GMM-Based Robot Locomotion Planning with Learnt Latent Representations

GMM-basierte Planung von Laufroboter-Bewegungen mithilfe gelernter latenter Repräsentationen

Master thesis by Joshua Johannson

Date of submission: October 06, 2025

1. Review: Ph.D. Oleg Arenz
2. Review: Prof. Ph.D. Jan Peters
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Joshua Johannson, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 06.10.2025

Joshua Johannson

Abstract

Legged locomotion has proven to be effective for robots navigating complex environments. Incorporation of reference motion datasets into learning locomotion controllers offers benefits in efficiency by introducing additional guidance towards desired motions. In this context, a core challenge lies in obtaining control policies that can solve diverse high-level tasks while imitating the reference motions, without expensive retraining for new tasks, which limits generalizability.

To address this, we introduce a trajectory optimization framework designed to distill a reference motion dataset into a task-agnostic planner and a low-level execution policy. The planner utilizes a Gaussian Mixture Model (GMM) to represent a distribution over multiple possible solutions, enabling the consideration of diverse, high-level locomotion trajectories. Optimization is performed in the latent space of a Variational Autoencoder-based general motion representation model, which is trained on the reference dataset. We investigate both autoregressive and latent spline trajectory-based architectures. Task objectives are incorporated via reward functions which are applied on the decoded trajectories. We formulate the reward maximization as a Variational Inference problem to enable distributional optimization of the GMM.

Evaluated on the Go2 quadrupedal robot, our framework demonstrates successful execution in tasks like goal reaching and planar obstacle avoidance, including stepping-aware planning. We validate the performance for open-loop execution (for long planning horizons) and closed-loop Model Predictive Control (for real-time planning) in simulation and real-world experiments.

Zusammenfassung

Die Fortbewegung mit Beinen hat sich als effektiv für Roboter erwiesen, die in komplexen Umgebungen navigieren. Das Einbeziehen von Referenzbewegungsdatensätzen in das Training der Fortbewegungssteuerung bietet Effizienzvorteile, indem diese zusätzliche Anleitungen für die gewünschten Bewegungen liefern. In diesem Zusammenhang besteht eine zentrale Herausforderung darin, Steuerungspolicies zu erhalten, die vielfältige übergeordnete Aufgaben lösen und gleichzeitig die Referenzbewegungen nachahmen, ohne dass für jede neue Aufgabe ein aufwendiges, erneutes Training erforderlich ist, was die Generalisierbarkeit einschränkt.

Hierfür stellen wir ein Trajektorienoptimierungs-framework vor, das einen Referenzbewegungsdatensatz in einen aufgabenunabhängigen Planer und eine Low-Level-Policy destilliert. Der Planer verwendet ein Gaussian Mixture Model (GMM), um eine Verteilung über mehrere mögliche Lösungen darzustellen. Dadurch ist es möglich, verschiedene High-Level Fortbewegungstrajektoren zu berücksichtigen. Die Optimierung wird im latenten Raum eines Variational Autoencoder (VAE) basierten allgemeinen Bewegungsrepräsentationsmodells durchgeführt, das auf dem Referenzdatensatz trainiert wurde. Wir untersuchen sowohl autoregressive als auch auf latenten Spline-Trajektorien basierende Architekturen. Aufgabenziele werden durch Rewardfunktionen integriert, die auf die dekodierten Trajektorien angewendet werden. Wir formulieren die Rewardmaximierung als ein Variational Inference (VI) Problem, um eine distributionelle Optimierung des GMM zu ermöglichen.

Evaluiert am vierbeinigen Roboter Go2, demonstriert unser Framework erfolgreiche Ausführung bei Aufgaben wie dem Erreichen eines Zielpunkts und planarer Hindernisvermeidung, einschließlich Planung unter Vermeidung unzulässiger Trittzonen. Wir validieren die Performance sowohl für die Open-Loop-Ausführung (für lange Planungshorizonte) als auch für die Closed-Loop Model Predictive Control (MPC) (für Echtzeitplanung) in Simulationsexperimenten und in Experimenten am echten Roboter.

Contents

Abbreviations and Symbols

List of Abbreviations

AE	Autoencoder
ELBO	Evidence Lower Bound
GMM	Gaussian Mixture Model
KL	Kullback-Leibler Divergence
MDP	Markov Decision Process
MLP	Multilayer Perceptron
MPC	Model Predictive Control
NN	Neural Network
PPO	Proximal Policy Optimization
QP	Quadratic Programming
RL	Reinforcement Learning
SRBD	Single Rigid Body Dynamics
TRPO	Trust Region Policy Optimization
VAE	Variational Autoencoder
VI	Variational Inference
VQ-VAE	Vector-Quantized Variational Autoencoder
WBC	Whole Body Controller

List of Symbols

$\mathcal{S}$	continuous state space
$\mathcal{A}$	continuous action space
$\boldsymbol{s}$	system state (in general $\boldsymbol{s} \in \mathcal{S}$)
$\boldsymbol{a}$	system control action (in general $\boldsymbol{a} \in \mathcal{A}$)
t	time
m	mass
$\boldsymbol{\varphi}$	Euler angles, $\boldsymbol{\varphi} \in \mathbb{R}^3$
$\dot{\boldsymbol{\varphi}}$	angular velocity as Euler angle change rate, $\dot{\boldsymbol{\varphi}} \in \mathbb{R}^3$
$\boldsymbol{\omega}$	angular velocity, $\boldsymbol{\omega} \in \mathbb{R}^3$
$\boldsymbol{p}$	position vector
$\boldsymbol{q}$	joint values, $\boldsymbol{q} \in \mathbb{R}^{N_q}$ where N_q is the number of joints
$\boldsymbol{z}$	latent vector
D_z	number of latent dimensions
$\boldsymbol{\tau}_z$	latent trajectory, $\boldsymbol{\tau}_z = (\boldsymbol{z}_0, ..., \boldsymbol{z}_t, ...)$
$\boldsymbol{\theta}$	parameter vector

1 Introduction

Legged locomotion for robots has proven effective for various scenarios, such as navigating complex environments and performing dynamic motions. Recent research enabled learning dynamic controllers for humanoid and quadrupedal robots based on Reinforcement Learning (RL). A promising aspect in this field lies in the incorporation of given reference motions into the learning process, introducing additional guidance towards desired motions and improving learning efficiency. A common challenge in this context is how to reuse the given low-level references and behaviors for learning different higher-level tasks such as navigation. The reference motions are generally given as a dataset either coming from recorded real-world animal data, trajectory optimization or from recording motions from other given locomotion policies. Here, many works focus on fully learning-based approaches where task-specific models are learned [1–5]. For example, in [1], an encoder-decoder-based policy is trained to imitate given reference motions. To realize high-level tasks, the previously learned reference encoder is replaced by a task-specific module trained to solve the given task. Similarly, in [2], a pretrained low-level motion imitation policy for quadrupedal locomotion is instructed by a second high-level policy trained on a specific task setting. Motion planning based on reference trajectories can also be achieved with kinematics-based approaches, where we learn a high-level representation and/or a planning module solely based on the data without employing RL. For example, in [3], a diffusion planning model trained on the reference dataset, produces trajectories which are then executed by a separate low-level policy. Note that this method requires a text-annotated dataset for training, since the planner receives text-based instructions. Note that all these approaches generally require retraining for new and previously unseen tasks. Here, we do not want to focus on learning specific tasks, but instead allow for generalizability to new tasks after the training process.

To address this challenge, we present our trajectory optimization framework that allows to distill a given motion dataset into a high-level task-agnostic planner and low-level execution policy. Our approach allows considering multiple possible solutions during the optimization by employing a Gaussian Mixture Model (GMM) to represent a whole distribution over possible trajectories. The individual components of the GMM distribution can thereby focus on different regions of the solution space. This is useful e.g. in a navigation setting where different possible trajectories exist that lead towards a given goal and solve the task. We perform the optimization in the latent space of a task-agnostic motion representation model which is trained on the given reference dataset. In this work, we consider two different Variational Autoencoder (VAE) [6] based architectures for the motion model. The former autoregressive VAE model is based on [7] and encodes trajectory segments of a specific length with latent vectors. The latter spline VAE model considers temporal latent space trajectories represented by splines. It incorporates a polar phase-based representation to address the periodic nature of locomotion motions. Both motion models allow representing locomotion trajectories of arbitrary lengths where the number of latent dimensions is proportional to the considered time horizon.

During the optimization, different tasks are represented by reward functions within our framework. Each reward acts on the absolute locomotion trajectories resulting from decoding latent

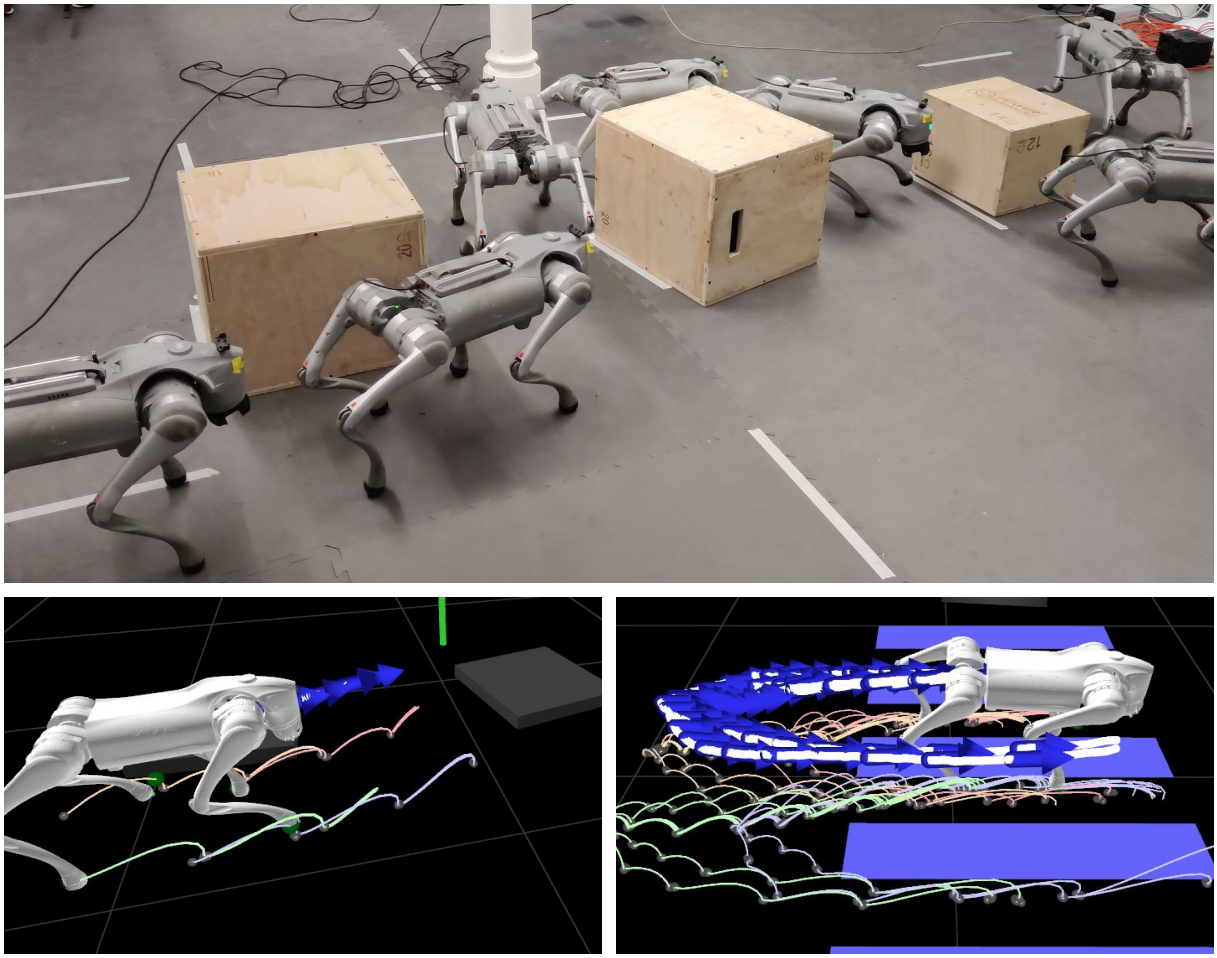


Figure 1: Real-world deployment of our framework. Top image: multiple time steps of closed-loop execution for an obstacle avoidance slalom task. Bottom left: planned trajectory towards the green goal at one time step of the slalom task, shown in the top picture. We visualize planned feet and base trajectories. Bottom right: planned trajectories from different GMM components for another stepping-aware planning task with long horizon and open-loop execution. Here, the robot is supposed to not step onto the blue areas on the ground while walking from the left to a goal on the right. The various components consider different solutions.

samples from the current GMM distribution. For example, a task reward function may encourage trajectories to end at a specified goal location or penalize collisions with obstacles. To enable distributional optimization, we frame the task reward maximization as Variational Inference (VI) problem, where the total reward is considered as the target distribution. Thereby, higher rewards correspond to regions with high density. We base the optimization of the GMM on the work of Arenz et al. [8]. To execute planned trajectories we train a RL-based low-level policy on the reference dataset.

For the evaluation of our approach, we consider the quadrupedal robot Go2. The reference motion dataset is generated from a simple heading direction instructed optimal control policy. After learning the motion representation model and low-level policy, our framework can imitate the motions of the original heading policy but has additional planning capabilities. We consider two different deployment configurations. First, for longer predictive planning horizons, i.e. 6s, we execute planned trajectories open-loop. Second, for a shorter planning horizon, i.e. 2s, where

the optimization is real-time capable, we perform closed-loop Model Predictive Control (MPC). Here, after a previously planned trajectory is executed by the low-level policy, we replan with the new robot state as start condition. We perform evaluation experiments and demonstrate that our framework is capable of performing different tasks, such as goal reaching and simple planar obstacle avoidance. Additionally, we demonstrate stepping-aware planning capabilities where the robot is not allowed to step into certain areas on the ground. We demonstrate our framework on simple flat terrain, both in simulation and in real-world experiments, see Figure 1.

2 Foundations

In this chapter, we describe the relevant concepts required to implement our framework. First, we discuss RL required to learn our low-level tracking policy, see Section 2.1. In Section 2.2, we introduce VAEs which are essential for our motion representation models. Subsequently, we describe VI with GMMs, on which our optimization algorithm is based, in Section 2.3. Finally, we discuss the related work in Section 2.4.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a foundational framework that enables learning behaviors (agents) through interaction with the environment and a learning feedback signal. More specifically, the environment is defined by a Markov Decision Process (MDP), which is given by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. The MDP defines the transition dynamics $p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$, specifying the probability of moving from a previous state $\boldsymbol{s} \in \mathcal{S}$ to a next state $\boldsymbol{s}' \in \mathcal{S}$ given action $\boldsymbol{a} \in \mathcal{A}$. The action is chosen by the agent. The behavior of the agent is described by its policy, a distribution over actions $\boldsymbol{a}$ in a given state $\boldsymbol{s}$, $\pi(\boldsymbol{a}|\boldsymbol{s})$. To enable learning, the MDP provides a feedback signal (the reward $r(\boldsymbol{s}, \boldsymbol{a}) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$). The reward is a scalar that defines how profitable it is to be in state $\boldsymbol{s}$ and performing the action $\boldsymbol{a}$ in that state. Additionally, the MDP defines the discount factor γ which indicates the importance of rewards received in the distant future relative to those received in the near future. $\mathcal{S}$ and $\mathcal{A}$ denote the sets of possible states and actions in the MDP. Figure 2 shows this agent-environment interaction process in more detail.

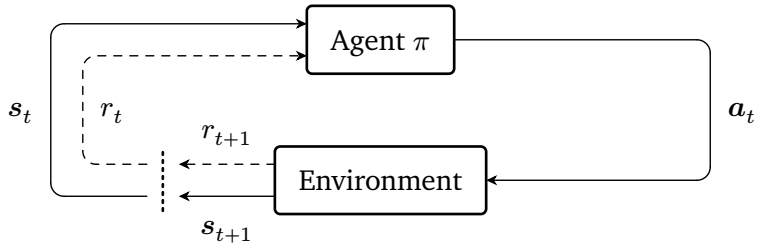


Figure 2: The agent interacts with the MDP environment, by providing an action to be executed in the current state $\boldsymbol{s}_t$. The environment then transitions to the next state $\boldsymbol{s}_{t+1}$ and provides the agent with a reward $r_{t+1} = r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. Figure adapted from Sutton et al. [9].

One rollout ("chain of interactions") of the agent with the environment yields the trajectory $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, r_1, \ \boldsymbol{s}_1, \boldsymbol{a}_1, r_2, \ \ldots, \ \boldsymbol{s}_t, \boldsymbol{a}_t, r_{t+1}, \ \ldots)$. We can compute the total reward, also called the return J, of a trajectory τ as the sum of its discounted rewards.

$$J(\tau) = \sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{1}$$

Note that $J(\tau)$ corresponds to the total reward of one possible trajectory starting from $\boldsymbol{s} = \boldsymbol{s}_0$. To quantify the expected return when starting from state $\boldsymbol{s}$ considering not just one, but multiple possible future trajectories, we use the value function $V^\pi(\boldsymbol{s})$ for a given policy π. This function yields the expected return for all trajectories starting from state $\boldsymbol{s}$ when using policy π to select actions during each rollout.

$$V^\pi(\boldsymbol{s}) = \mathbb{E}_{\tau\sim\pi}[J(\tau) \mid \boldsymbol{s}_0 = \boldsymbol{s}] \tag{2}$$

From the value function, we can derive the Q-function, which corresponds to the expected return when starting from state $\boldsymbol{s}$ and taking action $\boldsymbol{a}$ in that state.

$$Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\tau\sim\pi}[J(\tau) \mid \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}] = r(\boldsymbol{s}, \boldsymbol{a}) + \gamma\mathbb{E}_{\boldsymbol{s}'\sim p(\cdot|\boldsymbol{s},\boldsymbol{a})}[V^\pi(\boldsymbol{s}')] \tag{3}$$

Our objective is to find a policy that maximizes the expected return V^π over all possible states. The various RL algorithms to find such a policy can be divided into two groups [10]. *Value-based* algorithms focus on learning the value or Q-function. The policy is then indirectly defined via the value or Q-function. For example, a "greedy" policy can be obtained by maximizing the Q-value ($\pi_{\text{greedy}}(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a})$). For better exploration, often the ε-greedy action is chosen, where we sample a random action with probability ε, instead of the action from the purely greedy policy . The latter *policy-based* algorithms directly learn the policy and may also learn a value or Q-function.

For simpler environments with discrete states and actions, the value function, Q-function or policy can be parameterized by a table. For example, SARSA [11] is a value-based algorithm that learns the Q-function. It iteratively chooses an ε-greedy action using the current Q-function and updates the Q-function afterward. SARSA employs TD-learning which updates the Q-function based on the observed transition $(\boldsymbol{s}, \boldsymbol{a}, r,\ \boldsymbol{s}', \boldsymbol{a}')$, see Equation 4. Here, $\boldsymbol{a}, \boldsymbol{a}'$ are chosen as ε-greedy actions in $\boldsymbol{s}$ and $\boldsymbol{s}'$. The update is scaled by α. Note that TD-learning can be defined for both the value and Q-function. It works by bootstrapping the value of the next state $\boldsymbol{s}'$ via the current value or Q-function itself. This yields the TD-target, which the Q-function is updated towards.

$$Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow Q(\boldsymbol{s}, \boldsymbol{a}) + \alpha\left(\underbrace{r + \gamma Q(\boldsymbol{s}', \boldsymbol{a}')}_{\text{TD target}} - Q(\boldsymbol{s}, \boldsymbol{a})\right) \tag{4}$$

Note that there are also value-based methods for continuous state and/or action spaces, such as Deep Q-learning [12].

2.1.1 Policy Gradient Methods

For our application, we focus on policy-based methods with states and actions being continuous. Here, the policy π_θ is not given by a table but by a parametric function, e.g. a neural network, with parameters θ. To improve the current policy, we can take the gradient of the expected return based on a set of collected trajectories [13], see Equation 5. The term $p(\tau|\theta)$ refers to the probability of trajectory τ when running policy π_θ.

$$\boldsymbol{\nabla}_\theta J(\theta) = \boldsymbol{\nabla}_\theta \mathbb{E}_{\tau\sim\pi_\theta}[J(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta}[\boldsymbol{\nabla}_\theta \log p(\tau|\theta) J(\tau)] \tag{5}$$

This formalism allows us to define an iterative base algorithm for policy search with policy gradients. First, we generate multiple trajectories by creating rollouts with our current policy π_θ, then we compute the policy gradient via Equation 5. Finally, we improve our policy with a gradient update using learning rate α: $\theta \leftarrow \theta + \alpha\boldsymbol{\nabla}_\theta J(\theta)$.

There are different ways to compute the return $J(\tau)$ in Equation 5. Here, we focus on utilizing an additionally learned value function, which leads to *actor-critic* methods. To express the policy gradient in terms of the value function, we can make use of the advantage function $A^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) - V^{\pi}(\boldsymbol{s})$, which captures the advantage of taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$ compared to the current policy behavior. This yields Equation 6. Using the advantage function instead of the Q-function directly has the advantage of reducing the variance of the gradient without introducing an additional bias, since it subtracts a state-dependent baseline (the value function is subtracted from the Q-function).

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0} \nabla_{\theta} \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t) A^{\pi_{\theta}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \qquad \text{where } \tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, ...) \tag{6}$$

In practice we often perform multiple small updates of the policy based on the same previous rollouts. This leads to a formulation with an old policy $\pi_{\theta_{\text{old}}}$ that was used for data collection and a new policy π_{θ} that may be different from $\pi_{\theta_{\text{old}}}$. We can account for this by adding the importance sampling term $r_t(\theta)$ into the expectation of Equation 6.

$$r_t(\theta) = \frac{\pi_{\theta}(\boldsymbol{a}_t | \boldsymbol{s}_t)}{\pi_{\theta_{\text{old}}}(\boldsymbol{a}_t | \boldsymbol{s}_t)} \tag{7}$$

Note that we also would have to add an importance sampling term for the shift in the state distribution, since the states visited by rolling out the old and new policies are not the same. But this requires knowing the state distribution of the new policy, thus we cannot directly evaluate this adapted version of Equation 6.

