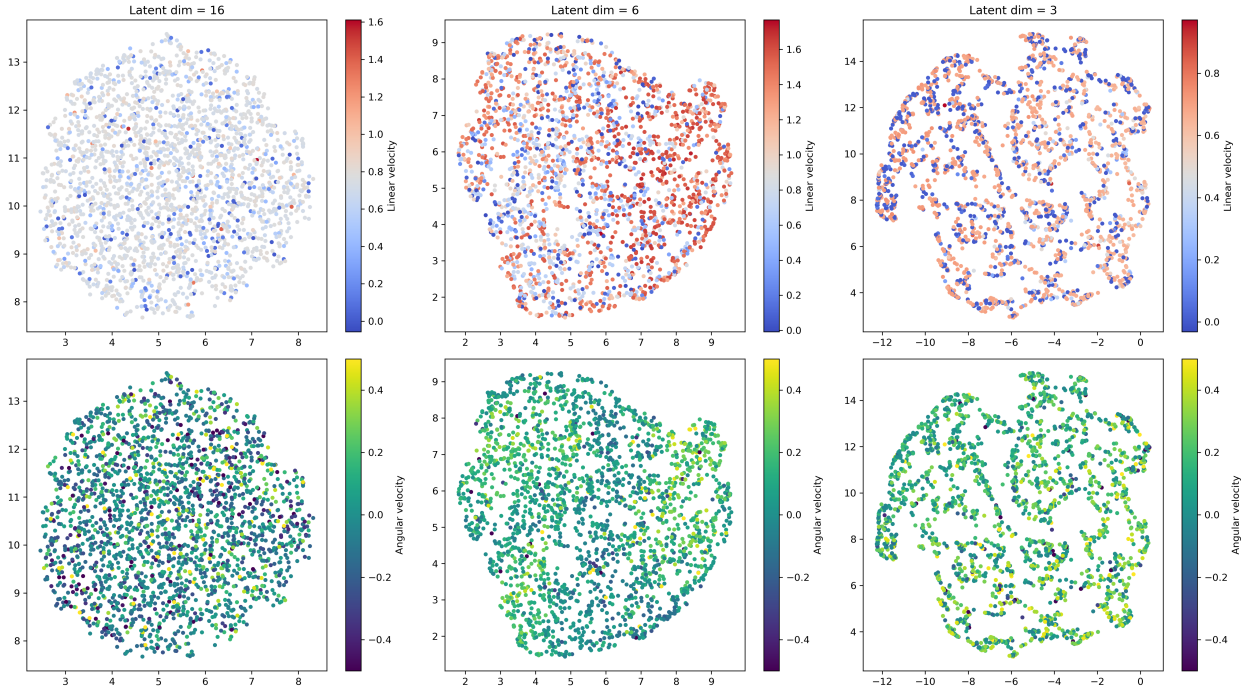

Figure 2.5: UMAP projections of latent spaces extracted from three different policies trained on the LAFAN1 dataset. Each column corresponds to a different latent dimensionality: **Left**: 16, **Middle**: 6, and **Right**: 3. Colors indicate linear velocity (top row) and angular velocity (bottom row). The 16-dimensional latent space shows poor generalization. The 6-dimensional latent space provides a more coherent clustering, capturing velocity information effectively. The 3-dimensional latent space, corresponding to Figure 2.4, shows weaker clustering of velocities, suggesting a loss of representation capacity.

When the latent space is too high-dimensional, the encoder almost always discovers

a new region that it associates with a skill. Instead of generalizing by selecting latent variables close to an already identified area, it jumps to a completely different part of the space and assigns it to the same skill. Over time, this leads to a fragmented latent space, where multiple distant regions correspond to nearly identical motions. As a result, the representation loses its usefulness, since most latent variables end up reproducing the same behavior rather than offering meaningful variation.

Conversely, when the latent space is too small, the skills are forced to cluster very close to one another. This leaves little room to differentiate between them, reducing the network's capacity to generalize. To avoid abrupt changes in the latent space, the policy instead learns to reproduce only closely related skills, at the cost of diversity.

We also begin to observe the impact of removing the diversity loss, as the latent space now contains too many outliers. This severely reduces the responsiveness of the low-level policy, particularly when trained on the LAFAN1 dataset. One issue with LAFAN1 is that the actor's arms remain very close to the body, whereas the H1 robot walks more naturally with bent elbows, which helps it compensate for momentum. In addition, the linear velocities in LAFAN1 are unrealistically high for the H1's capabilities; slower walking motions would be more appropriate.