To avoid this issue, algorithms like Trust Region Policy Optimization (TRPO) [14] optimize a surrogate objective $\mathcal{L}^{\text{CPI}}(\theta)$ whose gradient is an approximation of the policy gradient. This is done under the assumption that the state distribution does not change too much, and thus the old and new policies are similar.

$$\mathcal{L}^{\text{CPI}}(\theta) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \pi_{\theta_{\text{old}}}} [r_t(\theta) A^{\pi_{\theta_{\text{old}}}}(\boldsymbol{s}_t, \boldsymbol{a}_t)] \tag{8}$$

Additionally, TRPO constrains the update of the new policy by restricting the Kullback-Leibler Divergence (KL) between $\pi_{\theta_{\text{old}}}$ and π_{θ}, to keep the similarity assumption of the new and old policies. Instead of directly learning the advantage function, TRPO updates a value function V_{ω} during each iteration with the dataset collected from $\pi_{\theta_{\text{old}}}$ via TD updates. The value function is represented by a neural network with parameters ω. For calculating the surrogate loss, the advantage value $A^{\pi_{\theta_{\text{old}}}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is calculated from V_{ω} and the collected transition dataset for each $\boldsymbol{s}_t, \boldsymbol{a}_t$ pair, via generalized advantage estimation.

2.1.2 Proximal Policy Optimization

Like TRPO, Proximal Policy Optimization (PPO) [15] uses a surrogate loss for the policy gradient and generalized advantage estimation. But PPO ensures that the old and new policies stay close by introducing a clipped version $\mathcal{L}^{\text{CLIP}}(\theta)$ of the surrogate loss. This makes PPO simpler in comparison to TRPO.

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \pi_{\theta_{\text{old}}}} [\min(r_t(\theta) A^{\pi_{\theta_{\text{old}}}}(\boldsymbol{s}_t, \boldsymbol{a}_t)), \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon) A^{\pi_{\theta_{\text{old}}}}(\boldsymbol{s}_t, \boldsymbol{a}_t))] \tag{9}$$

The clipping term in Equation 9 leads to the objective not improving when the importance sampling ratio moves outside the range $[1-\varepsilon, 1+\varepsilon]$. This ensures stability during training by restricting the policy update. The outer minimum operation ensures that $\mathcal{L}^{\text{CLIP}}(\theta)$ is a lower bound of $\mathcal{L}^{\text{CPI}}(\theta)$.

2.2 Variational Autoencoder

The Autoencoder (AE) is a neural network architecture that learns to encode data into a low-dimensional latent space and then decodes it back to the original data space. The learned latent representations can be used for other tasks such as classification of an encoded data sample. The VAE [6] extends the AE by making it probabilistic and transforming it into a generative model where we can sample from the latent space to generate new data samples. Here we assume that samples $\boldsymbol{x}^{(i)}$ for the dataset $\boldsymbol{X} = \{\boldsymbol{x}^{(0)}, ..., \boldsymbol{x}^{(N)}\}$ were generated from a hidden variable $\boldsymbol{z}$ via the likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. Thereby the distribution of $\boldsymbol{z}$ is given by $p_\theta(\boldsymbol{z})$. We now want to learn this generative distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. Therefore we can estimate its parameters θ by maximizing the marginal likelihood $p_\theta(\boldsymbol{x})$ for our whole dataset in Equation 10. The posterior in Equation 11 is the distribution of $\boldsymbol{z}$ given a data sample $\boldsymbol{x}$.

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z}) d\boldsymbol{z} \tag{10}$$

$$p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{x})} \tag{11}$$

Since the integral in Equation 10 and the posterior in Equation 11 are intractable, we use a Neural Network (NN) $q_\phi(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x})$ to approximate the posterior distribution in Equation 11. Here $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ corresponds to a probabilistic encoder with parameters ϕ that maps a given data sample $\boldsymbol{x}$ to a distribution over the latent values $\boldsymbol{z}$. The likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is also represented by a NN with parameters θ and corresponds to a probabilistic decoder that reconstructs a given latent value $\boldsymbol{z}$ back into the original data space. To get the parameters θ and ϕ, we maximize the log of the marginal likelihood for all samples in the dataset $\boldsymbol{X}$, see Equation 12.

$$\log p_\theta(\boldsymbol{X}) = \sum_{i \in N} \log p_\theta\big(\boldsymbol{x}^{(i)}\big) \tag{12}$$

With our approximation of the posterior, we can rewrite the log marginal likelihood of each data sample as the sum of the Evidence Lower Bound (ELBO) and an additional KL-divergence term in Equation 13. Note that here $\beta = 1$ in Equation 13. The ELBO is a lower bound of the log marginal likelihood. Since it is fully computable, in contrast to the full log marginal likelihood, we use it to train the encoder and decoder networks and omit the additional KL term.

$$\begin{aligned} \log p_\theta\big(\boldsymbol{x}^{(i)}\big) &= \mathcal{L}^{\text{ELBO}}_{\theta,\phi}\big(\boldsymbol{x}^{(i)}\big) + \underbrace{D_{\text{KL}}\big(q_\phi\big(\boldsymbol{z}|\boldsymbol{x}^{(i)}\big) \parallel p_\theta\big(\boldsymbol{z}|\boldsymbol{x}^{(i)}\big)\big)}_{\geq 0} \\ \mathcal{L}^{\text{ELBO}}_{\theta,\phi}\big(\boldsymbol{x}^{(i)}\big) &= \underbrace{E_{z\sim q_\phi(\cdot|\boldsymbol{x}^{(i)})}\big[\log p_\theta\big(\boldsymbol{x}^{(i)}|\boldsymbol{z}\big)\big]}_{-L^{\text{recon}}_{\theta,\phi}(\boldsymbol{x}^{(i)})} - \beta \underbrace{D_{\text{KL}}\big(q_\phi\big(\boldsymbol{z}|\boldsymbol{x}^{(i)}\big) \parallel p_\theta(\boldsymbol{z})\big)}_{L^{\text{KL}}_{\theta,\phi}(\boldsymbol{x}^{(i)})} \end{aligned} \tag{13}$$

The objective $\mathcal{L}^{\text{ELBO}}_{\theta,\phi}\big(\boldsymbol{x}^{(i)}\big)$ that we want to maximize can be interpreted as two conflicting tasks. The first term tries to reconstruct the original data sample as well as possible, while the second KL term tries to keep the distribution of $\boldsymbol{z}$ as close as possible to the prior. To better tune this trade-off, we can consider the hyperparameter β that weights the KL term $L^{\text{KL}}_{\theta,\phi}\big(\boldsymbol{x}^{(i)}\big)$ in Equation 13 [16, 17]. A higher β value leads to a smoother latent space and better disentanglement of the latent variables, while a lower β value leads to better reconstructions.

To calculate the gradient of the objective, the reparametrization trick [6] is applied for sampling from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Thereby we just sample ε from a standard normal distribution and then shift and scale the sampled value by the output of the encoder network. Commonly, just one $\boldsymbol{z}$ sample from

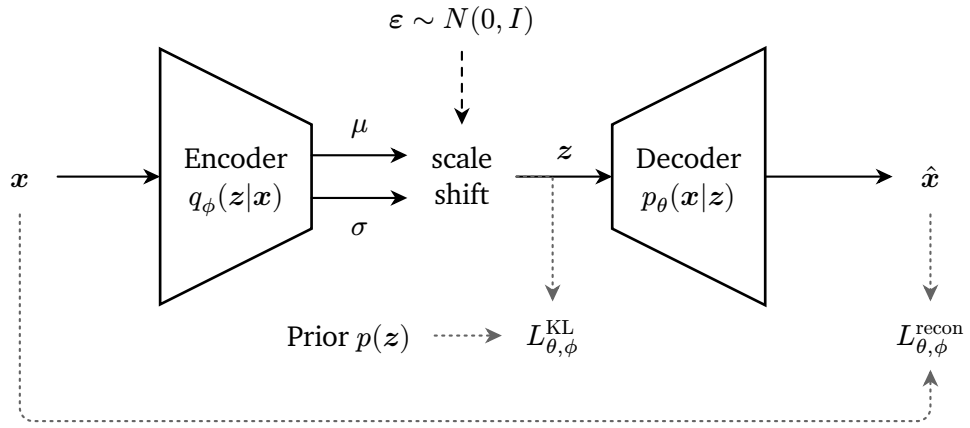


Figure 3: During training of the VAE we encode a data sample $\boldsymbol{x}$ and obtain μ and σ of $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. We then indirectly sample from this distribution and pass the sampled $\boldsymbol{z}$ through the decoder.

$q_\phi\left(\boldsymbol{z}|\boldsymbol{x}^{(i)}\right)$ per data point is used to estimate the expectation in Equation 13, which showed to be sufficient when batch size is large enough [6]. The final loss function used in training is the average of $-\mathcal{L}^{\text{ELBO}}_{\theta,\phi}$ for all samples in a batch of the full dataset. Note that we want to maximize the ELBO and thus minimize $-\mathcal{L}^{\text{ELBO}}_{\theta,\phi}$.

In practice, the prior $p_\theta(\boldsymbol{z})$ is often chosen to be a standard normal distribution $\mathcal{N}(0, \boldsymbol{I})$ with mean 0 and identity covariance matrix. Thus, the prior does not have any parameters and can be expressed as $p(\boldsymbol{z})$. The encoder outputs a Gaussian distribution with mean $\mu_{q_\phi}(\boldsymbol{x})$ and standard deviation $\sigma_{q_\phi}(\boldsymbol{x})$. Here, the KL-divergence term in Equation 13 is simplified in Equation 14.

$$L^{\text{KL}}_{\theta,\phi}\left(\boldsymbol{x}^{(i)}\right) = -\frac{1}{2}\sum_{j=1}^{D_{\boldsymbol{z}}}\left(1 + \log\left(\left(\sigma_{q_\phi}\left(\boldsymbol{x}^{(i)}\right)_j\right)^2\right) - \left(\sigma_{q_\phi}\left(\boldsymbol{x}^{(i)}\right)_j\right)^2 - \left(\mu_{q_\phi}\left(\boldsymbol{x}^{(i)}\right)_j\right)^2\right) \tag{14}$$

For continuous data ($\boldsymbol{x} \in \mathbb{R}^M$) the decoder output distribution is often chosen to be a Gaussian, where the decoder NN output determines the mean, and the covariance is set to be a fixed identity matrix. In this case, the gradient of the reconstruction loss does not depend on the fixed covariance Gaussian parameters and thus can be simplified to the mean squared error[1] $\hat{L}^{\text{recon}}_{\theta,\phi}\left(\boldsymbol{x}^{(i)}\right)$ between the data sample and the mean decoder output $\mu_{p_\theta}\left(\boldsymbol{x}^{(i)}|\boldsymbol{z}\right)$, see Equation 15. Note that the $\boldsymbol{z}$ value is computed by taking a single sample from the encoder output given $\boldsymbol{x}^{(i)}$. An overview of the VAE architecture is shown in Figure 3.

$$L^{\text{recon}}_{\theta,\phi}\left(\boldsymbol{x}^{(i)}\right) \propto \hat{L}^{\text{recon}}_{\theta,\phi}\left(\boldsymbol{x}^{(i)}\right) = \frac{1}{D}\sum_{j=1}^{D}\left(x_j^{(i)} - \mu_{p_\theta}\left(\boldsymbol{x}^{(i)}|\boldsymbol{z}\right)\right)^2 \text{ if } p_\theta(\boldsymbol{x}|\boldsymbol{z}) = N\left(\mu_{p_\theta}(\boldsymbol{x}|\boldsymbol{z}), I\right) \tag{15}$$

2.2.1 Variational Autoencoders for Trajectory Representations

To learn trajectory representations of walking motions for legged robots from a given dataset, various approaches have been explored, see Section 2.4.0.1. One way to approach this problem is by framing it as an autoregressive prediction task, where we want to learn a model that predicts possible future trajectories given the current pose or state. To capture the variety of possible future trajectory continuations, Ling et al. [7] employ a VAE where the latent space distribution

[1] In practice, we perform one update over multiple data samples in one batch, where we average over the batch, so that the loss is not affected by the batch size.

represents this variety. Their MotionVAE uses the current pose as a condition to ensure that predicted future poses, produced by the decoder, fit to the current given pose. In this conditional VAE architecture, both the encoder and decoder are provided with an additional context or condition input $\boldsymbol{c}$. Here, $\boldsymbol{c}$ consists of the current pose $\boldsymbol{p}_t$. MotionVAE then encodes and decodes the next pose $\hat{\boldsymbol{p}}_{t+1}$. After training, we can use the decoder to get possible next poses by passing it the current pose and sampling a latent $\boldsymbol{z}$ from the normally-distributed latent space prior.

Ling et al. consider a 17 minute long humanoid locomotion dataset consisting of different motion types like running, walking, or turning. The data is recorded with a frame rate of 30Hz where each frame represents one pose including link positions of all legs, feet, arms, head, and torso. The dataset is then converted into the VAE input/output format where the absolute hip pose is projected onto the ground to obtain the root frame xy-position and z-angle. The root pose is then converted to its linear and angular velocities. The link positions are represented by their relative position and orientation in this root frame. The authors highlight the importance of tuning the KL weight β of the VAE loss, see Equation 13, to balance between generalization capabilities and the reconstruction quality. The autoregressive nature of the method causes the prediction error to accumulate for multiple preceding next pose predictions. This leads to the model drifting away from the dataset distribution which yields infeasible poses. To prevent this, Ling et al. employ *scheduled sampling* where instead of providing the current pose $\boldsymbol{p}_t$ from the dataset as the model context input, the corresponding output prediction pose $\hat{\boldsymbol{p}}_t$ produced by passing $\hat{\boldsymbol{p}}_{t-1}$ or $\boldsymbol{p}_{t-1}$ through the VAE, is chosen with the probability $p_{\hat{\boldsymbol{p}}_t}$. The probability $p_{\hat{\boldsymbol{p}}_t}$ is thereby increased over time during the training process.

The learned MotionVAE can then be utilized to perform higher-level tasks, such as following a path or walking to a goal. In this scenario, we want to find a latent $\boldsymbol{z}$ for producing the next pose, given the current pose, while fulfilling the task objective. We then repeat this while using the decoder output as the new current pose. Consequently, the decoder acts as a simulator that is provided with an action, the latent $\boldsymbol{z}$, to execute. Ling et al. provide two different approaches for retrieving the latent $\boldsymbol{z}$. 1) With sampling-based control, autoregressive rollouts of 4 future steps are performed with the decoder by sampling latent values from the prior for each time step. From 200 of these rollouts the best one is chosen and the first latent value of this rollout is selected as action. Note that this approach only works for simpler tasks, like walking to a goal, and it is only able to plan 133 milliseconds into the future. 2) Alternatively, an RL policy can be trained that outputs a latent value as action. The learned policy is able to solve more complex tasks such as path following or navigating through a maze.

2.3 Variational Inference with Gaussian Mixture Models

We consider a Variational Inference (VI) problem where we want to approximate a target distribution $p(\boldsymbol{x}) = \frac{\tilde{p}(\boldsymbol{x})}{Z}$ where $\tilde{p}(\boldsymbol{x})$ can be evaluated for different values of $\boldsymbol{x}$. Here the normalization constant Z is not available. Generally this problem can be approached by optimizing a parametric distribution $q_{\boldsymbol{\theta}(\boldsymbol{x})}$ to approximate $p(\boldsymbol{x})$. Therefore, the KL between the two distributions is minimized by maximizing the ELBO. An appealing choice of $q_{\boldsymbol{\theta}(\boldsymbol{x})}$ are GMMs. GMMs are efficient to evaluate and allow approximation of arbitrary target distributions given a large enough number of components. A GMM is a sum of weighted Gaussians, see Equation 16, where the parameters $\boldsymbol{\theta} = \left(\boldsymbol{\theta}_{\boldsymbol{w}}, \boldsymbol{\theta}_{\boldsymbol{\mu}}, \boldsymbol{\theta}_{\boldsymbol{\Sigma}}\right)$ consist of the weights, mean and covariance values for all Gaussian components N_o.

$$
\begin{aligned}
q_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \sum_{o=1}^{N_o} q_{\boldsymbol{\theta}}(o) q_{\boldsymbol{\theta}}(\boldsymbol{x}|o) \\
&\text{where} \\
&q_{\boldsymbol{\theta}}(o) = \boldsymbol{\theta}_{w_o} \\
&q_{\boldsymbol{\theta}}(\boldsymbol{x}|o) = \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\theta}_{\boldsymbol{\mu}_o}, \boldsymbol{\theta}_{\boldsymbol{\Sigma}_o}\right)
\end{aligned}
\tag{16}
$$

Note, that initially we do not know the areas where $p(\boldsymbol{x})$ has a high density, which implies that some form of exploration or search is required for discovering these regions. Thus, a trade-off between exploration, i.e. searching for new high density regions, and exploitation, where we focus on improving the approximation of high density areas that were already discovered, has to be taken into account while optimizing $q_{\boldsymbol{\theta}}(\boldsymbol{x})$.

To do VI with GMMs, Arenz et al. [8] provide a general and flexible framework (GMMVI) that allows to address the exploration-exploitation trade-off via trust-region updates. GMMVI approximates the target distribution by performing multiple iterations of updating $q_{\boldsymbol{\theta}}(\boldsymbol{x})$. Arenz et al. provide many different variations of their algorithm. Here we only focus on the variant also used in [18] (SAMTRUX) with two applied changes: We only use a fixed number of Gaussian components and do not employ a sample database, which results in the "SEPTRUX" configuration. These changes are further explained in Section 3.4. To describe the proposed algorithm we first derive the ELBO objective for $q_{\boldsymbol{\theta}}$ in Equation 17. Here we can see that minimizing the KL between our approximation and the target distribution, is equivalent to maximizing the ELBO. Arenz et al. emphasize that this objective can be reinterpreted as maximizing the reward $r(\boldsymbol{x}) = \log(\tilde{p}(\boldsymbol{x}))$ under an additional entropy objective $H(\boldsymbol{\theta})$ [8, 19]. Here $H(\boldsymbol{\theta}) = -\int_{\boldsymbol{x}} q_{\boldsymbol{\theta}}(\boldsymbol{x}) \log q_{\boldsymbol{\theta}}(\boldsymbol{x}) d\boldsymbol{x}$ encourages high entropy of our approximation $q_{\boldsymbol{\theta}}$.

$$
\begin{aligned}
\arg\max_{\boldsymbol{\theta}} \; -D_{\mathrm{KL}}(q_{\boldsymbol{\theta}} \parallel p) &= \arg\max_{\boldsymbol{\theta}} \; \underbrace{-\int_{\boldsymbol{x}} q_{\boldsymbol{\theta}}(\boldsymbol{x}) \log\left(\frac{q_{\boldsymbol{\theta}}(\boldsymbol{x})}{\tilde{p}(\boldsymbol{x})}\right) d\boldsymbol{x}}_{\text{ELBO}} - \cancel{\log Z} \\
&\overset{(2)}{=} \arg\max_{\boldsymbol{\theta}} \; \underbrace{\int_{\boldsymbol{x}} q_{\boldsymbol{\theta}}(\boldsymbol{x}) r(\boldsymbol{x}) d\boldsymbol{x} + H(\boldsymbol{\theta})}_{J(\boldsymbol{\theta})}
\end{aligned}
\tag{17}
$$

[2] use $-\frac{\log a}{\log b} = \log b - \log a$

A key aspect of GMMVI are independent GMM component updates to enable efficient computation. In Equation 18, where $q_{\boldsymbol{\theta}}(\boldsymbol{x})$ is substituted by the GMM in the objective $J(\boldsymbol{\theta})$, we can already see that the component updates are independent[3] for the first integral term.

$$J(\boldsymbol{\theta}) = \sum_{o=1}^{N_o} q_{\boldsymbol{\theta}}(o) \int_{\boldsymbol{x}} q_{\boldsymbol{\theta}}(\boldsymbol{x}|o) r(\boldsymbol{x}) d\boldsymbol{x} + H(\boldsymbol{\theta}) \tag{18}$$

The entropy term $H(\boldsymbol{\theta})$ in $J(\boldsymbol{\theta})$ prevents component-independent updates since it relies on the responsibilities $q_{\boldsymbol{\theta}}(o|\boldsymbol{x})$ that represent the probability of a component belonging to a data sample $\boldsymbol{x}$, see Equation 19.

$$H(\boldsymbol{\theta}) = -\sum_{o=1}^{N_o} q_{\boldsymbol{\theta}}(o) \int_{x} q_{\boldsymbol{\theta}}(\boldsymbol{x}|o) \log\left(\frac{q_{\boldsymbol{\theta}}(o) q_{\boldsymbol{\theta}}(\boldsymbol{x}|o)}{q_{\boldsymbol{\theta}}(o|\boldsymbol{x})}\right) d\boldsymbol{x} \tag{19}$$

To address this issue, Arenz et al. derive a lower bound $\tilde{J}(\boldsymbol{\theta})$ for the objective $J(\tilde{q}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$, see Equation 20. Here an auxiliary distribution $\tilde{q}_{\boldsymbol{\theta}}(o|\boldsymbol{x})$ is introduced which approximates $q_{\boldsymbol{\theta}}(o|\boldsymbol{x})$. We can directly see that $J(\tilde{q}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ is a lower bound, since the KL term in the second row is always ≥ 0. The bound gets tighter when $\tilde{q}_{\boldsymbol{\theta}}(o|\boldsymbol{x})$ more closely matches $q_{\boldsymbol{\theta}}(o|\boldsymbol{x})$. When $\tilde{q}_{\boldsymbol{\theta}}(o|\boldsymbol{x}) = q_{\boldsymbol{\theta}}(o|\boldsymbol{x})$, the bound is tight and $\tilde{J}(\tilde{q}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ will be equal to $J(\boldsymbol{\theta})$.

$$\begin{aligned} \tilde{J}(\tilde{q}_{\boldsymbol{\theta}}, \boldsymbol{\theta}) &= \sum_{o=1}^{N_o} q_{\boldsymbol{\theta}}(o) \left[\int_{\boldsymbol{x}} q_{\boldsymbol{\theta}}(\boldsymbol{x}|o)(r(\boldsymbol{x}) + \log \tilde{q}_{\boldsymbol{\theta}}(o|\boldsymbol{x})) d\boldsymbol{x} + H(q_{\boldsymbol{\theta}}(\boldsymbol{x}|o)) \right] + H(q_{\boldsymbol{\theta}}(o)) \\ &= J(\boldsymbol{\theta}) - \int_{\boldsymbol{x}} q_{\boldsymbol{\theta}}(\boldsymbol{x}) \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\theta}}(o|\boldsymbol{x}) \parallel \tilde{q}_{\boldsymbol{\theta}}(o|\boldsymbol{x}))}_{\geq 0} d\boldsymbol{x} \end{aligned} \tag{20}$$

GMMVI employs $\tilde{J}(\tilde{q}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ to update the individual components and component weights in two separate steps. During both steps, the auxiliary distribution is set to the fixed value of the current GMM ($\tilde{q}_{\boldsymbol{\theta}}(o|\boldsymbol{x}) \leftarrow q_{\boldsymbol{\theta}}(o|\boldsymbol{x})$) First, $\tilde{J}(\tilde{q}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ is maximized w.r.t. the parameters $\boldsymbol{\theta}_{\boldsymbol{\mu}_o}, \boldsymbol{\theta}_{\boldsymbol{\Sigma}_o}$ of each component. This can be done by maximizing the expression in the square brackets in Equation 20 separately for each component, since only this term depends on the individual Gaussians $q_\theta(\boldsymbol{x}|o)$. Second, the component weights are updated by maximizing $\tilde{J}(\tilde{q}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ w.r.t $q_{\boldsymbol{\theta}}(o)$, thereby $q_\theta(\boldsymbol{x}|o)$ is considered fixed to the parameters of the previous update step. During both update steps, the trust-region update is implemented by employing the natural gradient which directly reflects how much the GMM distribution changes by a gradient update.

The main algorithm works by iteratively sampling multiple $\boldsymbol{x}$ from the current GMM and subsequently updating the GMM parameters $\boldsymbol{\theta}$ based on the $\log \tilde{p}(\boldsymbol{x})$ values of the samples. In the beginning, the GMM parameters are randomly initialized based on a given prior distribution. For the version that we consider for this work, a fixed number of samples is drawn from each component to retrieve $\boldsymbol{x}$. The following update of the GMM, shown in Algorithm 1, utilizes the density/reward value $\log \tilde{p}(\boldsymbol{x})$ of the samples. Also, the gradient of $\frac{\partial}{\partial \boldsymbol{x}} \log \tilde{p}(\boldsymbol{x})$ is provided to the update algorithm. In Algorithm 1, we first update the history of reward values for the components in the algorithm state $\boldsymbol{\nu}$ which is required for the next sub-step. Then we determine the update sizes for all the components based on the reward history. If the last update improved the reward of a component, its step size is increased, otherwise decreased (Line 2). Note that here the update step size is interpreted as the trust region size of the following update of $\boldsymbol{\theta}_{\boldsymbol{\mu}}, \boldsymbol{\theta}_{\boldsymbol{\Sigma}}$ for each component.

[3]The gradient of first term of the objective $\tilde{J}(\boldsymbol{\theta})$ can be computed independently for each component due to the outer sum operation.

To realize the trust-region update, we have to project[4] the raw gradient[5] $\frac{\partial}{\partial \boldsymbol{x}} \log \tilde{p}(\boldsymbol{x})$ to the required changes of the parameters of each Gaussian component via the natural gradient. The natural gradient is a good choice for this, since it directly incorporates how the distribution of each component changes by the gradient update. Therefore natural gradient of each component is computed by Monte-Carlo estimate over all samples. To incorporate samples from different components, importance sampling is applied. The Hessian $\frac{\partial^2}{\partial \boldsymbol{x} \partial \boldsymbol{x}} \log \tilde{p}(\boldsymbol{x})$ required for this computation is thereby approximated only via the values of $\boldsymbol{\theta}$ and $\frac{\partial}{\partial \boldsymbol{x}} \log \tilde{p}$ [20].

Algorithm 1: GMM update

Input: current GMM params $\boldsymbol{\theta}$, samples $\boldsymbol{x}$, $\log \tilde{p}(\boldsymbol{x})$, $\frac{\partial}{\partial \boldsymbol{x}} \log \tilde{p}(\boldsymbol{x})$, algorithm state $\boldsymbol{\nu}$

1 $\boldsymbol{\nu} \leftarrow \text{updateSampleHist}(\boldsymbol{\nu}, \log \tilde{p}(\boldsymbol{x}))$

2 $\boldsymbol{\nu}_{\text{compStepsizes}} \leftarrow \text{updateComponentsStepsizes}(\boldsymbol{\nu}_{\text{compStepsizes}}, \boldsymbol{\nu})$

3 $\text{ng} \leftarrow \text{naturalGradient}(\boldsymbol{x}, \log \tilde{p}(\boldsymbol{x}), \frac{\partial}{\partial \boldsymbol{x}} \log \tilde{p}(\boldsymbol{x}), \boldsymbol{\theta})$

4 $\boldsymbol{\theta}_{\boldsymbol{\mu}}, \boldsymbol{\theta}_{\boldsymbol{\Sigma}}, \boldsymbol{\nu} \leftarrow \text{trustRegionComponentsUpdate}(\text{ng}, \boldsymbol{\theta}, \boldsymbol{\nu}_{\text{compStepsizes}}, \boldsymbol{\nu})$

5 $\boldsymbol{\theta}_{\boldsymbol{w}} \leftarrow \text{weightsUpdate}(\boldsymbol{x}, \log \tilde{p}(\boldsymbol{x}), \boldsymbol{\theta})$

return $\boldsymbol{\theta}, \boldsymbol{\nu}$

Finally, the update to the parameters of each component is performed in Line 4. Therefore, GMMVI performs a search over possible step sizes for the actual natural gradient update, such that the KL between distribution of the old and new component is below the previously computed bound $\boldsymbol{\nu}_{\text{compStepsizes}}$. This search also ensures that the new covariance matrix, which results from the update, is valid[6]. To improve efficiency, the step size search is warm-started using the search range bounds from the previous iteration, which is stored in $\boldsymbol{\nu}$. The search range bounds are updated afterward based on the actual best found step size and the previous bound values. In the final step of the iteration, in Line 5, the weights of all components are updated also based on computing the natural gradient. Here, the natural gradient is also computed via a Monte-Carlo estimate over all samples with importance sampling.

[4] More specifically, we compute the natural gradient of $\tilde{J}(\tilde{q}_{\boldsymbol{\theta}}, \boldsymbol{\theta})$ w.r.t. the component parameters.

[5] The gradient $\frac{\partial}{\partial \boldsymbol{x}} \log \tilde{p}(\boldsymbol{x})$ indicates how each sample has to move in order to improve the reward.

[6] $\boldsymbol{\theta}_{\boldsymbol{\Sigma}}$ needs to be positive definite.

2.4 Related Work

The problem of motion generation for robots based on given reference datasets has been the subject of many works in the literature. This methodology is appealing because it enables incorporation of prior knowledge, in the form of reference motions, into motion planning and robot policy learning. We can split existing works into two broader categories, kinematics- and physics-based approaches. 1) The former primarily originates from the field of computer animation and focuses on directly learning motion representation and/or generation models solely on the given data. Generally supervised learning is employed in this category. For usage in computer animations, the produced motions can be directly played back. However, to execute planned motions on a robot, generally a separate policy is required that maps target poses to actions. 2) In contrast, physics-based approaches directly require interaction with the environment, e.g. the robot or character physics simulator, to generate motions. In these methods, there is no direct separation between a motion generation model and the execution policy. Here, typically RL with some form of imitation objective in the reward function is used.

Kinematics-Based Motion Models

For kinematics-based approaches, we first look at fully supervised learning with direct incorporation of commands or labels [21–23]. For example, [21] learns a next pose prediction model for quadrupedal locomotion where a target velocity command is passed as an additional condition input. In contrast, model-then-control methods first learn a motion representation independent of task-specific conditioning. Afterward, a separate task-specific module utilizes the representations of the previously learned motion model. For example, MotionVAE [7], that we already discussed in Section 2.2.1, first learns a continuous latent representation for next pose prediction and subsequently utilizes RL to learn a task-specific upper-level policy acting on this latent space. Also, discrete latent space representations can be employed such as in [24], where an action-label conditioned next latent prediction model acts on a previously trained Vector-Quantized Variational Autoencoder (VQ-VAE).

The periodicity often found in legged locomotion can be exploited for more expressive latent representations [4, 25–29]. DeepPhase [25] encodes character motions as time sub-slices by an AE. Thereby each time sub-slice maps into a continuous temporal latent trajectory that is then fitted to a sinusoidal representation consisting of amplitude, frequency, phase per time step and offset. During training the decoder receives the temporal reconstruction of latent values obtained from the fitted sinusoidals. This way, the encoder is encouraged to align its latent values with the periodic representation by exploiting periodic motion patterns in the dataset. By learning an additional model that predicts the parameters of the sinusoidal representation for the next time step given current phase values and user commands, new motions can be generated. An extension of DeepPhase, which ensures consistent sinusoidal parameters over time via enforced latent dynamics, has been successfully applied to humanoid robots in simulation [27]. Thereby an additional execution policy is learned with RL that receives the latent sinusoidal parameters as command. This approach has also been applied to execute quadrupedal dance motions on a real robot in [28]. A different approach from Mitchel et al. [29] focuses on a VAE architecture that outputs possible future trajectories given a past trajectory. By adding a separate contact state decoder that outputs future contact predictions, the VAE learns to separate contact states in the two most dominant latent dimensions, leading to a periodic temporal structure in these latent dimensions. This allows to produce controllable walking motions by overwriting the latent value of

the most dominant latent dimension, that has the lowest variance output from the encoder, with a periodic driving signal. In their follow-up work, Gaitor [4], the authors learn an additional latent space planner that acts on a polar projection of the two most dominant latent values. Thereby the user specifies the gait speed by a polar phase angular velocity value which is integrated over time to get the phase value. The planner outputs the polar radius given the current phase value to adapt the stepping behavior to the currently given terrain. The phase and radius are transformed from the polar to the Cartesian representation, which overwrites the two main latent values. The other latent values come from encoding the past state trajectory. For execution on a real quadrupedal robot, the decoded trajectory is forwarded to a Whole Body Controller (WBC) in a MPC manner.

In some more recent works, diffusion models are employed instead of AE-based architectures. For example in [3], a diffusion model is trained that predicts a future kinematic trajectory given past states and text-based instructions. The first predictions are then executed with a separately learned RL policy in a closed-loop setup.

Note that our work also falls into this category since we use a kinematics-based planner that acts on a representation model that was solely trained on a reference dataset. Like [3, 4] we also employ a separately learned RL policy to execute planned motions.

Physics-Based Imitation Policies

One of the most elementary ways to learn an imitation policy end-to-end, is to incorporate a reward term for following the joint values of the current reference motion time step. For example, DeepMimic [30] allows learning imitation policies that simultaneously follow additional tasks. Here an imitation reward, which encourages following the reference pose and joint values, is combined with additional task rewards, such as following a target velocity. DeepMimic is also applied to real quadrupedal robots in [31]. Similar, [5] learns imitation policies for a bipedal robot for different motion types, such as running and walking, while simultaneously following other task objectives. Here, the policy receives the next target imitation joint positions for a future horizon directly as input. More complex imitation rewards such as Adversarial Motion Priors [32, 33] can be employed to focus on just imitating the style of the reference and not the exact motions. Thereby, during learning, the policy tries to fool a discriminator, that itself tries to differentiate between state transitions coming from the policy and transitions coming from the reference dataset.

Similar to the kinematics-based approach, we can also utilize latent representation models for learning polices. For example, [1], distills a VAE-based policy from previously learned imitation policies. The VAE is conditioned on the current state and encodes the next imitation target pose to a latent value. The decoder maps this latent value and given state condition to an action. This allows to subsequently replace the encoder with a higher-level task-specific policy. [2] takes a very similar approach, except it uses a discrete latent space via a VQ-VAE instead of a continuous one. [2] also focuses on directly learning the VAE-based policy instead of distillation. [34] can be seen as a hybrid approach, where a kinematic VAE is initially learned to represent reference motion poses. However, only the encoder is employed to convert the current robot state to a latent value which is used as an input by a subsequently learned policy. Therefore, the policy is represented by an action decoder.

3 Methodology

In this chapter, we present our approach for task-adaptive imitation locomotion planning. Based on a locomotion reference dataset, we derive a planner suitable for different scenarios such as obstacle avoidance, goal reaching or command velocity following. In an initial pre-training phase, we first generate the reference trajectory locomotion dataset for a specified type of robot, see Section 3.1. To later execute planned trajectories, an RL low-level execution policy is trained based on the dataset, see Section 3.3. Our planning algorithm requires a compact representation of locomotion trajectories, thus we train a latent representation model. For this representation

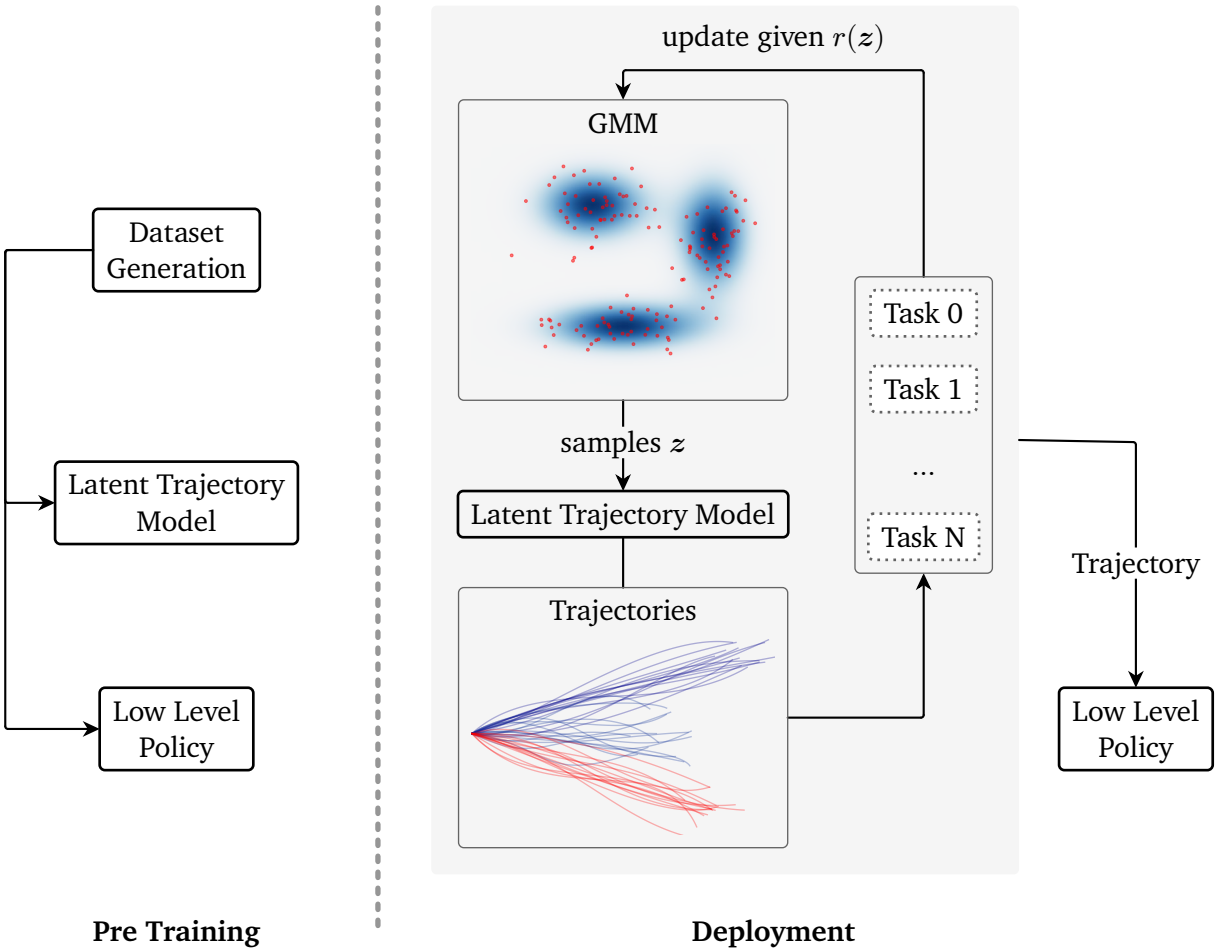


Figure 4: For our approach, we first train a latent trajectory model and a low-level execution policy based on a previously generated dataset. During deployment, we iteratively optimize a GMM to produce a reference trajectory that fulfills the given tasks. This reference is then executed with the low-level policy.

model, we provide two different versions, both based on the VAE architecture, see Section 3.2. Finally, we introduce our GMM-based planning algorithm which iteratively optimizes a distribution over latent values, see Section 3.4. This optimization process involves sampling from the current GMM distribution and decoding the latent values to their respective trajectories, using the pretrained latent trajectory model. Subsequently, given task definitions, each provides a reward value r for the decoded trajectories, which we use to update the GMM. After multiple iterations, the best sample from the current distribution is chosen and passed as decoded reference trajectory to the low-level policy. The policy then executes the reference. We provide an overview of our approach in Figure 4.

3.1 Locomotion Reference Dataset Generation

To apply our approach to a robot, we consider the quadruped Go2 from Unitree Robotics, where each of its four legs has 3 joints (hip, thigh, calf). For recording reference motions of the quadruped walking, we employ the MPC-based approach of Stark et al. [35].

Stark et al. implement a framework of 4 hierarchical stages that generate walking motions based on a given command. This command consists of the target base xy-velocity and z-height, the target base angular z-velocity and absolute xy-angles. 1) First, a *Gait Sequencer* computes a future contact sequence (including contact foot positions), target base pose and velocities for a horizon of 20 steps at 100Hz. 2) Based on a simplified Single Rigid Body Dynamics (SRBD) model, the subsequent MPC planner then computes the required contact forces based on the desired trajectory of the gait sequencer. This task is formulated as Quadratic Programming (QP) problem where the system dynamics are linearized around the current state. 3) For planned foot flight phases, the *Swing Leg Controller* computes leg swing trajectories. 4) The resulting planned foot force sequences and swing trajectories are then tracked with a WBC that itself is also formulated as QP problem.

To collect one reference trajectory, we execute the planner of Stark et al. for 15s in simulation using the Drake simulator [36]. We employ a dynamic *Gait Sequencer* version from [35] with a trot gait type where the contact timings are automatically adjusted to the base velocity. We record the absolute base position $\boldsymbol{p}_{\text{base}}$ and rotation $\boldsymbol{\varphi}_{\text{base}}$, absolute feet positions $\boldsymbol{p}_{\text{feet}}$, joint values $\boldsymbol{q}$ and foot contact states $\boldsymbol{c}$. For capturing different motion types and the transitions between them, e.g. switching from a right to a left turn, we randomly vary the planner input command multiple times during the recording of a reference. Therefore, the target base xy-velocity, base angular z-velocity and base z-height are sampled uniformly from a predefined range. To ensure smooth transitions, we linearly interpolate between the new and the previous command. To randomize the time of new commands, at each time step of the command loop[7], a new command is sampled with the probability p_{newcmd} and the old command is kept with the probability $1 - p_{\text{newcmd}}$. During a waiting phase after applying a new command, no new command is sampled to avoid abrupt changes. All the parameters used for the generation are listed in Table A.1. In total, we record 1000 trajectories.

For matching the control frequency of our low-level policy, we resample the recorded trajectories at 50Hz. We additionally filter out trajectories where the base of the robot makes an abrupt falling motion due to non-optimal foot positions produced by the planner. This yields a final number of 906 trajectories in the dataset.

[7]The command loop runs at 10Hz.

3.2 Gait Trajectory Representation Learning

Given the reference datasets obtained in Section 3.1, we can learn general representations for gait trajectories. These representations are not task-specific and therefore allow for potential usage across different tasks. To learn the gait representations, we focus on autoencoder-based approaches that represent trajectory segments via latent values $\boldsymbol{z}$. We use two different methods to achieve this. The former, Latent Spline VAE (Section 3.2.3), focuses on capturing the changes of a trajectory over time via latent space trajectories. It offers a compact representation for trajectories. The latter, Autoregressive VAE (Section 3.2.2), predicts possible completions of a trajectory given a current state.

3.2.1 Relative Trajectory Representation

Before training our trajectory models, we convert the dataset to the correct input/output format used by the VAEs. Similar to MotionVAE [7], we represent the base-frame of the robot by its linear xy ($\dot{\boldsymbol{p}}_{\text{baserel},xy}$) velocities and angular z-velocity ($\dot{\boldsymbol{\varphi}}_{\text{base},z}$). The linear velocities refer to the planar movement of the base-frame while ($\dot{\boldsymbol{\varphi}}_{\text{base},z}$) encodes the rotation around the vertical z-axis. Thereby $\dot{\boldsymbol{p}}_{\text{baserel},xy}$ is expressed in the z-rotated base frame where its z-axis is aligned with the global z-axis. Additionally, $\boldsymbol{p}_{\text{base},z}$ and $\boldsymbol{\varphi}_{\text{base},xy}$ describe the absolute z-height and absolute xy Euler angles of the base. Note that here $\dot{\boldsymbol{\varphi}}_{\text{base},z}$ is the derivative of $\boldsymbol{\varphi}_{\text{base},z}$ and thus refers to the Euler angle change rate. The foot positions are expressed relative to the base frame xy-position and z-rotation. See Equation 21 for the conversion process from an absolute foot position $\boldsymbol{p}_{\text{foot},i}$ to the relative one $\boldsymbol{p}_{\text{footrel},i}$. Here $\boldsymbol{R}_z\big(\boldsymbol{\varphi}_{\text{base},z}\big)$ is the rotation matrix that only rotates around the z-axis from the base frame to the global frame. Summing up, our final relative reference trajectory representation for one time step consists of $\boldsymbol{o} = \big(\dot{\boldsymbol{p}}_{\text{baserel},xy}, \boldsymbol{p}_{\text{base},z}, \boldsymbol{\varphi}_{\text{base},xy}, \dot{\boldsymbol{\varphi}}_{\text{base},z}, \boldsymbol{p}_{\text{footrel}}, \boldsymbol{c}\big)$ where $\boldsymbol{c}$ additionally refers to the contact state of each foot.

$$\boldsymbol{p}_{\text{footrel},i} = \boldsymbol{R}_z^T\big(\boldsymbol{\varphi}_{\text{base},z}\big)\left(\boldsymbol{p}_{\text{foot},i} - \begin{pmatrix}\boldsymbol{p}_{\text{base},xy}\\ 0\end{pmatrix}\right) \tag{21}$$

In general, when we want to reconstruct the original absolute trajectory format $\boldsymbol{o}_{\text{abs}}$ from the relative representation model output, the previously described conversion can be inverted. Therefore, an additional start base xy-position $\boldsymbol{p}_{\text{base},xy}$ and start base z-rotation $\boldsymbol{\varphi}_{\text{base},z}$ have to be provided for the required integration over time of the xy-position and z-angle.