As discussed earlier, the diversity loss plays a key role in ensuring smooth transitions between skills, even when they are far apart in the latent space. Without it, I had to design a dataset where the skills were inherently easier for the robot to transition between. Using the retargeting pipeline described in Section 2.1.2, I therefore built a new dataset where the actor walks more slowly, making it easier for the robot to adjust its velocity since the skills lie closer together in the latent space.

Consequently, I conducted tests using our own Mocap data recorded at JRL, and the results were significantly better. This improvement stems from the slower walking pace in these clips: the steps are shorter and therefore better respect the physical constraints of the robot. Qualitatively, responsiveness improved substantially, even though the walking style appears less human-like. Nevertheless, the motion is much smoother than with classical RL methods, and the transition from walking to standing is markedly improved.

A UMAP visualization of the latent space is shown in Figure 2.6, illustrating the improved organization with well-separated regions that correspond to distinct skills and enhanced generalization capacity. To further highlight this property, I trained a high-level policy using the reward formulation from (2.2).

The video high_level_policy_mocap_dataset.mp4 provides a qualitative demonstration of these improvements: the policy responds clearly to commands while maintaining smooth transitions between skills.

2.2.2 Sim-to-real transfer on the H1 robot

Deploying a policy trained in simulation onto a real robot often exposes a gap between simulated and real-world conditions, commonly referred to as the sim-to-real gap. Because the simulation environment differs in many ways from reality, the transferred policy typically performs poorly. This gap can arise from various sources: the policy may exploit artifacts specific to the simulator, the robot's mass and inertia may differ from the model, or there may be additional delays between issuing commands and their execution on the real hardware. In practice, nearly any mismatch between simulation parameters and the physical system can contribute to the sim-to-real gap.

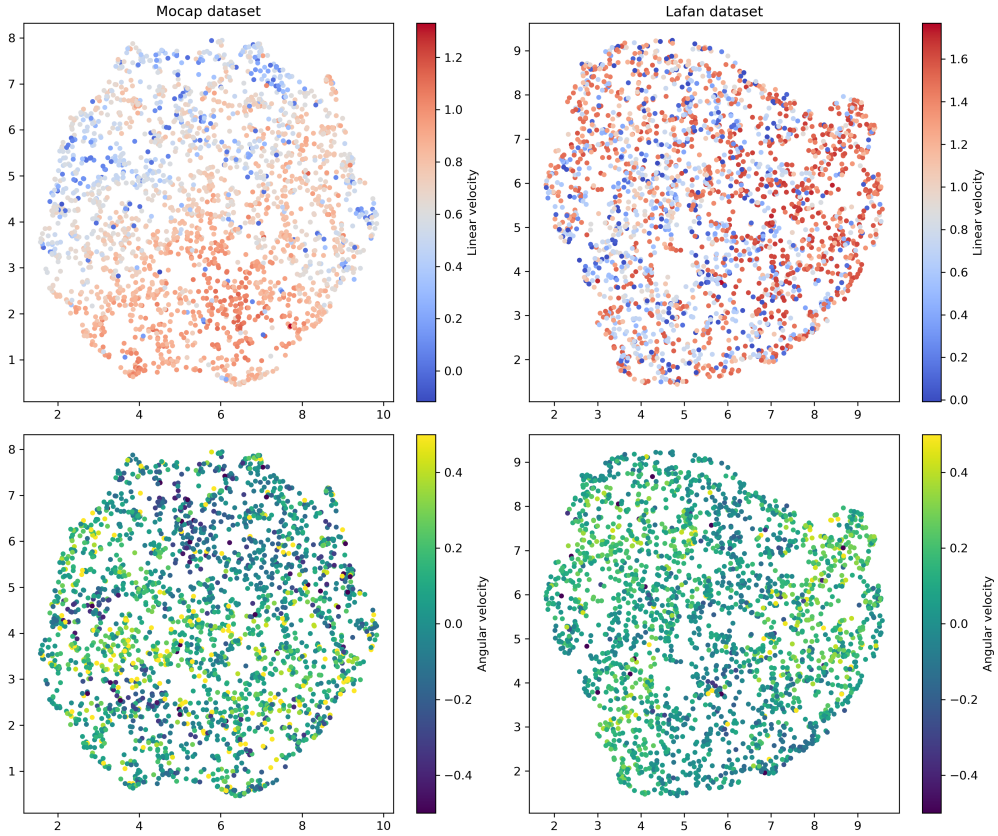

Figure 2.6: UMAP projections of latent spaces learned from two different datasets. **Left**: the Mocap JRL dataset. **Right**: the LAFAN dataset (same as in Figure 2.5). Both are trained with 6-dimensional latent space. The latent space obtained from the Mocap JRL dataset exhibits a clearer and more structured organization of both linear and angular velocities compared to the LAFAN dataset.

To make the policy robust to the sim-to-real gap, we introduce variability and corrections during training through the following techniques:

- **Domain randomization:** Several physical parameters of the Unitree H1 robot differ between simulation and reality. For example, at JRL we measured the robot's mass to be 58 kg, while the URDF specifies 51.7 kg. To address this discrepancy, we apply a random mass shift at the start of each training episode, allowing the policy to remain robust to changes in mass and dynamics. Similarly, joint actuation is modeled with a Proportional–Derivative (PD) controller. Given a target position $q^\star$ and current position q, the control command is

  $$\tau = K_p(q^\star - q) + K_d(\dot{q} - 0),$$

  where K_p determines the stiffness of the response to position error, and K_d damps or anticipates error growth. Since these gains differ in reality, we randomize K_p and K_d across episodes to improve robustness.

- **Observation noise:** Simulation often provides unrealistically accurate sensor values. To mimic real-world uncertainty, we add noise to each observation, using noise levels provided by the open-source Unitree Mujoco framework [14], which was specifically designed to address sim-to-real transfer.

- **Privileged observations:** In previous experiments, both the Actor (policy) and Critic (value function) used the same set of observations. However, in simulation we can access information not available on the real robot, such as the base linear velocity. To accelerate learning, we provide this privileged information only to the Critic. On the real robot, only the Actor runs, and thus it does not rely on unavailable signals.

- **Action delays:** In practice, action commands take longer to be executed on the real robot than the computation time of the policy. This mismatch can cause the policy to accumulate errors and produce unstable behaviors. To mitigate this, we apply the action computed at time t with a fixed delay Δt, here chosen as $\Delta t = 4$. This encourages the policy to anticipate the real-world latency.

After training, the policy can communicate and send commands through the open-source framework Unitree Mujoco [14], allowing us to evaluate its performance under quasi-real conditions in the Mujoco simulator [15] (see video MuJoCo_Deployment.mp4). However, incorporating all these sim-to-real gap mitigation techniques comes at a cost: they significantly affect the stability of the framework and, in particular, the structure of the latent space. Figure 2.7 presents a UMAP visualization of the latent space obtained with the Mocap dataset under these variations, to be compared with the cleaner structure observed in Figure 2.6.

The sim-to-real gap mitigation tools completely disrupted the original organization of the latent space. The added noise prevented the encoder from clustering skills properly, exposing the limitations of applying ASE under real-world conditions. From this point on, tuning became a tedious search for hyperparameters that might allow the framework to remain functional in sim-to-real settings. Despite the disorganized latent space, we proceeded with a deployment on the H1 robot. By manually selecting two latent variables corresponding to forward and backward walking, I was able to test how well the policy transferred to the real platform.

This approach produces human-like walking (see video real_forward_backward.mp4), with vibrations significantly reduced by applying an exponential moving average. Compared to more classical RL policies, the walking policy produces discrete steps and is able to stop smoothly (see video real_forward_backward_2.mp4). In a final attempt, I added a penalty in the low-level policy for arm vibrations, which then caused the legs to vibrate instead. I suspect this behavior arises from the GAN architecture: at certain points, the network may exploit small vibrations to fool the discriminator.

2.2.3 Exploring ASE for contact-rich tasks

As highlighted in [16], "*RL with physical simulators [ASE is cited here] has been used to produce physically plausible movements but faces challenges in generalizing across varied scenes and objects.*" This limitation makes ASE poorly suited for tasks that require precise interaction with the environment. Nevertheless, I decided to experiment with this direction by adapting ASE using ideas drawn from related work.