3.2.2 Autoregressive VAE

We base our autoregressive latent trajectory model on the work of Ling et al. [7], discussed in Section 2.2.1, where a conditional VAE encodes possible next states into the latent space given a previous state as context. In contrast to Ling et al., we not only represent the next state, but the next $N = 20$ states with a single latent value, which corresponds to a duration of 0.4s with our quadruped dataset sampled at 50Hz. This is necessary to allow for planning of longer trajectories. To improve consistency with the given previous trajectory state, we extend the context length l_{ctx} to two time steps. Furthermore, for the context, the VAE trajectory format $\boldsymbol{o}$ is extended with additional velocity terms for the base z-height $\dot{\boldsymbol{p}}_{\text{base},z}$ and foot link positions $\dot{\boldsymbol{p}}_{\text{footrel}}$ yielding $\bar{\boldsymbol{o}} = \big(\boldsymbol{o}, \dot{\boldsymbol{p}}_{\text{base},z}, \dot{\boldsymbol{p}}_{\text{footrel}}\big)$. We can convert from $\boldsymbol{o}$ to $\bar{\boldsymbol{o}}$ by calculating the additional velocities with finite differences. For the NN architecture of the encoder we choose a Multilayer Perceptron (MLP) with

4 hidden layers, where the size of the hidden neurons decreases from the input to the output. The decoder is mainly a mirrored version of the encoder. For more details on the layer structure, see Figure A.1.

It is crucial to push the VAE into using the given context instead of just encoding the whole trajectory into $\boldsymbol{z}$, while still ensuring high variability of the decoder output. This is achieved in two ways. First, we pass the context $\boldsymbol{c}_{\text{ctx}}$ not only to the first, but all hidden layers except the last one, which is similar to [7]. Second, the final decoder output is parameterized as a delta value relative to the last context time step. This way the decoder has to take the context into account to produce a matching trajectory reconstruction. To still ensure variety of the decoded output, $\boldsymbol{z}$ is passed to all hidden layers and the final output layer.

Training

For the training process, we first apply z-score normalization to the dataset as in [7], which converts the values in the range $[\mu - \sigma, \mu + \sigma]$ to the range of $[-1, 1]$, where μ, σ are the mean and standard deviation of a Gaussian fitted to each dimension of the dataset. The training procedure itself applies the *scheduled sampling* from [7]. Thereby, for each batch element, we take a time-slice $(t_{\text{start}}, ..., t_{\text{start}} + l_{\text{ctx}} + (N - l_{\text{ctx}}) \cdot M_{\text{rollouts}})$ from the dataset. This slice then expands to M_{rollouts} sub-slices, where each is shifted by N time steps, resulting in temporally sliding over the time-slice. Note that the last l_{ctx} time steps of the previous sub-slice overlaps with the first l_{ctx} time steps of the next sub-slice. Before each sub-slice is passed through the VAE, it is split into $\boldsymbol{o}$ and $\boldsymbol{c}_{\text{ctx}}$, see Figure 5. We sequentially process the sub-slices, which allows for randomly replacing the real context $\boldsymbol{c}_{\text{ctx}}$ from the dataset with the end of the previous decoder reconstruction output. This way the VAE is able to learn handling accumulated errors in the reconstruction output, which is relevant for the later optimization process that utilizes autoregressive reconstruction over multiple latent steps. The previous decoder output $\hat{\boldsymbol{o}}$ first needs to be converted to the context format $\hat{\boldsymbol{\hat{o}}}$ by inferring the missing velocity values, see Figure 5. For switching between the real context from the dataset and the autoregressive context from the last decoder output we uniformly sample from a Bernoulli distribution where $p_{\text{ctx,autoreg}}$ denotes the probability of using the context from the last decoder output. Throughout the training epochs, $p_{\text{ctx,autoreg}}$ follows a 3 phase schedule, like in [7], where in the first phase $p_{\text{ctx,autoreg}} = 0$. In the second phase we linearly increase $p_{\text{ctx,autoreg}}$ from 0 to 1. In the last phase $p_{\text{ctx,autoreg}}$ is constantly 1, which leads to training with full M_{rollouts}-step autoregressive predictions.

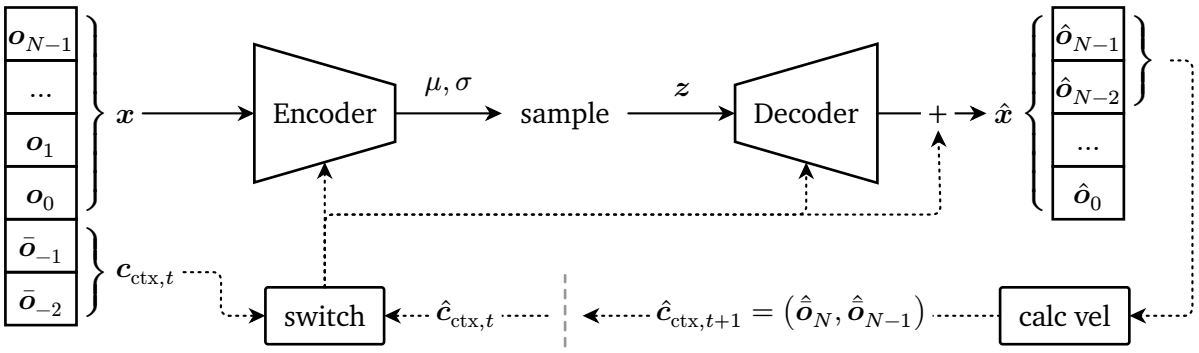


Figure 5: Training of the autoregressive conditional VAE. We randomly choose the VAE context either as the true value from the dataset or as the end of the previous decoder output.

For the loss, we employ Equation 13 where the prior is determined by a standard normal distribution. This leads to Equation 14 being used for the KL loss term. For the reconstruction loss, we use $\hat{L}^{\text{recon}}_{\theta,\phi}$ in Equation 15 for all but the contact values. The foot contact output of the decoder represents a binary distribution given one logit $\hat{\boldsymbol{c}}_{\text{lg,contact},i}$ for each foot $i \in \{1, ..., D_{\text{feet}}\}$ and time step. Note that the contact probability for the decoder output corresponds to $\hat{\boldsymbol{c}}_{\text{contact},i} = \sigma_{\text{sig}}\big(\hat{\boldsymbol{c}}_{\text{lg,contact},i}\big)$, where $\sigma_{\text{sig}}(x)$ is the sigmoid function. To calculate the reconstruction error for the contact per time step, we use the binary cross entropy loss, see Equation 22.

$$\hat{L}^{\text{recon,contact}}_{\theta,\phi}(\boldsymbol{c}, \hat{\boldsymbol{c}}) = -\frac{1}{D}\sum_{i=1}^{D_{\text{feet}}}\big[\boldsymbol{c}_i \log\big(\sigma_{\text{sig}}\big(\hat{\boldsymbol{c}}_{\text{lg,contact},i}\big)\big) + (1-\boldsymbol{c}_i)\log\big(1-\sigma_{\text{sig}}\big(\hat{\boldsymbol{c}}_{\text{lg,contact},i}\big)\big)\big] \quad (22)$$

In the final loss $\mathcal{L}_{\text{vae,autoreg}}$ that we want to minimize, we take the mean over the time steps for the reconstruction terms. The contact loss term is weighted with the hyperparameter α_{contact}. For one update, $\mathcal{L}_{\text{vae,autoreg}}$ is computed for one sub-slice for each batch element, then we take the gradient of the mean of $\mathcal{L}_{\text{vae,autoreg}}$ over all batch elements and sub-slices.

$$\mathcal{L}_{\text{vae,autoreg}} = \frac{1}{N}\sum_{t=0}^{N-1}\Big[\hat{L}^{\text{recon}}_{\theta,\phi}(\boldsymbol{o}_t, \hat{\boldsymbol{o}}_t) + \alpha_{\text{contact}} L^{\text{recon,contact}}_{\theta,\phi}(\boldsymbol{o}_t, \hat{\boldsymbol{o}}_t)\Big] + \beta L^{\text{KL}}_{\theta,\phi}(\mu_q, \sigma_q) \quad (23)$$

The model is trained by the Adam optimizer with linear decreasing learning rate over the epochs. As stated in [7], it is important to tune the VAE β hyperparameter for finding a good trade-off between decoded output variety and reconstruction quality. For the parameters used in training the model with the quadruped dataset, see Table A.2.

Inference

During deployment, we can generate longer trajectories by first sampling multiple latent values from the prior. We start with a given context and decode the first latent value. Afterward, we use the end of the last decoder output as new context for the next latent value for all remaining latent values. All resulting decoded sub-trajectories are then concatenated.

3.2.3 Latent Spline VAE

Our latent spline VAE model is motivated by other works [4, 25] that employ periodic temporal latent space trajectories for encoding reference motions. In particular, we incorporate the split decoder structure from Gaitor [4] which separately decodes poses and contact states. This encourages the model to learn a latent space that separates contact states. Our VAE maps N time steps of a sub-slice of a reference trajectory to one latent value $\boldsymbol{z}_t$ with D_z dimensions. To encode the one whole reference trajectory, we temporally slide over the reference, shifting by one time step at a time, which produces a latent space trajectory $\boldsymbol{\tau}_z = \big(\boldsymbol{z}_{t=0}, ..., \boldsymbol{z}_{t=T_z}\big)$ over T_z time steps. We want to encourage our model to learn expressive higher-level features in the latent space that smoothly change over time, such as the gait phase, current walking direction or walking velocity. At the same time, a longer latent space trajectory should be representable with a limited number of parameters to allow for trajectory optimization based on these parameters, which we show in Section 3.4. To address both of these requirements, we assume that each dimension of $\boldsymbol{\tau}_z$ is representable by splines. Splines can be defined by multiple knot points and they ensure temporal smoothness. In order to enable representability of periodic patterns in the latent space with higher frequencies without increasing the number of knot points, we parameterize two latent dimensions as polar coordinates. Thereby the spline trajectories of these two dimensions correspond to the

phase velocity and radius over time. Our spline parameterization thereby provides a reversible mapping from the encoder output latent trajectory $\boldsymbol{\tau}_z$ to spline parameters $\boldsymbol{\theta}_{\boldsymbol{\tau}_z}$.

Spline Latent Trajectory Representation

For this spline $\leftrightarrow$ latent trajectory mapping we define a B-spline basis matrix $\boldsymbol{B} \in \mathbb{R}^{T_z \times M_{\text{basis}}}$ where M_{basis} is the number of spline basis functions. Each column of $\boldsymbol{B}$ represents the basis for one spline which is a polynomial curve of degree 3 (cubic spline). Thereby each polynomial spline only has local support in the sense that it only influences a local region. Since each parameter in $\boldsymbol{\theta}_{\boldsymbol{\tau}_z}$ scales one basis spline, the same local-only influence holds for the spline parameters. The spline basis columns of $\boldsymbol{B}$ are calculated via the Cox-de Boor recursion formula where we add two additional knot points for clamped boundary conditions, see Figure 6.

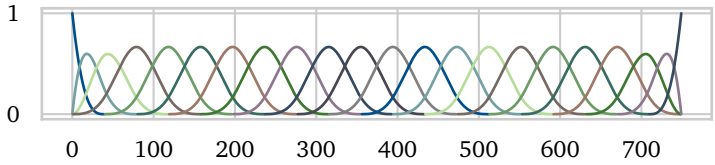


Figure 6: Spline basis functions for a 15s latent trajectory with 20 knot points (22 with boundary conditions) resulting in 22 basis functions in $\boldsymbol{B}$. The x-axis denotes the time step at 50Hz.

Given spline parameters, we can reconstruct the latent space trajectory of the non-polar dimensions ($\boldsymbol{\tau}_{z,\text{cart}} = \boldsymbol{\tau}_{z,(2,\dots,D-1)}$) via Equation 24.

$$\boldsymbol{\tau}_{z,\text{cart}} = \boldsymbol{B} \begin{pmatrix} | & & | \\ \boldsymbol{\theta}_{\boldsymbol{\tau}_z,\text{cart},2} & \cdots & \boldsymbol{\theta}_{\boldsymbol{\tau}_z,\text{cart},D-1} \\ | & & | \end{pmatrix} \tag{24}$$

The other way around, we can obtain spline parameters for a given latent trajectory by applying ridge regression. A least squares fit is performed while regularizing the squared norm of the spline parameters. Here, a higher value of the hyperparameter λ_{reg} encourages smaller spline coefficients, which leads to a higher smoothness of the fitted trajectory and improves numerical stability. At the same time, a larger λ_{reg} increases the fitting error. Solving Equation 25 for each dimension of the latent trajectory yields the spline parameters.

$$\boldsymbol{\theta}_{\boldsymbol{\tau}_z,\text{cart},i} = \left(\boldsymbol{B}^T\boldsymbol{B} + \lambda_{\text{reg}}\boldsymbol{I}\right)^{-1}\boldsymbol{B}^T\boldsymbol{\tau}_{z,\text{cart},i} \tag{25}$$

To handle the first two dimensions of the latent trajectory, which are parameterized in polar coordinates, we define one mapping from the Cartesian to the polar space $m_{c\to p} : \mathbb{R}^2 \to \mathbb{R}^2$ and one reverse mapping $m_{p\to c} : \mathbb{R}^2 \to \mathbb{R}^2$ in Equation 26. Here the unwrap operation ensures that there are no jumps between π and $-\pi$ for the phase value. Note that we parameterize the radius in log space with a predefined minimal radius r_{min}. This prevents the latent trajectory from passing through the polar center causing jumps in the phase value which would not be representable by splines.

$$\begin{aligned} \boldsymbol{\tau}_{z,\text{pol}} = m_{c\to p}\left(\boldsymbol{\tau}_{z,(0,1)}\right) &= \begin{pmatrix} \text{unwrap}\left(\arctan2\left(\boldsymbol{\tau}_{z,1}, \boldsymbol{\tau}_{z,0}\right)\right) \\ \log\left[\max\left(0, \|\boldsymbol{\tau}_{z,(0,1)}\|_2 - r_{\text{min}}\right)\right] \end{pmatrix} \\ m_{p\to c}\left(\boldsymbol{\tau}_{z,\text{pol}}\right) &= \begin{pmatrix} \sin\left(\boldsymbol{\tau}_{z,\text{pol},0}\right) \cdot \left(\exp\left(\boldsymbol{\tau}_{z,\text{pol},1}\right) + r_{\text{min}}\right) \\ \cos\left(\boldsymbol{\tau}_{z,\text{pol},0}\right) \cdot \left(\exp\left(\boldsymbol{\tau}_{z,\text{pol},1}\right) + r_{\text{min}}\right) \end{pmatrix} \end{aligned} \tag{26}$$

We found that for the trajectory optimization with our learned model it is beneficial to optimize the phase velocity instead of the phase value directly. Thus, the spline trajectory encodes the phase velocity $\dot{\varphi}_{\text{ph}}$ that we obtain by finite differences of the phase φ_{ph}. To be able to fully reconstruct the polar dimensions we additionally add the phase start value φ_{phstart} to our spline representation parameters. The final pipeline for obtaining the polar spline parameters first applies $m_{c\to p}$ to the latent trajectory. Then we calculate the phase velocity and fit the spline representation to the phase velocity and radius, analog to Equation 25.

For the reconstruction of the temporal latent trajectory $\hat{\boldsymbol{\tau}}_{z,(0,1)}$ from the spline parameters, we first integrate the spline for the phase velocity and add the phase start value to obtain the phase trajectory. Then we apply $m_{p\to c}$ to obtain the Cartesian representation of the first two latent dimensions. Figure 7 summarizes the whole spline fitting and $\hat{\boldsymbol{\tau}}_z$ reconstruction procedure.

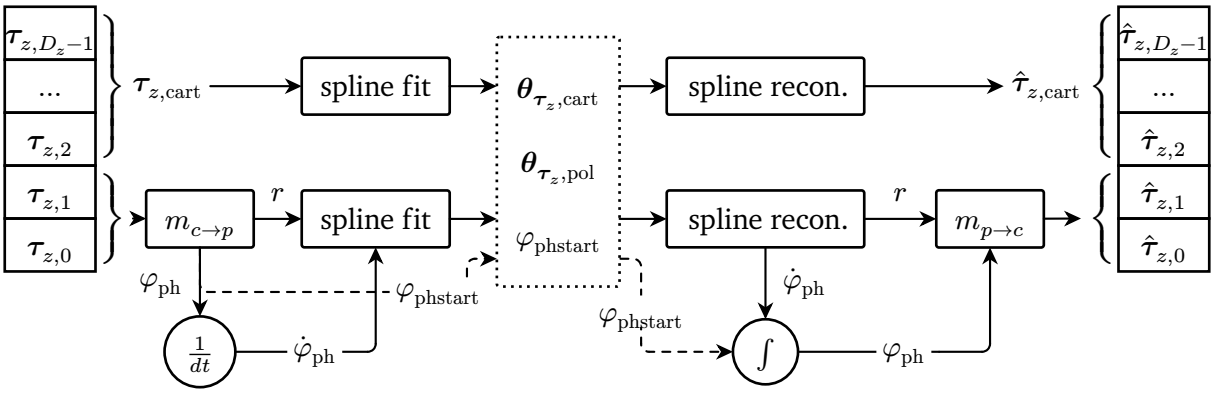


Figure 7: Fitting latent trajectory $\boldsymbol{\tau}_z$ to spline parameters (left side) and vice versa (right side).

VAE Architecture and Training

The architecture of the VAE itself is similar to our autoregressive VAE model. We use a 3 hidden layer MLP with shrinking layer sizes towards the output for the encoder. The decoder is again mainly a mirrored version of the encoder. Note that here we have two decoders, one for the contact states and one for all other trajectory entities. Both decoders have the same architecture. For the full layer structure see Figure A.2. We use a Gaussian prior with diagonal covariance matrix for the latent space prior. The main loss terms are equivalent to $\mathcal{L}_{\text{vae,autoreg}}$ from Section 3.2.2. Note that we only employ the relative trajectory representation $\boldsymbol{o}$, described in Section 3.2.1, since our VAE has no additional context input. The dataset is normalized to the range $[-1, 1]$ by min-max scaling.

It is important to align the latent space with our spline-based representation, thus we include an additional reconstruction loss utilizing the spline representation during training. The training procedure works as follows. For each element in a batch, we sample a time slice from one random reference trajectory from our dataset. The length $l_{\text{stepstrain}}$ of this slice is fixed while the start time step is also random. Now we temporally slide over this slice, encoding N time steps at a time to one mean and variance value for the latent $\boldsymbol{z}_t$. Here $\boldsymbol{z}_t$ represents the encoded time steps $(t, ..., t + N - 1)$ of the slice. Concatenating the mean latent values yields the latent trajectory $\boldsymbol{\tau}_z$ for the batch element reference trajectory slice. Subsequently, $\boldsymbol{\tau}_z$ is fitted to the parameters of our spline representation $(\boldsymbol{\theta}_{\boldsymbol{\tau}_z,\text{cart}}, \boldsymbol{\theta}_{\boldsymbol{\tau}_z,\text{pol}}, \varphi_{\text{phstart}})$, and reconstructed to the latent trajectory $\hat{\boldsymbol{\tau}}_z$ as shown in Figure 7. For the VAE reconstruction loss terms for each latent time step value, we now have two paths through the decoder. The first path stems from the normal VAE loss where we sample the latent from the encoder output distribution. The second path takes the

latent time step value from $\hat{\boldsymbol{\tau}}_z$ and passes it through the decoder. Thus, our total loss is equivalent to $\mathcal{L}_{\text{vae,autoreg}}$ with an added second reconstruction loss term for the latent values from $\hat{\boldsymbol{\tau}}_z$. This second reconstruction term is scaled by $\alpha_{\text{recon,spline}}$. Note that here we just take the sum over the dimensions of the reference trajectory and latent in the KL and reconstruction loss terms. In summary, for one update we calculate the loss value for each latent time step with the KL loss and both reconstruction loss terms. Subsequently, we take the mean over all latent time steps and batch elements to get the final loss value. All parameters used for training on the quadruped dataset are shown in Table A.3.

Inference

For inference, we can obtain a latent trajectory from encoding a given reference trajectory or by employing our spline representation. Given the latent trajectory we reconstruct it to the full reference trajectory $\hat{\boldsymbol{o}}_{\text{full}}$ by decoding each latent $\boldsymbol{z}_t$. This yields multiple overlapping decoder outputs, where each overlaps by $N-1$ time steps. The simplest way to reconstruct the full reference trajectory lies in just concatenating the first time step outputs $\hat{\boldsymbol{o}}_{z_t,0}$ of each decoded latent value. Here $\hat{\boldsymbol{o}}_{z_t,i}$ denotes the i-th time step of the decoder output for the latent $\boldsymbol{z}_t$. We can further improve the reconstructions by also utilizing the overlapping decoder outputs. Therefore we average over all previous time shifted decoder outputs that correspond to the same final time step. Each output is thereby assigned a weight, with the weight decreasing exponentially for outputs that belong to latent values further in the past. With this method, we construct an exponential filter over past decoder predictions. The calculation of each filtered reconstructed value $\hat{\boldsymbol{o}}_{\text{full},t}$ is shown in Equation 27.

$$\hat{\boldsymbol{o}}_{\text{full},t} = a_{\text{norm}} \sum_{i=0}^{N-1} (1-\alpha_{\text{filter}})^i \hat{\boldsymbol{o}}_{z_{t-i},i} \quad \text{where } a_{\text{norm}} = \frac{\alpha_{\text{filter}}}{1-(1-\alpha_{\text{filter}})^N} \tag{27}$$

3.3 Low-Level Tracking Policy Learning

To obtain a low-level policy for executing the reference trajectories from the optimization, we employ PPO described in Section 2.1.2. Thereby we consider an asymmetric actor-critic version of PPO where the value function NN receives additional privileged observations unavailable to the policy. The policy action output consists of the joint target positions. These target values are then converted to torques with a PD-controller.

The observation/state input for the policy is divided into two groups: proprioception and reference command. 1) For the former proprioception observations, the policy receives the current base-frame angular velocity, joint angles relative to the default pose, joint velocities and gravity vector. 2) The reference command contains the target base-frame angular velocity, target base-frame z-height and z-height error. Additionally, we include the relative angle between the current base-frame and the target base-frame rotation as rotation vector. For the base xy-positions we use the difference between the current and the target base position. Thereby this difference is clipped to a maximum vector length of 1. The clipping ensures that values of this observation are limited when the base is far away from the target position. For the feet, the policy receives the target positions for each foot in the base-frame, as well as the target contact state. To ensure that the policy has a future view of the reference, we compute all the previously described reference command values for the next 3 time steps. In addition to the values of the two main observation groups, the policy also gets the last applied action as observation, which allows learning smoother motions. The observations are summarized in Table 4a.