As discussed in Section 2.2.1, the choice of latent dimensionality is critical, and ASE tends to perform better on simpler, more generalizable movements. Door opening, however, is inherently a precise motion rather than a broadly generalizable one. For example, if we train directly on a motion clip of an actor opening a door, ASE may not generalize

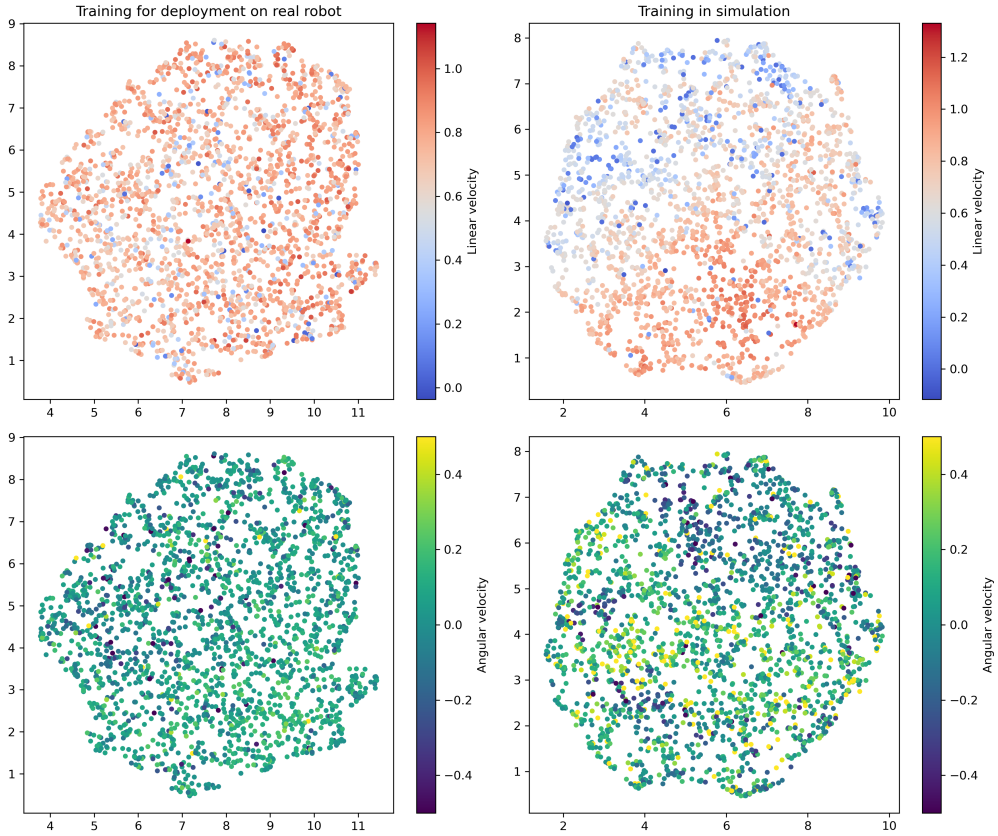

Figure 2.7: UMAP projections of the latent spaces learned from the Mocap dataset with a 6-dimensional latent representation. **Left**: Training under deployment conditions on the real robot. **Right**: Training in simulation (same as Figure 2.6, left). Compared to the simulation setting, the structure observed in the latent space is largely lost under deployment, likely due to noise and domain randomization.

to different opening speeds, but it could capture the general arm placement on the handle. To reduce complexity, I focused specifically on the simpler variant of pushing doors rather than turning a handle

My idea was that opening a door could be approached as a combination of walking and pushing, making it a natural candidate for ASE. The low-level policy handles locomotion, while the adaptation occurs at the high-level policy, which is extended to incorporate arm use for pushing the door. To achieve this, I added a few additional rewards, with the main objective being to encourage successful door opening. Since I had access to Mocap clips of an actor performing the task, I extracted the door angle and angular velocity from these recordings.

These data were then used to design an imitation reward specifically tailored to the door. In practice, I trained a discriminator during high-level policy training to distinguish whether the door motion originated from the actor's Mocap performance or from the robot in simulation. To replicate the door dynamics as realistically as possible, I modeled its physics with a simple force equation:

$$F = -k\theta + b\dot{\theta} \tag{2.3}$$

where the first term represents the spring-like restoring force and the second accounts for friction.

The overall pipeline is illustrated in Figure 2.8. Compared to the standard ASE framework, two main modifications are introduced: (i) an additional discriminator is trained to provide an imitation reward specifically for the door-opening motion, and (ii) a residual action term is added to correct the arm's position, ensuring effective physical interaction with the door by applying a pushing force.