To ensure successful sim-to-real transfer, we make use of domain randomization. First, we add noise to the proprioception policy observations. Here we make use of the asymmetric actor-critic design by providing the value function with the original values of the proprioception observations without the noise, in addition to all the observations that the policy receives. Second, we randomize the physics parameters which includes, among other things, the ground friction, body masses and PD-controller gains. All these randomization parameters are sampled uniformly from a predefined range and are either directly applied or added to the corresponding default value. We provide a list of all randomized properties and their sampling ranges in Table A.5. Furthermore, we add action delay where the last action is applied to the simulation instead of the one coming from the policy with probability $p_{\text{a\_delay}}$.

For increasing the robustness of the policy, we randomly apply pushes on the base-frame following the procedure presented in [37]. Therefore, at the beginning of each episode, we sample the duration of the pushes t_{push}, the wait time between pushes and the push force magnitude $v_{\text{push,max}}$. The angle of attack is sampled for each push during the episode. During each push we first increase the force from zero to the maximum value, then we scale it back down to zero following the shape of a sinusoidal curve. The maximum push force is thereby computed via $0.5 \cdot m_{\text{base}} v_{\text{push,max}} \left(1/t_{\text{push}}\right)$, where m_{base} is the mass of the base frame. Also, the initial state of the robot is slightly randomized at the beginning of each episode. Therefore, we add random offsets to the default pose, e.g. for the base-link position, rotation, and linear and angular velocities. For the used parameters, see Table A.7.

Since we also want to deploy our optimization framework closed-loop as MPC, we need to consider possible reference trajectory optimization delays. In practice, these delays lead to a mismatch between the current robot state and the reference start when switching from an old to a new optimized reference. To account for these mismatches, we include random temporal jumps of the reference trajectories during training of the low-level policy. Therefore, we sample a

duration for the wait time until the the next jump and the number of time steps for a jump during the episode from predefined ranges, see Table A.7.

The additional privileged observations for the value function contain more entries than just the non-noisy proprioception. The value function also receives observations for the base-link linear velocities, the base-link acceleration, actuator forces, foot velocities, contact state and applied push force on the base-frame. Furthermore, we also provide information about when the next push on the base will happen and how long the feet are currently in the air to the value function. All these additional privileged observations are summarized in Table 4b.

Rewards

All reward terms can be split into two groups: imitation and regularization. The imitation terms are similar to [30] and encourage the policy to follow the current reference trajectory command. These terms are expressed as the exponential of the negative tracking error. In Equation 28 we show the reward terms for tracking the base position, velocity, rotation, and angular velocity. For the feet, we provide reward terms for tracking the feet positions (relative to the base frame) and contact states. Here $\boldsymbol{p}_t^{\text{base}}$ denotes the real current base-frame position, while entities with $\hat{}$ correspond to the current value of the reference that we want to track. For the base rotation reward, we compute the quaternion difference via $\ominus$ as the angle between the two quaternions $\boldsymbol{q}_t^{\text{baserot}}, \hat{\boldsymbol{q}}_t^{\text{baserot}}$. The feet positions for the feet tracking reward are expressed in the base-frame. To match the reference foot contact state $\hat{\boldsymbol{c}}_t$, we apply a simple penalty value for each foot where the contact state does not match.

$$
\begin{aligned}
r_t^{\text{track base pos xy}} &= \exp\left(-20 \| \boldsymbol{p}_t^{\text{base,xy}} - \hat{\boldsymbol{p}}_t^{\text{base,xy}} \|_2^2\right) \\
r_t^{\text{track base pos z}} &= \exp\left(-200 \| \boldsymbol{p}_t^{\text{base,z}} - \hat{\boldsymbol{p}}_t^{\text{base,z}} \|_2^2\right) \\
r_t^{\text{track base linvel}} &= \exp\left(-4 \| \dot{\boldsymbol{p}}_t^{\text{base}} - \hat{\dot{\boldsymbol{p}}}_t^{\text{base}} \|_2^2\right) \\
r_t^{\text{track base rot}} &= \exp\left(-15 \| \boldsymbol{q}_t^{\text{baserot}} \ominus \hat{\boldsymbol{q}}_t^{\text{baserot}} \|_2^2\right) \\
r_t^{\text{track base angvel}} &= \exp\left(-4 \| \boldsymbol{\omega}_t^{\text{base angvel,z}} - \hat{\boldsymbol{\omega}}_t^{\text{base angvel,z}} \|_2^2\right) \\
r_t^{\text{track feet pos}} &= \exp\left(-80 \| \boldsymbol{p}_t^{\text{feet in base}} - \hat{\boldsymbol{p}}_t^{\text{feet in base}} \|_2^2\right) \\
r_t^{\text{track feet contact}} &= -\sum_{k=1}^{4} |\boldsymbol{c}_{t,k} - \hat{\boldsymbol{c}}_{t,k}|
\end{aligned}
\tag{28}
$$

The second group of regularization rewards improve the general policy behavior. Here, we use similar terms as in [37]. First, we introduce the regularization rewards for the joints in Equation 29. Thereby, we encourage staying in the joint position and velocity limits where we add a margin around the real limit values of the hardware. Furthermore, the policy should stay close to the default standing joint positions. We allow more deviation in the calf joint, since it will move the most during walking. For less aggressive motions, the joint accelerations are penalized.

$$
\begin{aligned}
r_t^{\text{q limit}} &= \min\left(\boldsymbol{q}_t - 0.95 \cdot \boldsymbol{q}^{\text{lower limit}}, 0\right) - \max\left(\boldsymbol{q}_t - 0.95 \cdot \boldsymbol{q}^{\text{upper limit}}, 0\right) \\
r_t^{\text{q default}} &= \exp\left(-\sum_{k=1}^{12} \left(\boldsymbol{q}_{t,k} - \boldsymbol{q}^{\text{default}}\right)^2 w(k)\right) \text{ where } \; w(k) = \begin{cases} 0.1 \text{ if type}(k) = \text{calf} \\ 1 \text{ else} \end{cases} \\
r_t^{\text{q vel}} &= -\sum_k \max\left(|\dot{\boldsymbol{q}}_{t,k}| - 0.65 \cdot \dot{\boldsymbol{q}}_k^{\text{limit}}, 0\right) \\
r_t^{\text{q acc}} &= -\frac{1}{12} \left\|\ddot{\boldsymbol{q}}\right\|_2
\end{aligned}
\tag{29}
$$

We also penalize high joint torques, energy, and fast changes in the action to enable smoother motions, see Equation 30. When in contact, the feet should not move to avoid slipping. Additionally, high angular velocities of the base are penalized. We encourage more horizontal base orientations with the $r_t^{\text{base rot}}$ reward where $\boldsymbol{R}^{\text{rot}}\left(\boldsymbol{q}_t^{\text{baserot}}\right)$ corresponds to the rotation matrix from the base to the global frame.

$$
\begin{aligned}
r_t^{\text{torque}} &= -\left\|\boldsymbol{\tau}_t\right\|_2 - \left\|\boldsymbol{\tau}_t\right\|_1 \\
r_t^{\text{action rate}} &= -\left\|\boldsymbol{a}_t - \boldsymbol{a}_{t-1}\right\|_2 \\
r_t^{\text{energy}} &= -\sum_k |\dot{\boldsymbol{q}}_{t,k}| \cdot |\boldsymbol{\tau}_{t-1,k}| \\
r_t^{\text{feet slip}} &= -\sum_{n=1}^{4} \|\dot{\boldsymbol{p}}_{t,n}^{\text{feet global}}\|_2^2 \cdot \boldsymbol{c}_{t,n} \\
r_t^{\text{base angvel}} &= -\|\boldsymbol{\omega}_t^{\text{base angvel}}\|_2^2 \\
r_t^{\text{base rot}} &= -\left\|\left[\boldsymbol{R}^{\text{rot}}\left(\boldsymbol{q}_t^{\text{baserot}}\right)(0,0,1)^T\right]_{\text{xy}}\right\|_2^2 \\
r_t^{\text{terminate}} &= \{-1 \text{ if terminate else } 0 \\
r_t^{\text{survival}} &= \{0 \text{ if terminate else } 1
\end{aligned}
\tag{30}
$$

To counteract too small overall reward values caused by the regularization penalty terms, we add a survival bonus for each step where we do not terminate. When we terminate early, e.g. by falling down, the agent receives a negative penalty. The final reward for each step is computed by summing the individual weighted reward terms. We provide the weights for all reward terms in Table A.8.

Training

For the simulation, we employ mujoco [38] and implement our environment in JAX [39] for efficient training on the GPU. For the PPO implementation, we use Brax [40]. The policy and value function NN architectures are shown in Figure A.3. At the beginning of each episode we sample one reference trajectory from the dataset. Then we run the policy while providing it with the current reference command. The episode is terminated when we either reach the end of the reference or when the base-frame z-height falls below 5cm or when the z-vector of the base-frame points downwards[8] in the global frame. The parameters used for the PPO training are listed in Table A.9.

[8]This happens when the orientation of the base-link is inverted, which is the case when the robot is upside down.

3.4 GMM Reference Trajectory Optimization

Using our learned latent gait representation model, we want to search for suitable trajectories given multiple tasks, while enabling multimodality of possible solutions. The GMM-based approach, described in Section 2.3, is well suited for this purpose, since different components can capture distinct regions of the solution space. For example, for a goal reaching task where multiple paths to the goal exist, two distinct components each may represent a different path. In order to employ this VI-based approach, we frame the rewards of our tasks as density value $\log \tilde{p}(\boldsymbol{z})$ of the target distribution that we want to approximate with the GMM, see Equation 17. Thereby our samples $\boldsymbol{z}$ are parameterized in the latent space of the previously trained latent trajectory models, see Section 3.2.

Algorithm 2: GMM Latent Trajectory Optimization

Input:
initial GMM params $\boldsymbol{\theta}$, initial algorithm state $\boldsymbol{\nu}$, context $\boldsymbol{c}_{\text{ctx}}$,
reward weights and reward functions $\left(w_1, ..., w_{M_r}\right)$, $\left(r_1, ..., r_{M_r}\right)$

1 **repeat**
2 $\quad \boldsymbol{z} \leftarrow$ sampleFromComponents($\boldsymbol{\theta}$)
3 $\quad$ **for** sample $\boldsymbol{z}_i$ in samples $\boldsymbol{z}$
4 $\quad\quad \boldsymbol{o}_{\text{abs},i}, \boldsymbol{o}_i \leftarrow$ latentDecode($\boldsymbol{z}_i, \boldsymbol{c}_{\text{ctx}}$)
5 $\quad\quad \boldsymbol{R}_i \leftarrow \sum_{k=1}^{M_r} w_k r_k\left(\boldsymbol{z}_i, \boldsymbol{o}_{\text{abs},i}, \boldsymbol{o}_i\right)$ ▷ $\log \tilde{p}(\boldsymbol{z}_i)$
6 $\quad\quad \frac{\partial}{\partial \boldsymbol{z}_i}\boldsymbol{R}_i \leftarrow$ calcGradientZi($\boldsymbol{R}_i$) ▷ $\frac{\partial}{\partial \boldsymbol{z}_i} \log \tilde{p}(\boldsymbol{z}_i)$
7 $\quad \boldsymbol{\theta}, \boldsymbol{\nu} \leftarrow$ GMMUpdate($\boldsymbol{\theta}, \boldsymbol{z}, \boldsymbol{R}, \frac{\partial}{\partial \boldsymbol{z}}\boldsymbol{R}, \boldsymbol{\nu}$) ▷ Algorithm 1
8 **until** max iterations
9 $i_{\text{best}} \leftarrow \arg\max_i \boldsymbol{R}_i$
return $\boldsymbol{o}_{\text{abs},i_{\text{best}}}$

Our method is shown in Algorithm 2, where we first define a set of tasks. Each task consists of one or multiple reward functions $r_k, ..., r_{k+M_{r,\text{ task } l}}$ and corresponding weights $w_k, ..., w_{k+M_{r,\text{ task } l}}$. Here, a reward indicates how well a decoded sample satisfies a task, e.g. the distance to a goal position at a specific time step of the decoded reference trajectory. A weight indicates the priority of a reward in relation to all the other rewards. The algorithm starts with an initial GMM distribution and an initial algorithm state $\boldsymbol{\nu}$ for the GMM update, see Section 2.3. The initial GMM distribution parameters, which we pass to Algorithm 2, are obtained by sampling the mean and covariance values from a specified prior. A context $\boldsymbol{c}_{\text{ctx}}$ can be provided that is required for decoding the latent samples $\boldsymbol{z}$ depending on the trajectory representation model used. For one iteration, we first sample latent values $\boldsymbol{z}$ from the current GMM (componentwise). Then each sample is decoded by the trajectory representation model to the absolute and relative reference trajectory representations $\boldsymbol{o}_{\text{abs},i}, \boldsymbol{o}_i$ (Line 4). To get one scalar reward value for each sample, we evaluate all rewards from all tasks and sum up the weighted values (Line 5). Then the gradient of the reward of each sample $\boldsymbol{z}_i$ w.r.t. that sample is then computed (Line 6). Note that the loop in Line 3-6 is executed in parallel in our implementation. Finally, the current GMM parameters $\boldsymbol{\theta}$ are updated with the samples and evaluated rewards according to Algorithm 1. After we executed

the defined number of optimization iterations, we return the decoded reference trajectory of the final sample with the highest final summed reward.

We base our implementation for the GMM update on the work of [18] which implements the "SAMTRUX" variation of GMMVI [8]. To enable fast real-time capable computation, we implement the whole Algorithm 2 using JAX [39]. Thereby Algorithm 2 is compiled as a single JAX function call, which can run end-to-end on the GPU. This requires all shapes of used variables to be fixed. To satisfy this constraint, we make two changes to the GMMVI version used in [18]. First, we remove the sample database where a variable number of samples is reused from previous iterations. Usage of the sampling database would result in a variable number of samples to be redrawn (Line 2) and reevaluated (Line 4-6). Instead, a fixed number of samples is drawn from each component. Second, also the number of components is fixed to a defined value. This corresponds to the GMMVI "SEPTRUX" algorithm variation. Note that although Algorithm 2 is compiled as one GPU function call, reward weights can be adapted during runtime by passing them as parameters to the function. This allows for efficient tuning of reward weights for different tasks due to instantaneous feedback.

For converting from the relative trajectory representation to the absolute one $\boldsymbol{o}_{\mathrm{abs},i}$ in the decoding process in Line 4, the current robot state base xy-position $\boldsymbol{p}_{\mathrm{base},xy}$ and z-rotation $\varphi_{\mathrm{base},z}$ is used, see Section 3.2.1. These values are provided as part of the context $\boldsymbol{c}_{\mathrm{ctx}}$.

Autoregressive VAE Parameterization: For the parameterization of one sample, $\boldsymbol{z}_i$ corresponds to the concatenated latent values over a defined number of prediction steps for the autoregressive VAE, see Section 3.2.2. The first decoder step thereby receives $\boldsymbol{c}_{\mathrm{ctx}}$ as context input. Here $\boldsymbol{c}_{\mathrm{ctx}}$ is provided to Algorithm 2 as the last time steps of the current robot state history in the format described in Section 3.2.2.

Spline VAE Parameterization: For the Spline VAE, $\boldsymbol{z}_i$ contains the latent trajectory parameters $\left(\boldsymbol{\theta}_{\boldsymbol{\tau}_z,\mathrm{cart}}, \boldsymbol{\theta}_{\boldsymbol{\tau}_z,\mathrm{pol}}, \varphi_{\mathrm{phstart}}\right)$. The total number of latent time steps and spline knot points are predetermined as hyperparameters. We noticed that for convergence of Algorithm 2 with this representation model, it is crucial to use the polar trajectory log-radius and angular velocity as normalized form in $\boldsymbol{z}_i$. Thus, we first calculate the latent time step values of the log-radius and angular velocity over the whole training dataset and calculate the z-score normalization, where we map the range of the standard deviation around the mean to a range of $[-1, 1]$. During optimization, the log-radius and phase velocity values of the sample are then denormalized after reconstruction from the spline parameters in $\boldsymbol{z}_i$. For the decoder reconstruction output of the latent trajectory, we additionally apply the filtering described in Equation 27.

3.4.1 Base Task

There are some general reward terms used in all scenarios, which we consider as the base task. First, we want the sampled latent values to stay in a reasonable range, see Equation 31. Thereby, we penalize when latent values leave the range $[-z_{\max}, z_{\max}]$. Here $\tilde{\boldsymbol{z}}_i$ contains all the latent values in the Cartesian VAE latent space. For the autoregressive VAE we set $\tilde{\boldsymbol{z}}_i = \boldsymbol{z}_i$. But for the spline VAE, $\tilde{\boldsymbol{z}}_i$ contains all the Cartesian latent time steps reconstructed from the spline parameters of $\boldsymbol{z}_i$. This reward can be interpreted as a prior with an additional tolerance range.

$$r_i^{\text{z in range}} = -\frac{1}{D_{\tilde{z}_i}} \left\| \max(|\tilde{\boldsymbol{z}}_i| - z_{\max}, 0) \right\|_2^2 \tag{31}$$

Next we encourage our optimizer to produce physically feasible trajectories by adding inductive bias rewards. This is necessary since both representations can produce non-realistic trajectories where e.g. the feet slide over the ground when in contact. This is especially important for the spline VAE. First, we penalize contact values that are not clearly 1 or 0 with $r_i^{\text{indbias cont}}$ in Equation 32. Figure 8 visualizes the penalty value per time step for this reward.

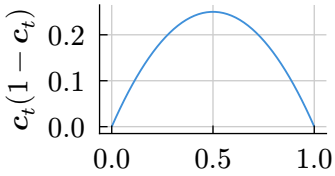


Figure 8: Inner penalty value for $r_i^{\text{indbias cont}}$ for one foot per time step. X-axis shows $\boldsymbol{c}_t$.

For all inductive bias reward terms we take the average violations over the number of reconstructed time steps N_T and number of feet D_{feet}. Each reward is also clipped by a maximum tolerance value $s_{\dots\ \text{total}}$, where the penalty becomes zero if the violation is below the tolerance value. Second, the velocity of a foot in xy direction should be zero when the foot is in contact to avoid slipping motions, see $r_i^{\text{indbias slip}}$. Here, an additional tolerance value $s_{\text{slip max step}}$ is used as a maximum allowed slipping velocity per time step. Similar, the final reward term encourages a foot to touch the ground when $\boldsymbol{c}$ indicates that it is in contact. In case of a contact, the foot z-height should match the foot radius r_{foot}. Note that this penalty assumes a flat ground plane at $\boldsymbol{p}_{\text{ground z}} = 0$. Introducing a height field function $\boldsymbol{p}_{\text{ground z}}(\boldsymbol{p}_{\text{foot xy}})$ could be used to extend this to non-flat terrain in future work. The tolerance parameter values are listed in Table A.10.

$$
\begin{aligned}
r_i^{\text{indbias cont}} &= -\max\left(\frac{1}{D_{\text{feet}} N_T}\sum_{k,t} \boldsymbol{c}_{i,k,t}\left(1-\boldsymbol{c}_{i,k,t}\right) - s_{\text{contact max total}}, 0\right) \\
r_i^{\text{indbias slip}} &= -\max\left(\frac{1}{D_{\text{feet}} N_T}\sum_{k,t} \boldsymbol{c}_{i,k,t}\max\left(\|\dot{\boldsymbol{p}}_{\text{footxy }i,k,t}\|_2 - s_{\text{slip max step}}, 0\right) - s_{\text{slip max total}}, 0\right) \\
r_i^{\text{indbias footz}} &= -\max\left(\frac{1}{D_{\text{feet}} N_T}\sum_{k,t} \boldsymbol{c}_{i,k,t}\max\left(\boldsymbol{p}_{\text{footz }i,k,t} - r_{\text{foot}} - s_{\text{footz max step}}, 0\right) - s_{\text{footz max total}}, 0\right)
\end{aligned}
\tag{32}
$$

Lastly, we add two regularization rewards that encourage avoiding large linear xy-velocity and angular z-velocity values of the base-frame, see Equation 33.

$$
\begin{aligned}
r_i^{\text{reg base angular z vel}} &= \frac{1}{N_T}\sum_t \exp\left(-\left(\dot{\boldsymbol{\varphi}}_{\text{base z }i,t}\right)^2\right) \\
r_i^{\text{reg base linear xy vel}} &= \frac{1}{2N_T}\sum_t \exp\left(-\left(\dot{\boldsymbol{p}}_{\text{base x }i,t}\right)^2\right) + \exp\left(-\left(\dot{\boldsymbol{p}}_{\text{base y }i,t}\right)^2\right)
\end{aligned}
\tag{33}
$$

Spline VAE Specific Rewards

The spline VAE requires some additional reward term for matching the current state of the robot as the start of the optimized reference, since it does not support a context input for the decoder. Therefore, we first include one reward $r_i^{\text{spline start rel}}$ for matching the relative representation with the current robot state. This reward penalizes the 2-norm error of the difference between the start state of the sample and the given start state for multiple entities. This includes the base-frame xy-velocity, base-frame z-angular velocity, feet positions in the base-frame and foot contact states. All error terms are summed up and multiplied with -1. Analogously, the second reward

$r_i^{\text{spline start abs}}$ penalizes the sum of 2-norm errors of the base-frame rotation, base-frame absolute position and absolute feet positions in the global frame.

Note that we also tested encoding the past state history with the encoder and using the resulting latent values as fixed start state for the spline latent trajectory. But this consistently caused strong artifacts at the beginning of the reconstructed trajectory while still having a high matching error for the last state of the past state history. Thus, we discard this approach of matching the start state and use the reward-based matching instead.