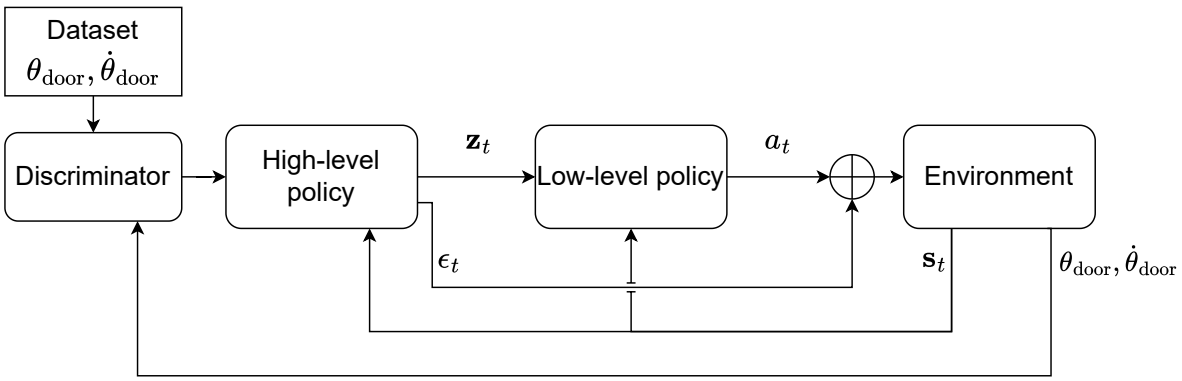

Figure 2.8: Modified ASE pipeline for the door-opening task. The high-level policy generates latent variables $\mathbf{z}_t$ for the low-level controller, which outputs actions a_t along with a residual correction ϵ_t. The final executed action is $a_t + \epsilon_t$. A door-specific discriminator provides imitation rewards by comparing the simulated motion with Mocap demonstrations of door opening.

The results obtained with this approach remain inconclusive. While the robot is able to walk, it does not consistently apply effective force on the door (see video residuals_door.mp4). In some cases, the door is opened, but the robot fails to pass through, often becoming stuck in the door frame. The residual terms ϵ_t remain very small in magnitude throughout training and show little variation, as the low-level policy alone performs better. Consequently, the optimization favors suppressing the residuals, which in practice only introduces additional vibrations. This indicates that the proposed pipeline is ineffective in its current form. While more carefully designed rewards might improve performance, the results highlight a limitation of ASE in terms of generalization.

As a comparison, I implemented a simple imitation reward using motion capture clips that I recorded myself (see OptiTrack_record.mp4). I chose to create my own dataset because the imitation pipeline is highly sensitive to the robot's kinematic constraints. Using human motion clips would not be suitable, as the robot has fewer degrees of freedom and cannot accurately reproduce such trajectories. To address this, I recorded two specific demonstrations: one where I open a door at a normal pace, and another where I perform the motion more slowly.

When I used human walking motions to condition the door-opening discriminator, the robot initially learned to walk and then to push the door. However, as training progressed, the walking behavior started to deteriorate. As shown in door_impacting_walk.mp4, the policy modifies its gait in order to fool the discriminator, effectively "forgetting" how to walk properly in favor of improving its door-opening score.

In contrast, door_robotic_opening.mp4 demonstrates that using a robotic-specific opening motion results in more stable behavior, though the movements remain rigid and lack natural fluidity. I could have continued working on this task, but doing so would have required extensive reward shaping rather than exploring true generalization. This leads

to the conclusion that generative adversarial architectures are not well suited for tasks requiring precise loco-manipulation.

Chapter 3

Conclusion and Perspectives

This internship has been a rewarding experience, combining both theoretical and practical aspects of reinforcement learning with humanoid robotics. What makes the JRL truly unique is the opportunity to work directly with a real humanoid robot, and I feel fortunate to have been able to contribute to this area. Through this experience, I deepened my understanding of Generative Adversarial Networks and Autoencoders while applying them to real-world robotics problems.

During this internship, my focus was on applying Adversarial Skill Embeddings (ASE) to humanoid locomotion using the Unitree H1 robot. The work involved a combination of simulation-based training of multi-skill policies and addressing the challenges of transferring these policies to a real robot. I implemented ASE pipelines, investigated the impact of latent dimensionality on skill representation, explored the role of diversity loss, and developed a retargeting pipeline to adapt motion capture datasets for the robot's kinematic constraints. These efforts resulted in policies capable of producing human-like walking behaviors with smooth transitions, while also highlighting the limitations of ASE in practical applications.

Despite its potential, ASE remains a complex and labor-intensive framework. Its sensitivity to noise and new skills makes it challenging to deploy robustly, and it has largely been superseded by newer guided diffusion policies [17] that achieve superior performance for multi-skill tasks. In practice, ASE is not a scalable solution for generating general locomotion policies, and it is even less suitable for precise loco-manipulation tasks.

Looking forward, the current trend for learning highly generalizable policies is exemplified by Forward-Backward Representations [18]. This framework can provide near-optimal policies for arbitrary rewards specified a posteriori, and it has shown remarkable results in character simulation. It represents a promising direction for the development of generalized locomotion and multi-skill policies.

On a personal note, this internship solidified my decision to pursue a PhD. I will continue my research at the Institut des Systèmes Intelligents et de Robotique (ISIR) at La Sorbonne, Paris, focusing on improving sample efficiency in reinforcement learning by leveraging differential equations of the system state and by using generative models.

Appendices

A.1 Proximal Policy Optimization

As mentioned in the introduction, there are two main ways to update a policy: (i) by computing a value function and then optimizing with respect to the policy parameters in order to maximize it, or (ii) by directly optimizing the reinforcement learning objective (1.1) with respect to the policy parameters.

Let us denote the policy by π_θ, where θ are the parameters. The objective is:

$$\max_\theta J(\theta) = \mathbb{E}_{p(\tau|\pi_\theta)}\left[\mathbf{R}(\tau)\right],$$

where τ is a trajectory and $\mathbf{R}(\tau)$ its return.

To optimize with respect to θ, we need the gradient of $J(\theta)$, which is given by the policy gradient theorem:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \, \mathbb{E}_{p(\tau|\pi_\theta)}[\mathbf{R}(\tau)] \\
&= \int_\tau \mathbf{R}(\tau) \nabla_\theta p(\tau|\pi_\theta) \qquad \text{since } \mathbf{R}(\tau) \text{ is independent of } \theta \\
&= \int_\tau \mathbf{R}(\tau) p(\tau|\pi_\theta) \frac{\nabla_\theta p(\tau|\pi_\theta)}{p(\tau|\pi_\theta)} \\
&= \mathbb{E}_{p(\tau|\pi_\theta)}\left[\mathbf{R}(\tau) \, \nabla_\theta \log p(\tau|\pi_\theta)\right].
\end{aligned}$$

The probability of a trajectory τ under π_θ can be written as:

$$p(\tau|\pi_\theta) = \prod_{t=0} p(s_{t+1}|s_t, a_t) \, \pi_\theta(a_t|s_t).$$

Therefore,

$$\begin{aligned}
\nabla_\theta \log p(\tau|\pi_\theta) &= \nabla_\theta \left(\sum_{t=0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t) \right) \\
&= \sum_{t=0} \nabla_\theta \log \pi_\theta(a_t|s_t),
\end{aligned}$$

since the environment dynamics $p(s_{t+1}|s_t, a_t)$ do not depend on θ.

Thus, the gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{p(\tau|\pi_\theta)} \left[\mathbf{R}(\tau) \sum_{t=0} \nabla_\theta \log \pi_\theta(a_t|s_t) \right].$$

Equivalently, this can be written as

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\mathbf{R}(\tau) \, \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})\right],$$

where $\pi_\theta(\mathbf{a}|\mathbf{s})$ is the probability of the action sequence $\mathbf{a}$ given the state sequence $\mathbf{s}$.

In practice, this expression can suffer from high variance and unstable updates, since

$$\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) = \frac{\nabla_\theta \pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_\theta(\mathbf{a}|\mathbf{s})},$$

which may lead to very large gradients. Occasional large steps in the optimization can cause divergence (see footnote 1).

To address this, we replace returns by advantages, which measure how much better an action is compared to the average action in a given state. Positive advantages encourage the policy to favor the corresponding actions, while negative ones discourage them.

Proximal Policy Optimization (PPO) constrains policy updates to remain close to the previous policy, preventing excessively large steps. To do so, we define the probability ratio:

$$r(\theta) = \frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{\text{old}}}(\mathbf{a}|\mathbf{s})},$$

where $\pi_{\theta_{\text{old}}}$ is the policy before the update.

The clipped surrogate objective is:

$$J(\theta) = \mathbb{E}\left[\min\left(r(\theta)\mathbf{A},\ \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\,\mathbf{A}\right)\right],$$

where $\mathbf{A}$ is the advantage and ϵ is a small hyperparameter.