Additionally, we add a regularization reward to ensure that the values for the phase velocity stay in a reasonable range, see Equation 34. The allowed phase velocity range is determined by the range $3 \cdot \sigma_{\dot{\varphi}_{\text{ph}}}$ around $\mu_{\dot{\varphi}_{\text{ph}}}$, where $\sigma_{\dot{\varphi}_{\text{ph}}}, \mu_{\dot{\varphi}_{\text{ph}}}$ come from the z-score normalization of the phase velocity values.

$$r_i^{\text{spline dphase in range}} = -\frac{1}{T_z} \left\| \max\left(|\dot{\varphi}_{\text{ph},i,0\ldots T_z} - \mu_{\dot{\varphi}_{\text{ph}}}| - 3 \cdot \sigma_{\dot{\varphi}_{\text{ph}}}, 0 \right) \right\|_2^2 \tag{34}$$

3.4.2 Goal Reaching Task

This task adds rewards for reaching optional mid and end base-frame goal positions, which all employ a Huber-loss-based penalty term, shown in Equation 35. In contrast to a squared error based loss, $e_{\text{goal}}(a)$ introduces a cutoff value δ_{lin}, which ensures only linear growth of the penalty value if the error is larger than δ_{lin}. A nice consequence of this is that the gradient of the error term is constant 1 for errors beyond δ_{lin}. We found that this improves solutions for cases when the goal is too far away to be reached in the predictive horizon. Here, e_{goal} ensures that the goal error will not dominate if the goal is unreachable.

$$e_{\text{goal}}(a) = \begin{cases} \frac{1}{2\delta_{\text{lin}}} a^2 & \text{if } |a| < \delta_{\text{lin}} \\ \left(|a| - \frac{1}{2}\delta_{\text{lin}} \right) & \text{otherwise} \end{cases} \tag{35}$$

Figure 9 illustrates the shape of $e_{\text{goal}}(a)$. We can see that in the range from 0 to δ_{lin}, the penalty value increases smoothly until it reaches a gradient of 1 at δ_{lin} and only increases linearly afterward.

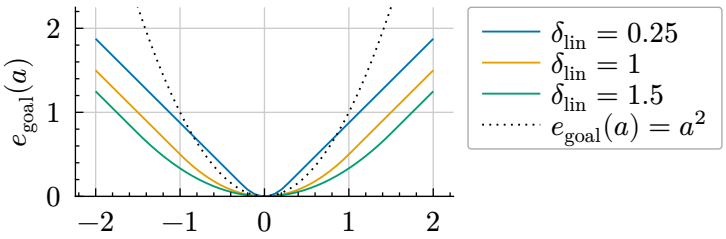


Figure 9: Huber-loss-based penalty $e_{\text{goal}}(a)$ for goal reaching for different δ_{lin} (x-axis shows a). For comparison, the black dotted line shows the squared error loss.

First, we define the mid term goal reward in Equation 36 where we take the mean penalty value from our Huber loss over the xy dimensions. Note that the goal positions just define the xy coordinates. Second, the same reward is formulated for the goal at the end of the trajectory $\boldsymbol{g}_{\text{pos end}}$. Additionally, a goal base-link z-rotation angle can be defined via $g_{\text{angle z end}}$ and included as reward $r_i^{\text{goal angle end}}$. Finally, we add a multi time step reward for the final goal to encourage minimizing the distance to the final goal as soon as possible. This can be helpful in cases where the goal position is close to the starting position. This reward $r_i^{\text{goal pos end steps}}$ penalizes the distance

to the final goal for the last $g_{\text{end steps}}$ time steps where errors closer to the end of the trajectory are weighted higher than those closer to the start. Note that in the end, we calculate $g_{\text{end steps}}$ via the fraction of the total time steps $g_{\text{end steps frac}}$. Thereby the weight increases linearly to 1. The default hyperparameters for this task are shown in Table A.10.

$$
\begin{aligned}
r_i^{\text{goal pos mid}} &= -\frac{1}{2} \sum_{d \in \{x,y\}} e_{\text{goal}}\left(\boldsymbol{p}_{\text{base},d,i,N_T \cdot 0.5} - \boldsymbol{g}_{\text{pos mid},d}\right) \\
r_i^{\text{goal pos end}} &= -\frac{1}{2} \sum_{d \in \{x,y\}} e_{\text{goal}}\left(\boldsymbol{p}_{\text{base},d,i,N_T} - \boldsymbol{g}_{\text{pos end},d}\right) \\
r_i^{\text{goal angle end}} &= -\frac{1}{2} e_{\text{goal}}\left(\boldsymbol{\varphi}_{\text{base z},i,N_T} - g_{\text{angle z end}}\right) \\
r_i^{\text{goal pos end steps}} &= -\frac{1}{2} \sum_{\tilde{t}=1}^{g_{\text{end steps}}} \frac{\tilde{t}}{g_{\text{end steps}}} \sum_{d \in \{x,y\}} e_{\text{goal}}\left(\boldsymbol{p}_{\text{base},d,i,N_T - g_{\text{end steps}} + \tilde{t} - 1} - \boldsymbol{g}_{\text{pos end},d}\right)
\end{aligned}
\tag{36}
$$

3.4.3 Obstacle Avoidance Task

With this task we consider a set of circular and a set of rectangular obstacles our robot should not collide with over the whole planned reference trajectory. Thereby the collision avoidance is formulated in the two planar xy dimensions. For each time step of the reference, we evaluate the positions of multiple points of the robot and add a penalty to the overall reward for each of the points which is in collision with an obstacle. The penalty evaluation is thereby inspired by potential field obstacle avoidance. For each obstacle we define a potential function $f(\boldsymbol{p}_{\text{xy}})$ that outputs a penalty value based on a given position. The penalty gets lower when $\boldsymbol{p}_{\text{xy}}$ moves closer to the outer boundary of the obstacle and higher otherwise. When $\boldsymbol{p}_{\text{xy}}$ is outside of the obstacle the penalty becomes zero. Note that we add a slack boundary value around each obstacle that virtually extends its size. This allows for a smooth falloff region of the penalty value around the border of the obstacle, see Figure 10.

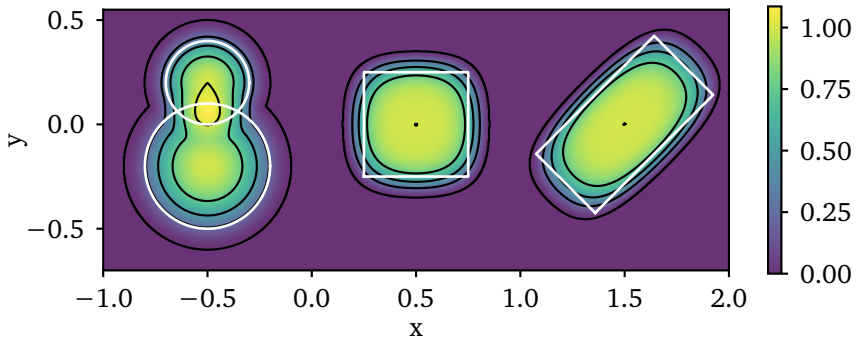


Figure 10: Obstacles potential function values $f(\boldsymbol{p}_{\text{xy}})$ summed over obstacles. On the left, there are two overlapping circular obstacles. On the right two rectangular obstacles with different size and rotations are shown. White lines denote the object boundary while Isolines are shown in black.

Circular Obstacles: Each circular obstacle is defined by its xy-position and radius $r_{\text{obs circle},n}, \boldsymbol{p}_{\text{obs circle},n}$. We define the potential function based on the squared distance to the

center, see Equation 38. Subsequently, we further smooth the distance by applying the smooth step function, defined in Equation 37. The resulting potential value is larger than zero when the position is inside the obstacle and zero when it is outside. Note that we extend the radius by a defined tolerance value $s_{\text{obs circle tol}}$ to ensure that during planning the robot does not directly touch the border of the obstacle.

$$\text{smoothstep}(a) = \begin{cases} 0 & \text{if } a \leq 0 \\ 1 & \text{if } a \geq 1 \\ 3a^2 - 2a^3 & \text{otherwise.} \end{cases} \tag{37}$$

$$f_{\text{circle},n}(\boldsymbol{p}_{\text{xy}}) = \text{smoothstep}\left(1 - \frac{\|\boldsymbol{p}_{\text{xy}} - \boldsymbol{p}_{\text{obs circle},n}\|_2^2}{\left(r_{\text{obs circle},n} + s_{\text{obs circle tol}}\right)^2}\right) \tag{38}$$

Rectangular Obstacles: A rectangular obstacle is given by its xy-position $\boldsymbol{p}_{\text{obs rect},n}$, xy-size $\boldsymbol{u}_{\text{obs rect},n}$ and z-rotation $\varphi_{\text{z obs rect},n}$. To improve smoothness of the penalty value at the rectangle corners, we employ ellipsoids which correspond to ellipsoid versions of the original rectangular shapes. First, the original position $\boldsymbol{p}_{\text{xy}}$ is transformed into the local rectangle frame ($\hat{\boldsymbol{p}}_{\text{xy},n}$) by employing the 2d rotation matrix $\boldsymbol{R}(\varphi_{\text{z obs rect},n})_z^T$ in Equation 39. We then compute a direction-dependent tolerance value $\boldsymbol{g}_{\text{strech},n}$ based on the given rectangle boundary tolerance $s_{\text{obs rect tol}}$. As with the circular obstacles, this allows to increase the obstacle size. Here $\boldsymbol{g}_{\text{strech},n}$ will be larger in the dimensions where the rectangle size is bigger (either x or y). This is needed to compensate for the ellipsoid shapes that are normally stretched along the larger size dimension, leading to overly rounded corners. Thus, $\boldsymbol{g}_{\text{strech},n}$ counteracts this. Ellipsoids can have different degrees. With a higher degree, the corners get sharper, while the ellipsoid function becomes less smooth. To combine smoothness with sharper corners we use the weighted sum of multiple ellipsoids with different degrees, see $g_{\text{ells},n}$. Thereby $s_{\text{obs rect ells num}}$ defines the number of ellipsoids while $\boldsymbol{s}_{\text{obs rect ells expos}}$ contains the values for the degrees. $\boldsymbol{s}_{\text{obs rect ells weights}}$ corresponds to the weights of the individual ellipsoids. Analog to the circular obstacles, we apply the smoothstep function to get the final penalty value $f_{\text{rect},n}(\boldsymbol{p}_{\text{xy}})$.

$$\begin{aligned} \hat{\boldsymbol{p}}_{\text{xy},n} &= \boldsymbol{R}(\varphi_{\text{z obs rect},n})_z^T (\boldsymbol{p}_{\text{xy}} - \boldsymbol{p}_{\text{obs rect},n}) \\ \boldsymbol{g}_{\text{strech},n} &= 2 s_{\text{obs rect tol}} \left(\begin{pmatrix} 0.25 \\ 0.25 \end{pmatrix} + 0.5 \frac{\boldsymbol{u}_{\text{obs rect},n}}{\sum_{d \in \{x,y\}} \boldsymbol{u}_{\text{obs rect},n,d}} \right) \\ g_{\text{ells},n} &= \sum_{k}^{s_{\text{obs rect ells num}}} \sum_{d \in \{x,y\}} \left[\frac{2\hat{\boldsymbol{p}}_{\text{xy},n}}{\left(\boldsymbol{u}_{\text{obs rect},n} + \boldsymbol{g}_{\text{strech},n}\right)^{\boldsymbol{s}_{\text{obs rect ells expos},k}}} \right]_d \frac{\boldsymbol{s}_{\text{obs rect ells weights},k}}{\sum_l \boldsymbol{s}_{\text{obs rect ells weights},l}} \\ f_{\text{rect},n}(\boldsymbol{p}_{\text{xy}}) &= \text{smoothstep}(1 - g_{\text{ells},n}) \end{aligned} \tag{39}$$

Rewards: The final reward value for the whole trajectory is given by the mean over all time steps, where at each time step we calculate the sum of all potential functions for each point in $\boldsymbol{p}_{\text{xy}}$ and then take the sum over all points, see Equation 40.

$$r^{\text{obs avoid}}(\boldsymbol{p}_{\text{xy}}) = -\frac{1}{N_T} \sum_{t}^{N_T} \sum_{k}^{N_{\boldsymbol{p}_{\text{xy}}}} \left(\sum_{n} f_{\text{rect},n}(\boldsymbol{p}_{\text{xy},k,t}) + \sum_{n} f_{\text{circle},n}(\boldsymbol{p}_{\text{xy},k,t}) \right) \tag{40}$$

For the positions per time step, we split the obstacle avoidance reward into three groups. First the reward term $r_i^{\text{obs avoid base}}$ only considers the base-link position. Second, $r_i^{\text{obs avoid feet}}$ includes all

feet positions. Last, $r_i^{\text{obs avoid additional}}$ considers 8 additional points in the base frame that are placed around the outer bounding box of the base. Thereby the base bounding box points $\boldsymbol{s}_{\text{col base xy rel}}$ are parameterized in the base frame and then transformed to the world frame via Equation 41. Additionally, $r_i^{\text{obs avoid additional}}$ includes one additional point per foot, which corresponds to the foot position shifted backwards in the base frame. This corresponds to a heuristic estimation of the knee xy-position which may extend outside the main base-body bounding box points. Note that we assume a fixed extension of the knee in xy for fast compilation. Again, the additional foot points are parameterized relative to the foot positions ($\boldsymbol{s}_{\text{col feet xy rel}}$) and are transformed to their absolute positions in the world frame via Equation 41. The transformed position values $\boldsymbol{s}_{\text{col base xy}}, \boldsymbol{p}_{\text{col feet xy}}$ are finally used in Equation 40 to calculate $r_i^{\text{obs avoid additional}}$.

$$\begin{aligned} \boldsymbol{p}_{\text{col base xy},k} &= \boldsymbol{p}_{\text{base xy}} + \boldsymbol{R}_z(\varphi_{\text{base},z})\boldsymbol{s}_{\text{col base xy rel},k} \\ \boldsymbol{p}_{\text{col feet xy},k} &= \boldsymbol{p}_{\text{foot xy},k} + \boldsymbol{R}_z(\varphi_{\text{base},z})\boldsymbol{s}_{\text{col feet xy rel}} \end{aligned} \tag{41}$$

The default parameter values for the obstacle avoidance with the considered quadrupedal robot, including the position values for the additional collision points, are listed in Table A.11.

3.4.4 Foot Step Obstacle Avoidance Task

For this task we consider additional obstacles that correspond to "holes" in the ground. Here we define areas where the robot is not allowed to step onto, but can still move over without making foot contact. We utilize the same formulation for the potential functions of the obstacles as in Section 3.4.3. We formulate the reward in Equation 42, where we only apply the penalty when a foot is in contact. Note that the hyperparameters for the rectangle obstacles are slightly adapted for the step obstacles, see $\boldsymbol{s}_{\text{step obs rect ells expos}}, \boldsymbol{s}_{\text{step obs rect ells weights}}, s_{\text{step obs rect tol}}$ in Table A.11.

$$r_i^{\text{step obs avoid feet}} = -\frac{1}{N_T}\sum_{t}^{N_T}\sum_{k}^{D_{\text{feet}}} \boldsymbol{c}_{i,k,t}\left(\sum_n f_{\text{rect},n}(\boldsymbol{p}_{\text{footxy},i,k,t}) + \sum_n f_{\text{circle},n}(\boldsymbol{p}_{\text{footxy},i,k,t})\right) \tag{42}$$

The tolerance values for the obstacles in Table A.11 may be adapted to specific scenarios, e.g. when step obstacles are very close together it may be required to reduce the tolerance values for good convergence.

3.4.5 Default Parameters

The default weights for all reward terms are shown in Table A.12. These values may be adapted for specific scenarios to encourage specific behaviors. For example, the mid-term goal can be deactivated, or the velocity regularization could be increased for slower motions. Task-specific default hyperparameters are listed in Table A.10 and Table A.11. For Algorithm 2 we use the default GMMVI parameters for SEPTRUX with some adaptations, shown in Table A.13.

3.4.6 Deployment

First, we can deploy the trajectory optimization as open-loop, where we take the current robot state as start condition. The resulting reference trajectory is then run open-loop with the low-level policy. Thereby the result of Algorithm 2 is converted to the policy input reference format described in Section 3.3.

Closed-Loop MPC

To incorporate real-time feedback, the trajectory optimization procedure can be deployed together with the low-level policy as closed-loop MPC. We plan a reference trajectory based on the current state, but only execute the first n time steps with the low-level policy until we replan from the new current robot state. Note that in this setup, we have to take the delay caused by the optimization into account. Therefore, while the optimization is running, we continue to execute the old reference trajectory with the low-level policy. When Algorithm 2 terminates, we do not execute the resulting reference directly from the beginning, but from a later time step based on the duration of the optimization. This is required since the low-level policy has to continue to move while the optimization is running. From experimentation, we found that this works up to an optimization delay of about $300\mathrm{ms}$. This restricts the possible predictive horizon for the MPC setup, since a longer horizon increases the optimization time. One problem is that this delay can cause a mismatch between the actual current state, at the time when the optimization is done, and the shifted start state of the resulting reference that takes the delay into account. This is especially relevant for the absolute foot positions. To counteract this issue, we add an expected start state reward for the MPC setup. Here we use an expected delay value based on the duration of the last optimization. The reward encourages staying close to the last optimized reference, which is currently executed by the low-level policy, at the expected delay time step. This way, the divergence between the current robot state and the delay shifted start state of the optimized reference will be reduced. This reward is given by $r_i^{\text{match old ref abs}}$, where we penalize deviations of the absolute foot positions, base-frame position, base-frame xy-orientation and foot contact states with the mean squared error.

4 Evaluation

In this chapter, we perform an experimental evaluation of our framework. First, we report metrics for training of the motion representation models and low-level policy. Subsequently, we measure the general performance of the trajectory optimization with both VAE variants for different planning horizon lengths in an open-loop execution setting. Next, we consider a closed-loop MPC deployment setup in simulation. Finally, we perform real-world experiments for both open- and closed-loop deployments on the quadrupedal robot Go2. All evaluations are performed across a diverse set of scenarios, each consisting of a distinct set of tasks.

4.1 Gait Trajectory Representation Learning

In this section, we evaluate our trajectory representation models, trained on our quadruped reference dataset, in terms of their reconstruction capabilities and general expressiveness.

4.1.1 Autoregressive VAE

We train our autoregressive VAE model, described in Section 3.2.2, with the quadruped trajectory dataset for 400 epochs. Thereby we only use the first 100 recorded trajectories from the dataset to match a similar total trajectory duration as in the work of Ling et al. [7] our model is based on. This yields a total duration of 25 minutes. Additionally, by limiting the dataset to this duration, we can compress 0.4s trajectories to only 16 latent dimensions allowing for real-time capable optimization, shown in Section 4.5. We tested different hyperparameter combinations and show the final values in Table A.2.

Since we later want to deploy our optimization framework on the real robot where the foot-contact sensors are unavailable, we train two versions of our autoregressive VAE model. One version which is only used in simulation where the past contact states are part of the context. And one version that can be deployed on the real robot where the past real contact state is not part of the context. Note that in the second version, the decoder still outputs the future contact state values.

Results

In Figure 11 we show the loss terms over the training epochs for our two final model versions. We can clearly see that most loss terms increase after the autoregressive context training with $p_{\text{ctx,autoreg}} > 0$ starts at epoch 60. This is due to the VAE adjusting to the new context that includes accumulated prediction errors. Here all loss terms show increased noise, since we randomly sample between the real and autoregressive context in this phase. Also the KL loss increases, since the model can only rely less on the noisy context. When $p_{\text{ctx,autoreg}}$ is constantly 1 after 160 epochs,

the model can fully adapt to the new context distribution and the KL loss goes down again. When comparing both model versions, the loss curves are almost identical, except for the KL. This is due to the fact that the model without the contact state in the context receives less information through the provided context and thus has to rely more on the latent value z.

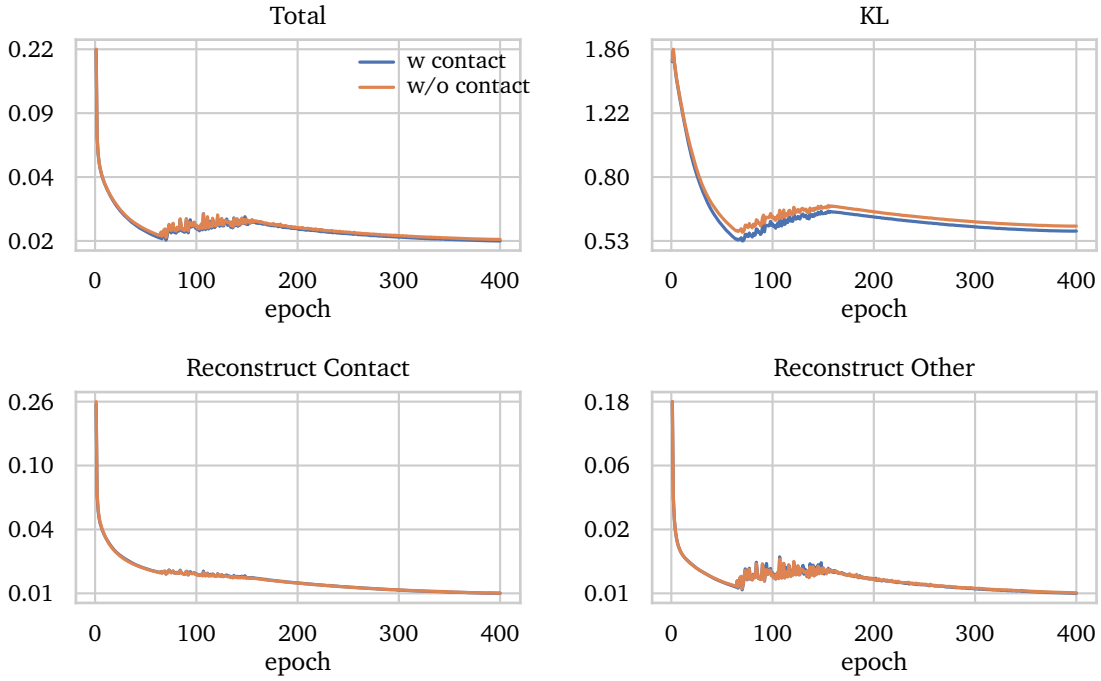
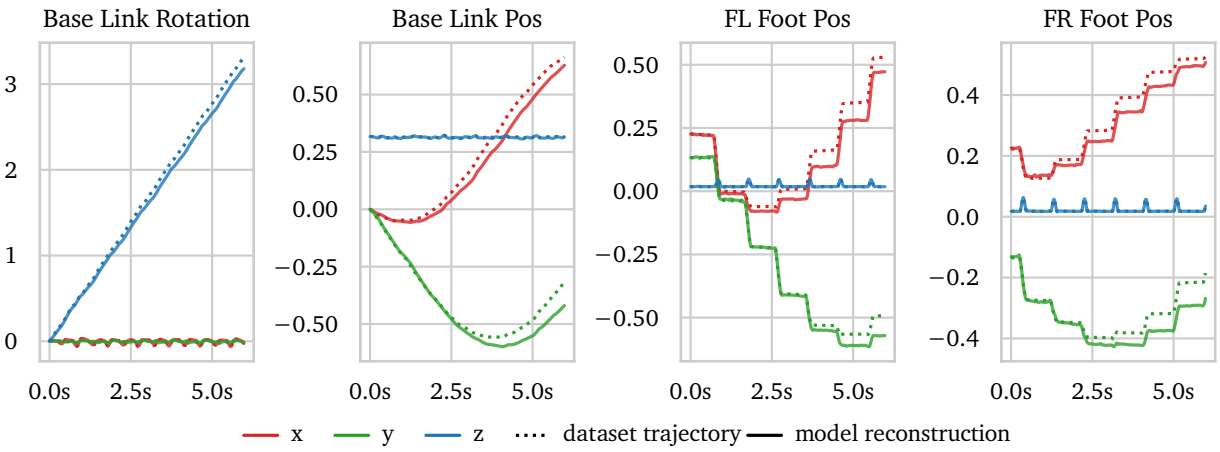


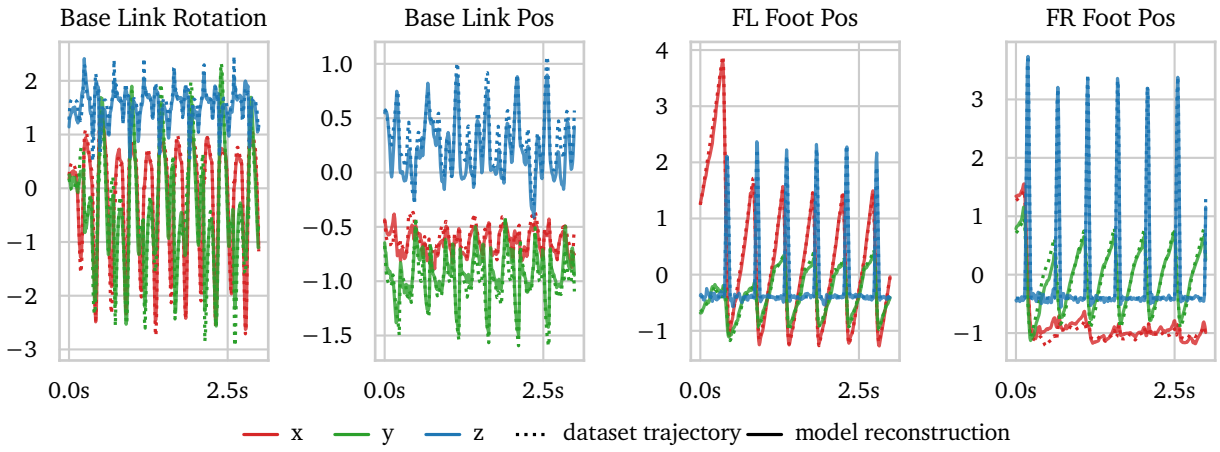
Figure 11: Training loss curves for the autoregressive VAE for the versions with and without contact in the context. The y-axis is displayed on a log scale. All loss terms except for "Total" are without scaling by their loss weight.