This objective ensures that if the ratio $r(\theta)$ deviates too far from 1, it is clipped, thereby preventing overly large policy updates and improving training stability.

A.2 Uniform Manifold Approximation Projection

Uniform Manifold Approximation and Projection (UMAP) [13] is a dimensionality reduction technique that is particularly well-suited for visualizing high-dimensional data. The key idea behind UMAP is to construct a high-dimensional graph representing the data manifold and then optimize a low-dimensional embedding that preserves the local structure of the data. The algorithm can be summarized in the following steps:

1. **Graph construction:** Each data point is connected to its nearest neighbors, creating a weighted graph where the edge weights represent the closeness or similarity between points. This graph captures the local topological structure of the high-dimensional space.

2. **Fuzzy simplicial set:** UMAP interprets this graph probabilistically by defining a fuzzy topological structure[1] that represents the likelihood of points being connected. This helps preserve both local and global structure.

3. **Optimization:** The algorithm finds a low-dimensional embedding (typically 2D or 3D) by minimizing the cross-entropy between the high-dimensional fuzzy graph and the low-dimensional representation. This ensures that points that are close in high dimensions remain close in the low-dimensional space, while distant points are appropriately separated.

[1] In this context, "fuzzy" means that the connections between points are not strictly binary but weighted by a probability representing how likely two points are neighbors on the manifold.

Bibliography

[1] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[2] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[3] Vijay Konda and John Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[4] Xue Bin Peng et al. "AMP: adversarial motion priors for stylized physics-based character control". In: *ACM Transactions on Graphics* 40.4 (July 2021), pp. 1–20. ISSN: 1557-7368. DOI: 10.1145/3450626.3459670. URL: http://dx.doi.org/10.1145/3450626.3459670.

[5] Unitree Robotics. *Unitree H1 / H1-2 Humanoid Robot*. https://www.unitree.com/h1. Accessed: 2025-08-14. 2025.

[6] Dewei Wang et al. *MoRE: Mixture of Residual Experts for Humanoid Lifelike Gaits Learning on Complex Terrains*. 2025. arXiv: 2506.08840 [cs.RO]. URL: https://arxiv.org/abs/2506.08840.

[7] Xue Bin Peng et al. "ASE: large-scale reusable adversarial skill embeddings for physically simulated characters". In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–17. ISSN: 1557-7368. DOI: 10.1145/3528223.3530110. URL: http://dx.doi.org/10.1145/3528223.3530110.

[8] Genesis Authors. *Genesis: A Generative and Universal Physics Engine for Robotics and Beyond*. Dec. 2024. URL: https://github.com/Genesis-Embodied-AI/Genesis.

[9] Nikita Rudin et al. "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning". In: *Proceedings of the 5th Conference on Robot Learning*. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2022, pp. 91–100. URL: https://proceedings.mlr.press/v164/rudin22a.html.

[10] H. Lv. *LAFAN1 Retargeting Dataset*. Hugging Face dataset. https://huggingface.co/datasets/lvhaidong/LAFAN1_Retargeting_Dataset. accessed in 2025.

[11] Chung Min Kim* et al. *PyRoki: A Modular Toolkit for Robot Kinematic Optimization*. 2025. arXiv: 2505.03728 [cs.RO]. URL: https://arxiv.org/abs/2505.03728.

[12] Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2018.

[13] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML]. URL: https://arxiv.org/abs/1802.03426.

[14] Unitree Robotics. *unitree_mujoco: A simulator based on Unitree SDK2 and MuJoCo*. https://github.com/unitreerobotics/unitree_mujoco. GitHub repository, accessed: 2025-09-26. 2025.

[15] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[16] Hongwei Yi et al. *Generating Human Interaction Motions in Scenes with Text Control*. 2024. arXiv: 2404.10685 [cs.CV]. URL: https://arxiv.org/abs/2404.10685.

[17] Qiayuan Liao et al. *BeyondMimic: From Motion Tracking to Versatile Humanoid Control via Guided Diffusion*. 2025. arXiv: 2508.08241 [cs.RO]. URL: https://arxiv.org/abs/2508.08241.

[18] Ahmed Touati and Yann Ollivier. *Learning One Representation to Optimize All Rewards*. 2021. arXiv: 2103.07945 [cs.LG]. URL: https://arxiv.org/abs/2103.07945.