In Figure 12 we show quantitative results for the reconstruction capabilities of the learned VAE model. For one trajectory from the training dataset, we encode its time sub-slices of length N with the encoder to multiple z values. Then we reconstruct the full trajectory by passing each latent z through the decoder. Thereby we only use the true context from the dataset for the first reconstructed sub-slice. For the subsequent sub-slices we autoregressively feed the decoder with the end of the last reconstructed trajectory as the context. As we can see in Figure 12a, the model can reconstruct the trajectory well, especially the stepping sequence and physical feasibility[9] are maintained. However, the absolute position and rotation increasingly drift away from the true dataset trajectory over time. When we look at the relative representation output of the decoder in Figure 12a, we see that the reference is mostly well reconstructed, except for some minor deviations. The drifting in the absolute rotation and position is most likely due to accumulation of errors in the rotation and base position output of the decoder. Although the errors in the relative reconstruction output are small, they accumulate when being integrated over time, as it is the case for the base z-rotation and base xy-position. When later employing this model in our optimization framework this is not an issue, since the feedback reward terms of the optimization also include the absolute representation and thus can compensate for potential drifts in the base position or rotation.

[9]E.g. that the absolute foot position is constant when the foot is on the ground.



(a) Absolute representation. The y-axis corresponds to radians for the rotation plot and to meters for all other plots.



(b) Relative representation normalized o_{norm}. The plots correspond to the entries of the representation described in Section 3.2.1

Figure 12: Reconstruction of a dataset trajectory where the context is passed autoregressively using the model that includes the contact in the context. We omit the rear feet and contact states for clarity of the plot. Solid lines correspond to the reconstruction model output while dashed lines show the true dataset trajectory.

To assess the variability of generated trajectories, we repeat the decoding of the trajectories shown in Figure 12a, but this time randomly sampling the latent z values from the prior instead of using the values from the encoder. The absolute decoded trajectories of 75 of these random rollouts from the same start state are shown in Figure 13. We can observe high variance in the base xy-position, z-rotation and stepping times over time. Note that some of the trajectories are of lower quality, since for example a foot moves when its z-height indicates ground contact.

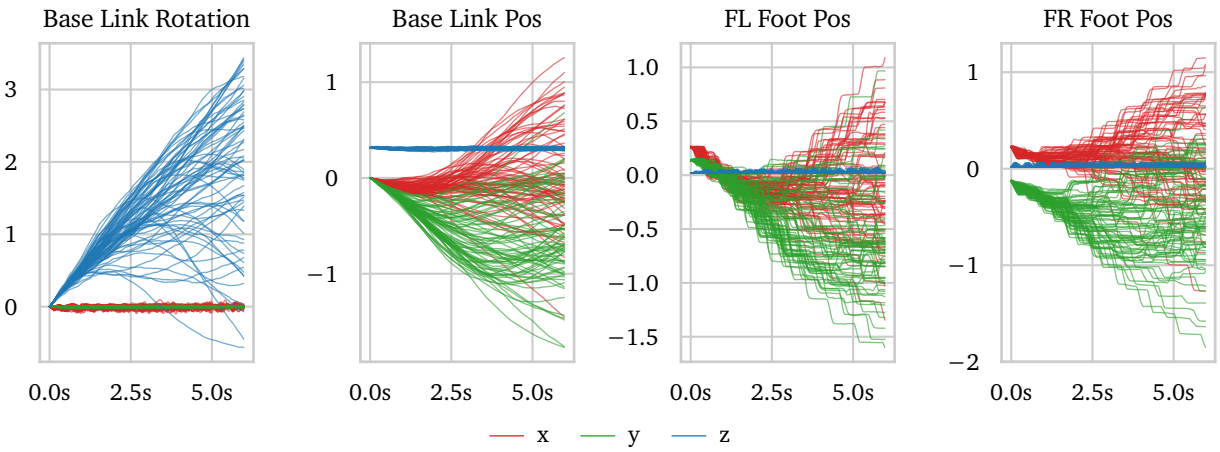


Figure 13: 75 random walk rollouts from the same start context, shown in the absolute representation. z values are sampled from the prior.

4.1.2 Latent Spline VAE

We train our latent spline VAE, described in Section 3.2.3, on the quadruped dataset from Section 3.1. We tested different hyperparameters and model sizes and show the final used values in Table A.3 and Figure A.2.

Results

Figure 14 shows the loss curves over $14 \cdot 10^3$ training iterations. We can observe for the reconstruction loss terms of scalar values ("Reconstruct Other"), that the spline path yields lower loss values than the normal VAE loss path. This is expected since the spline reconstruction path does not contain noise but only relies on the mean decoder output values. For the contact reconstructions we get the inverse behavior. When looking at the reconstructed decoder outputs in Figure 15, see similar results as for the Autoregressive VAE. The overall motions closely match the reference for the absolute values in Figure 15a, while showing some small mismatches. The same holds for the direct decoder output shown in Figure 15b. Note that here we apply the filtering described in Equation 27. The filtering generally improves the reconstruction quality. In the appendix in Figure A.4, we show a comparison between filtered and non-filtered outputs to quantify its effect.

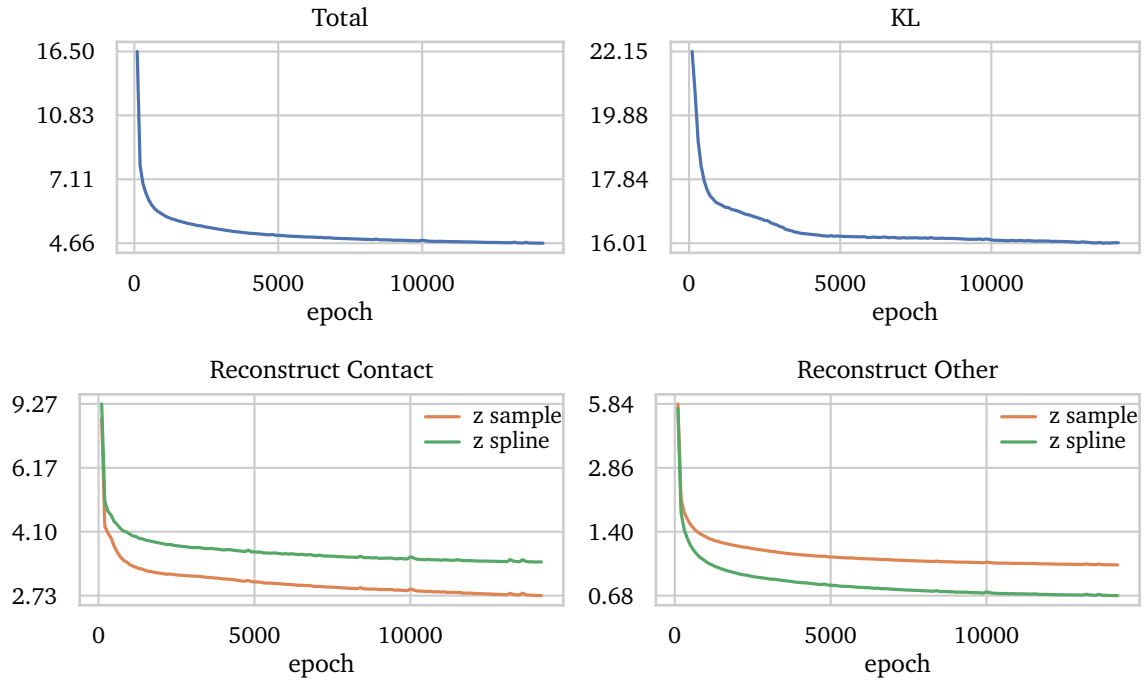
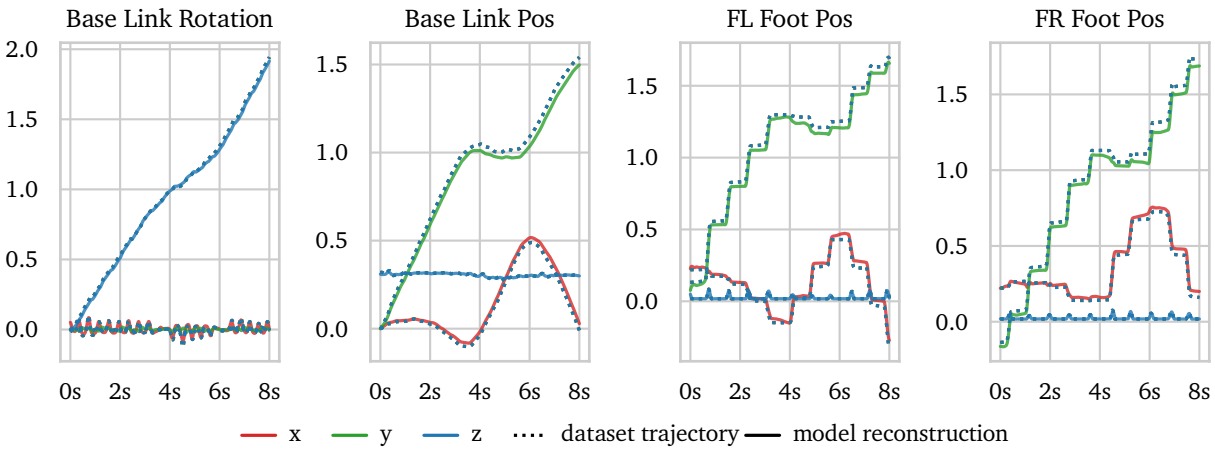
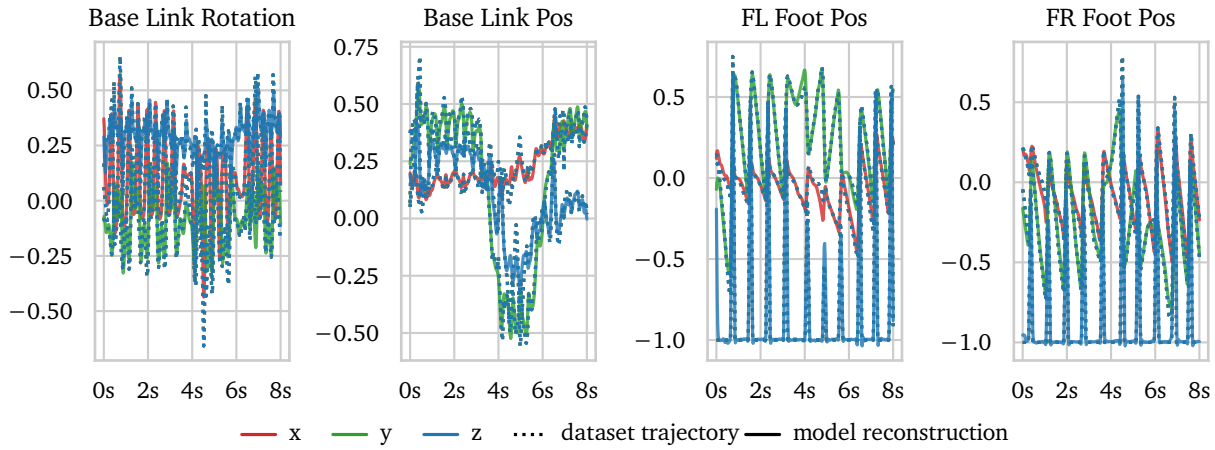


Figure 14: Training loss curves for the spline VAE. The y-axis is displayed on a log scale. All loss terms except for "Total" are without scaling by their loss weight. The normal VAE reconstruction and spline reconstruction loss values are combined in the same plot.



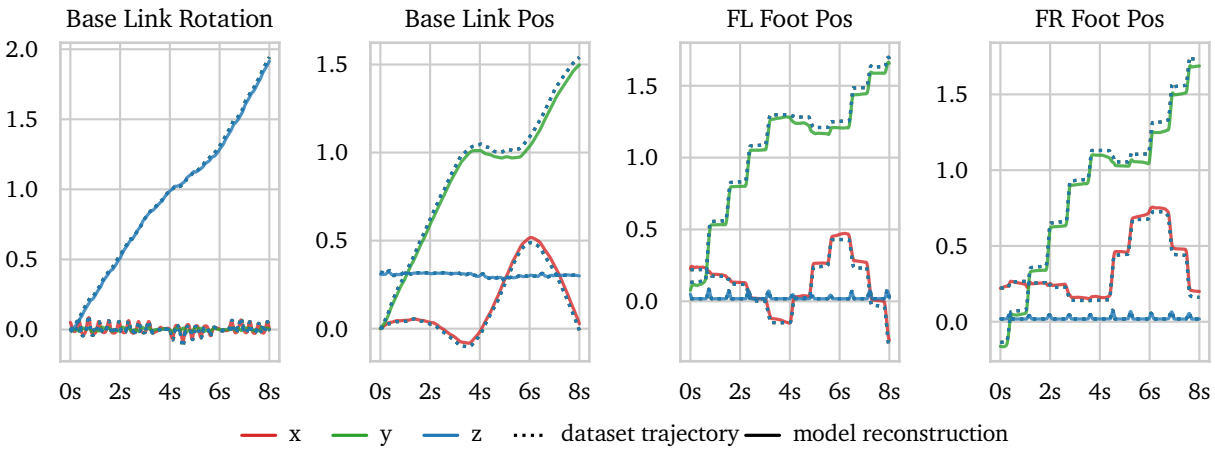
(a) Absolute representation. The y-axis corresponds to radians for the rotation plot and to meters for all other plots.



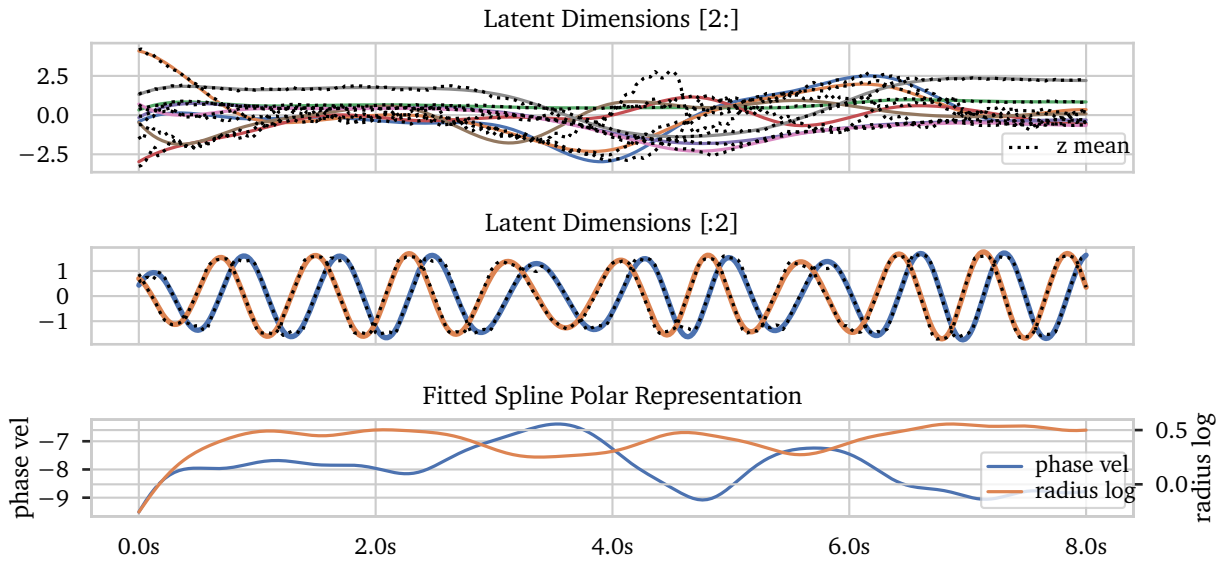
(b) Relative representation normalized. The plots correspond to the entries of the representation described in Section 3.2.1

Figure 15: Reconstruction when passing one dataset trajectory through our spline VAE with applying the filtering described in Equation 27 with $\alpha_{\text{filter}} = 0.05$. We omit the rear feet and contact states for clarity of the plot. Solid lines correspond to the reconstruction model output, while dashed lines show the true dataset trajectory.

When we consider the reconstructions using our spline latent trajectory representation in Figure 16a, we can observe that here the reconstruction quality is similar to the previously shown reconstructions through the normal VAE path. In Figure 16b we show the corresponding latent space trajectory. Here the fitted spline representation matches the decoder mean latent output well, except for some mismatches at around 4s. We can observe that the latent trajectory matches features of the reconstructed motion. For example, between 1s and 3s the motion (walking direction, step length, …) is roughly constant, see Figure 15b. Here the fitted latent trajectory also does not change much, see Figure 16b. Note that for the polar dimensions this means that the radius and phase velocity are roughly constant. In the next time frame, from 3s to 6s, the motion pattern changes and settles afterward. The latent trajectory reflects this pattern. We also examined the



(a) Absolute representation. The y-axis corresponds to radians for the rotation plot and to meters for all other plots.



(b) Latent trajectory fitted spline representation. Dotted lines show the raw encoder output latent trajectory, while the solid lines correspond to the fitted spline representation that is used as input for the decoder. The two polar dimensions are shown twice, in the middle plot in Cartesian coordinates and in the bottom plot in our polar representation.

Figure 16: Reconstruction when passing one dataset trajectory through our spline VAE where the intermediate latent trajectory is fitted to our spline representation. We also apply the filtering described in Equation 27 with $\alpha_{\text{filter}} = 0.05$.

references of the whole dataset and found that the phase velocity is always negative, where larger negative values correspond to faster stepping motions. Furthermore, we also changed individual latent dimension values of encoded trajectory slices to investigate the meaning of individual latent dimensions. Thereby we found that the most dominant latent dimensions represent quantities like heading direction, heading velocity, base angular velocity or step length.

4.2 Low-Level Tracking Policy Learning

We train our RL low-level tracking policy on the quadruped dataset with the parameters shown in Table A.9 and Table A.8. We perform multiple training runs over 5 seeds, each running for $0.8 \cdot 10^9$ environment steps.

Results

In Figure 17, we show metrics over the training iterations. We can observe that the overall return continuously increases while the tracking error decreases. For the base position, the final tracking error in the z direction is lower than in the xy-plane. This is due to the variation of the target base height being very low in comparison to the xy target movement.

Figure 18 shows a rollout for the policy from one seed. Over the shown duration of 1.4s, the policy follows the reference target well. After a disturbance push on the base, the deviation from the reference increases temporarily. But the policy recovers quickly and reduces the tracking deviation of the base-frame again after 0.4s.

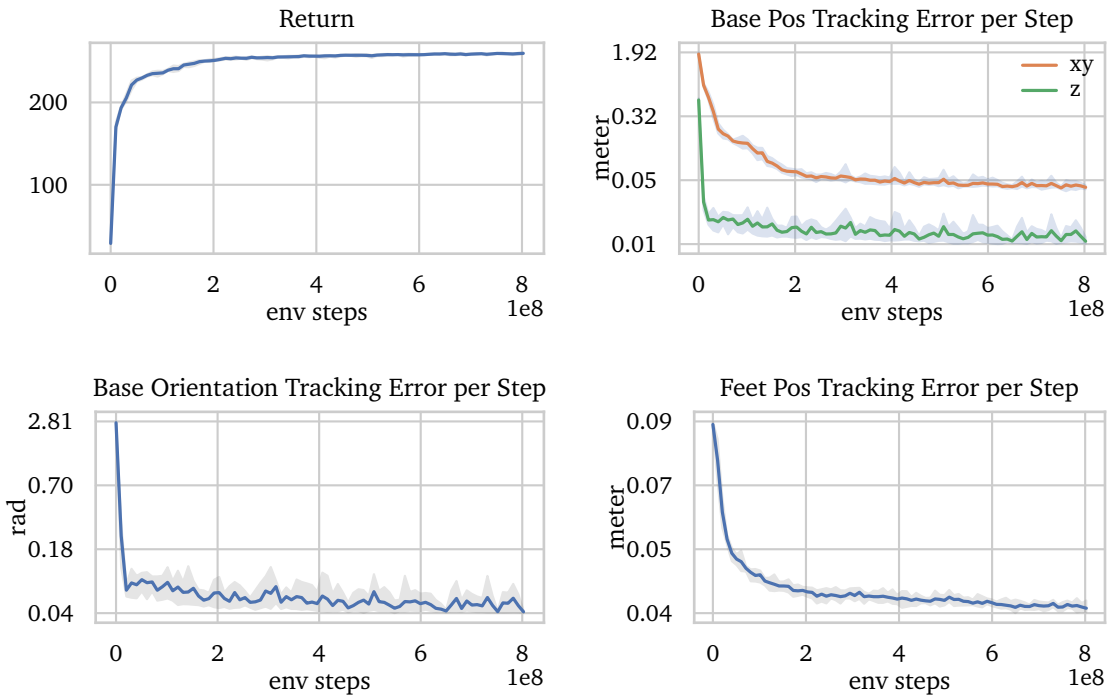


Figure 17: Tracking policy training metrics over 5 seeds, the shaded areas denote the range of min to max values over all seeds. The base and feet tracking errors are shown with a log-scaled y-axis. Note that the tracking error for the feet positions is measured in the base-frame.

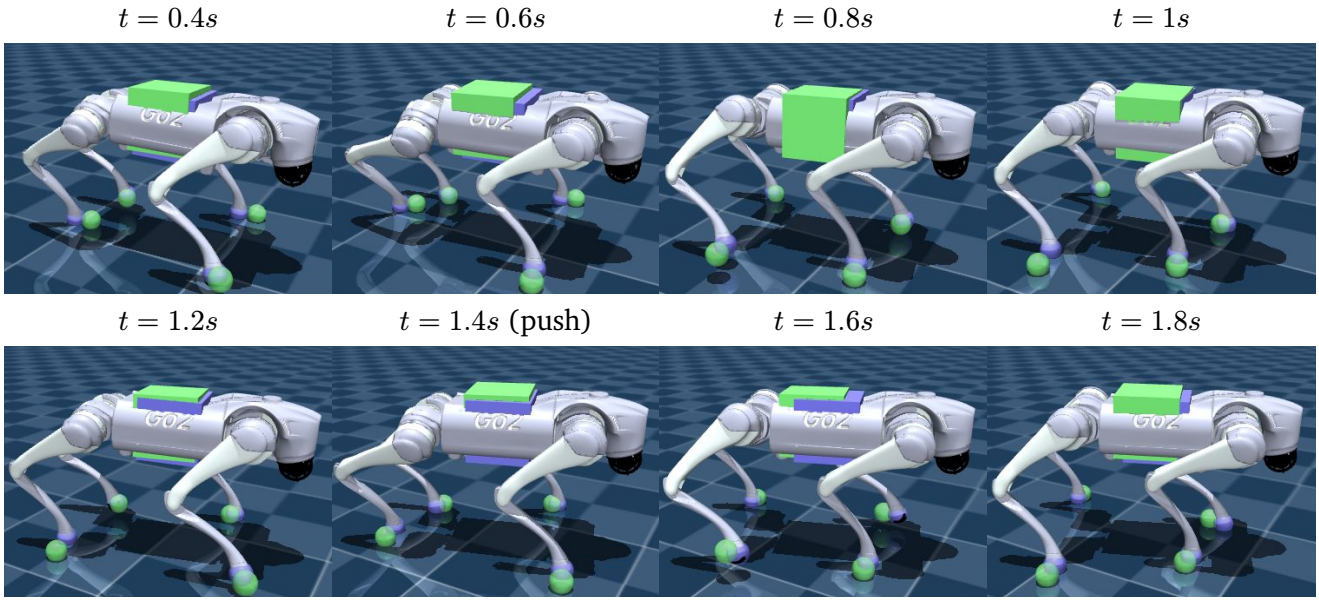


Figure 18: Tracking policy rollout following one reference from the dataset, shown in increments of $0.2s$. The purple box and spheres denote the current base and feet poses, while the green box and spheres show the target poses from the reference trajectory. The target feet positions correspond to the absolute values of the reference. A perturbation push is applied to the base-frame at around $t = 1.4s$. Note that the camera follows the base-frame.

4.3 GMM Trajectory Optimization

For our trajectory optimization algorithm, we first consider the open-loop deployment variant only in simulation to evaluate the general performance of our planning framework. Here, we want to compare both VAE variants across different scenarios and time horizon lengths. Therefore, we define 4 evaluation scenarios, where for each, we sample 12 random start states from the reference trajectory dataset used during training. Then for each start state, we randomly sample 2 different goal positions. This results in 24 different configurations per scenario. We apply our algorithm twice to each configuration with two different seeds for the GMM optimization. Both VAE versions share the same reward weights for each task, except for the VAE-specific reward terms. We employ the autoregressive VAE version with the contact state as condition input in this experiment. For each scenario and VAE type, we consider two prediction optimization time horizons of 4 and 6 seconds. For the $6\mathrm{s}$ prediction horizon, we additionally execute the planned trajectories with the low-level tracking policy. Thereby the simulation is initialized to the start state that was also provided to the trajectory optimization. The policy then tries to track the reconstructed trajectories from the best final sample. We also perform a small ablation study for the optimization parameters with the $6\mathrm{s}$ horizon for two of the 4 scenarios (*Wall Gap*, *Step Crosswalk*). Thereby we vary the number of components $(3, 5, 7)$ and samples per component $(25, 50, 75)$. In the following we describe the 4 scenarios.

Step Checkerboard: In this scenario, the robot is supposed to walk over a grid of stepping obstacles. The stepping obstacles are arranged in a checkerboard fashion, where each obstacle has a size of $15\mathrm{cm} \times 15\mathrm{cm}$. The checkerboard has a size of 8×15 cells, where each second cell is a stepping obstacle. For the start state, the xy-position is always set to a central position (at $x = 0, y = 0$) to the right of the checkerboard. The mid goal is deactivated and the end goal position is sampled in a range of $x \in U(-1.0, -2.0), y \in U(-1.5, 1.5)$. This yields optimized trajectories leading from left to right (towards the negative x direction). For a visualization see Figure 20. For this scenario, we increase the reward weight for $r^{\text{goal pos end}}$ to $1.3 \cdot 10^4$ and decrease the step obstacle tolerance to 0.05.

Step Crosswalk: Here, the robot has to walk over 6 lengthy holes on the ground (step obstacles), which are arranged as a 90 degree rotated crosswalk, see Figure 22. Each of these column obstacles on the ground has a size of $1\mathrm{m} \times 0.2\mathrm{m}$, where the spacing between each obstacle is $0.2\mathrm{m}$. As with the Step Checkerboard the xy start position is fixed to the right of the obstacles (at $x = 0, y = 0$). Only the end goal is active, where its position is again sampled in the range of $x \in U(-1.0, -2.0), y \in U(-1.5, 1.5)$. For this scenario we increase the reward weight for $r^{\text{goal pos end}}$ to $1.3 \cdot 10^4$ and the stepping obstacle reward $r^{\text{step obs avoid feet}}$ weight to $2.5 \cdot 10^4$.

Wall Gap: This scenario consists of two normal rectangle obstacles with a gap of $0.75\mathrm{m}$ between them, see Figure 24. Each obstacle has a size of $0.6\mathrm{m} \times 1.5\mathrm{m}$. The xy start position is on the right of the obstacles. It is set to a sampled value from $x \in U(-0.2, 0.2), y \in U(-0.5, 0.5)$. Again only the end goal is active, where its position is sampled in the range of $x \in U(-1.3, -2.1), y \in U(-0.5, 0.5)$.

2 Goal: We do not consider any obstacles in this scenario, but instead two goal positions. Thus, the mid and end goals are both active. The goal positions are sampled such that the total distance

from start to the mid goal and mid goal to the end goal is below 2m. At the same time, the sampling ensures that there is at least a distance of 0.2m between the goal and the mid goal and the start. For an illustration, see Figure 26.

Algorithm Configuration

We set the number of iterations for Algorithm 2 to the required value for convergence per scenario and VAE type. For the spline VAE, the number of iterations is 150 across all tasks. The autoregressive VAE generally requires more iterations, especially for scenarios with stepping obstacles. Thus, we run it for 600 iterations for the *Step Checkerboard* and *Step Crosswalk* scenarios, and for 200 iterations for the *Wall Gap* and *2 Goal* scenarios. We use the algorithm parameters shown in Table A.13, where we only consider diagonal covariance matrices for the GMM. With diagonal covariance matrices, the GMM is more efficient to optimize, which keeps the total optimization runtime within an acceptable range. We run the optimization on an NVIDIA RTX 5080 GPU.

4.3.1 Optimization Results

For each scenario we look at the sum of common reward terms. This only excludes the spline specific rewards, e.g. start state matching. We also evaluate the quality of the final best sample after the optimization. For all box plots shown in the following Figures, the box includes the range from the first to the third quartile, while the middle line corresponds to the median.

Step Checkerboard Scenario

For quantifying the performance of both VAE variants, we can first look at the common reward plot in Figure 19 (top row). We can observe that while both variants reach similar final rewards for both prediction horizons, the spline VAE needs fewer iterations for converging. But at the same time, with the spline VAE, the optimization algorithm can only perform far fewer iterations per second than with the autoregressive VAE. This is due to the spline VAE requiring to pass the latent value for each time step through the decoder. At the same time, the autoregressive VAE only needs one reconstruction pass through the decoder every 20 time steps.

For the 6s horizon, the lower required number of iterations counteracts the longer computation time per iteration for the spline VAE and both VAEs achieve a similar convergence rate over time. Note that when looking at the individual component reward distributions in Figure 19 (third row), we can see that more components of the spline VAE stay at a high reward level. For the autoregressive VAE the worst component has a significantly lower reward than all others. Here, also the rewards of the components are generally higher for the spline VAE. This indicates that, for the spline VAE, generally more components correspond to more valid solutions. At the same time, the start state mismatch error is higher for the spline VAE, see second row in Figure 19. This mismatch quantifies the difference between the given start state and the first time step of the predicted trajectory. In general, this value should be low, otherwise the trajectory "jumps" from the current start state to the first time step. For this aspect, the optimization is more difficult for the spline VAE, since matching the start state is formulated as a reward. This reward term can counteract other terms. For example by better matching the start state, the goal reward may be reduced when the robot points in the wrong direction at the start state. This may partially contribute to the better rewards of the spline VAE for the 6s horizon, since matching the start state stays in direct competition with the other reward terms. In contrast, the autoregressive VAE

indirectly includes the start state as condition for the decoder, which makes the optimization simpler by not requiring the additional matching rewards. Thus, for the autoregressive VAE this aspect is not an issue.

With the 4s horizon, the autoregressive VAE shows a faster convergence over time. Here, the reward distribution over the components is similar for both VAE variants and the differences between the better and worse component are more subtle. In general, optimizing a shorter trajectory is a simpler task since fewer dimensions need to be considered. Note that since the trajectory start position is beside the step obstacles, for shorter prediction horizons the fraction of the trajectory interacting with the obstacles is also shorter, further simplifying the task.

For both time horizons and VAE variants there is a low number of outliers where the obstacle boundaries are violated at some time steps of the trajectory, see third row in Figure 19. It is important to note that we can not enforce the obstacle avoidance with hard constraints, but only rewards. The obstacle avoidance rewards are always in competition with the other reward terms. Thus, it generally cannot be ensured, that only trajectories are produced, which do not violate the obstacle boundaries. For the final component weights, shown in the last row of Figure 19, we can observe that the final GMM distribution is fully dominated by one component for almost all optimizations. This is not ideal, since we want our final distribution to be as versatile as possible. Nevertheless, multiple components are still considered during the optimization, and the algorithm may switch out the best component multiple times during the iterations. We only show the Figure of the final component weights for this scenario, since for the other scenarios the weight distribution looks very similar.

In Figure 20 we present the reconstructed samples for the final GMM distribution in the top row for the 6s horizon for one scenario configuration. We can observe that both VAE variants produce reasonable trajectories that lead to the goal and avoid the stepping obstacle when being in contact. Note that not fully reaching the goal does not indicate a problem, as the considered horizon may be too short to reach the specific goal location in time. With both variants, the optimization converges to a solution where the robot walks slightly sideways, which is advantageous for this scenario to hit the allowed stepping locations.The variance of the final distribution of trajectories is very low, partially due to the samples only coming from one component. This also indicates that the variance of individual components is also low. At the same time the top sample from different components show higher diversity, see the bottom row of Figure 20.

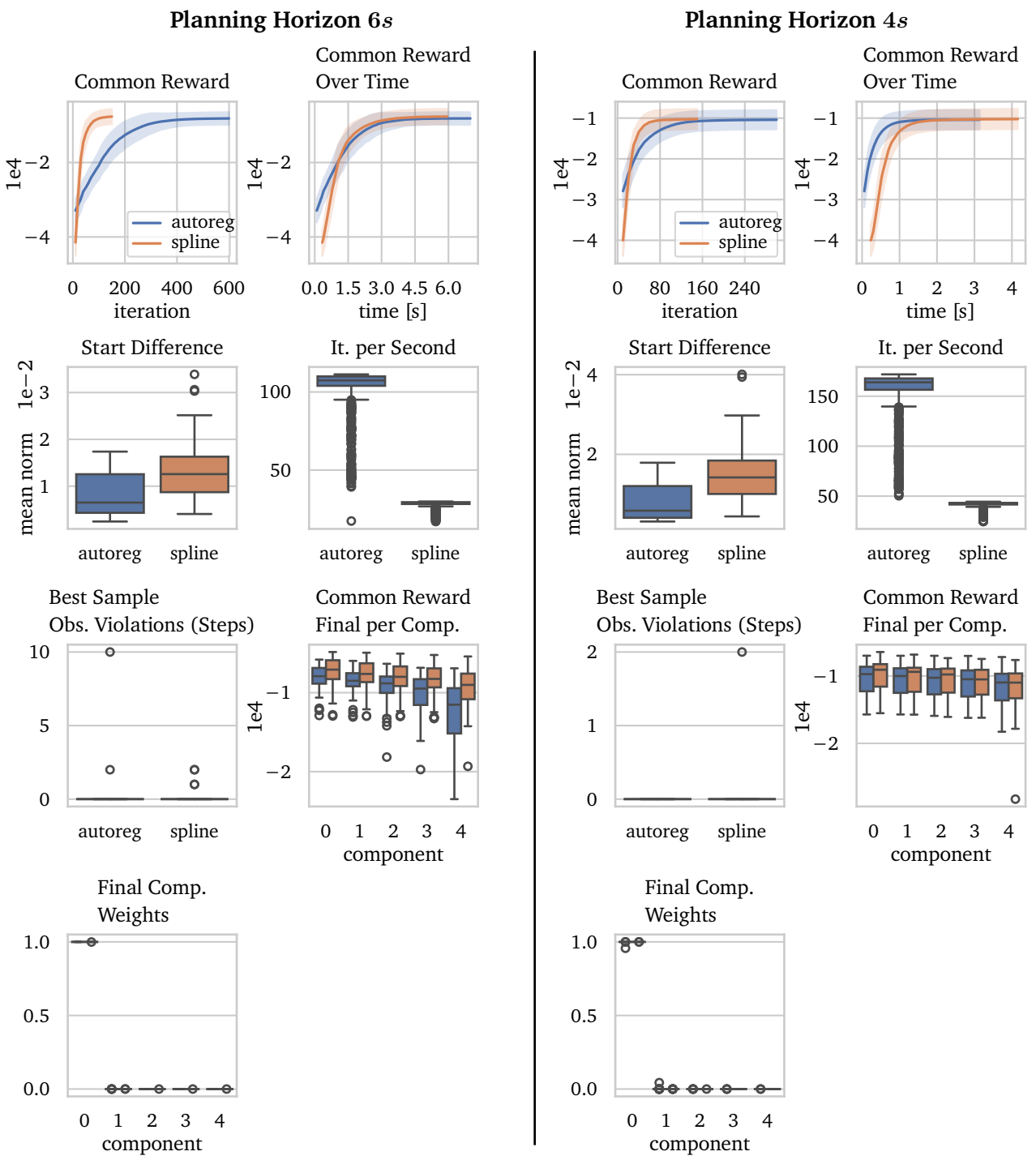


Figure 19: GMM trajectory optimization for the *Step Checkerboard* scenario. We show metrics for the 6s planning horizon on the left and metrics for the 4s planning horizon on the right. The plots in the first row show the common rewards sum over iterations and time. In the second row we show the total start state difference between the given start context and the first prediction time step as mean over the base position, base rotation angle and absolute feet positions. Here, we also show the iterations per second. In the third row, the obstacle violations Figure (Best Sample Obs. Violations) displays the distribution over the number of time steps of each run where obstacle boundaries are violated. Here, we also display the final common reward sum distribution per component (sorted by the mean component reward). The last row shows the final GMM component weights. Note that here we also differentiate between the VAEs, but this is hardly visible due to the small boxes. For the start state difference and obstacle violations plot, we consider the final best sample. The line plots show the mean as a line and the standard deviation as shaded areas.

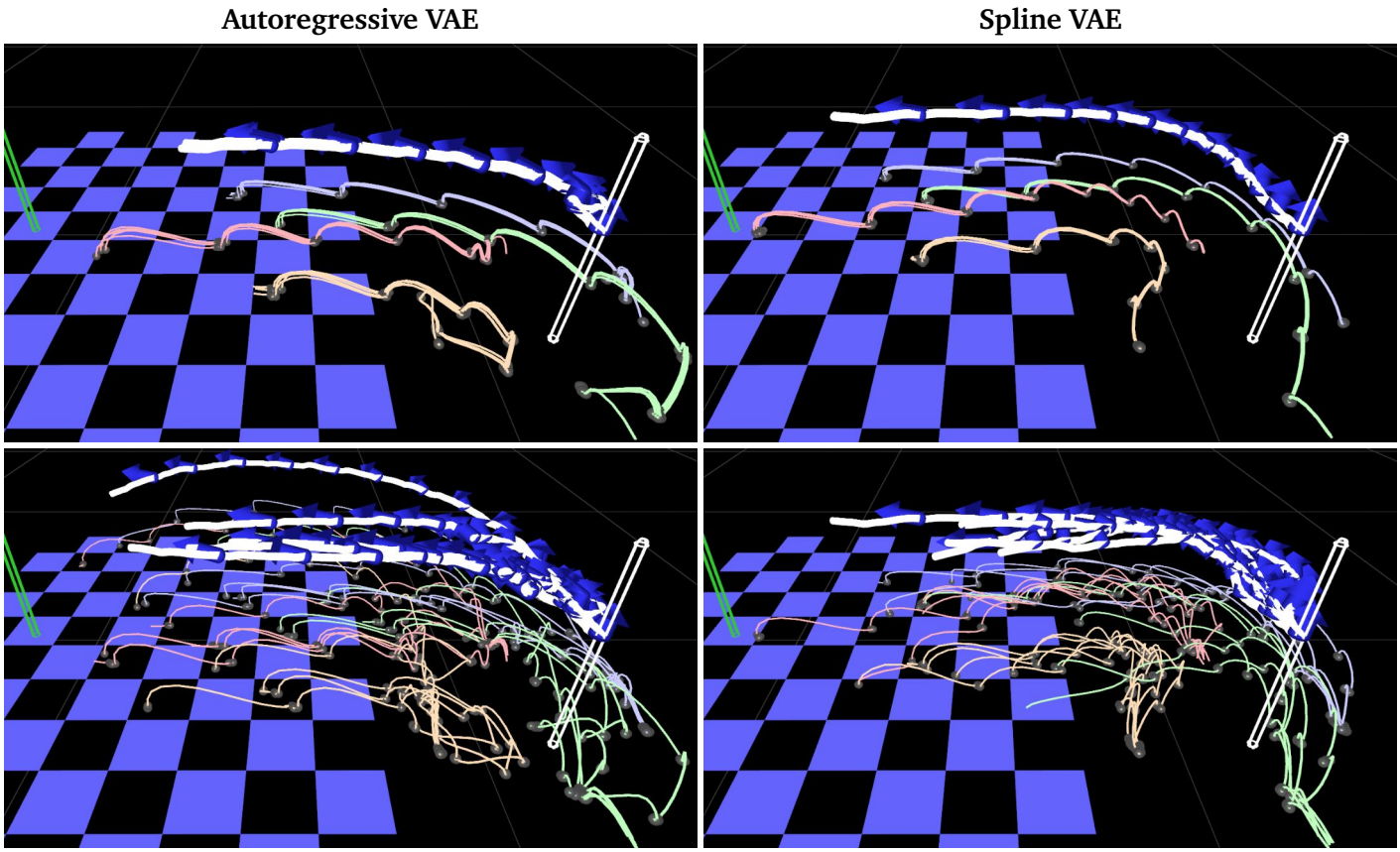


Figure 20: We show the reconstructions of 5 samples of the final GMM absolute trajectories for one run in the *Step Checkerboard* scenario ($6s$ horizon). On the left, we show the samples for the autoregressive VAE and on the right for the spline VAE. The top plots show 5 samples from the full GMM. The bottom plots illustrate one best sample from each component. The white pole denotes the start and the green pole the goal position. The white line shows the planned base trajectory where the blue arrows indicate the base rotation. The four colored lines on the ground each show one foot trajectory, where planned contact is indicated by grey dots. Normal obstacles are shown in gray, while foot stepping obstacles (non allowed ground-contact areas) are shown in blue.

Step Crosswalk Scenario

Here, we see a similar picture as with the *Step Checkerboard* scenario in Figure 21. Except that here, the convergence rate over time for the autoregressive VAE is also higher for the 6s horizon. Again, the spline VAE shows a higher start state mismatching while having a better component reward distribution for the 6s horizon. For the shorter horizon, both component distributions are again more similar, showing generally high rewards over all components, except for a few outliers for the worst component.

When looking at the sample reconstructions in the top row of Figure 22, we see that with both VAE variants, the optimization produces good trajectories that walk towards the goal while avoiding the stepping obstacles. The optimization finds the required base rotation to walk over the gap. The final GMM samples show low variance. At the same time the top samples over the different components, see bottom row of the Figure, are versatile while all produce valid solutions. When

looking at the autoregressive VAE, we can also see one solution where the robot walks over the gap sideways.

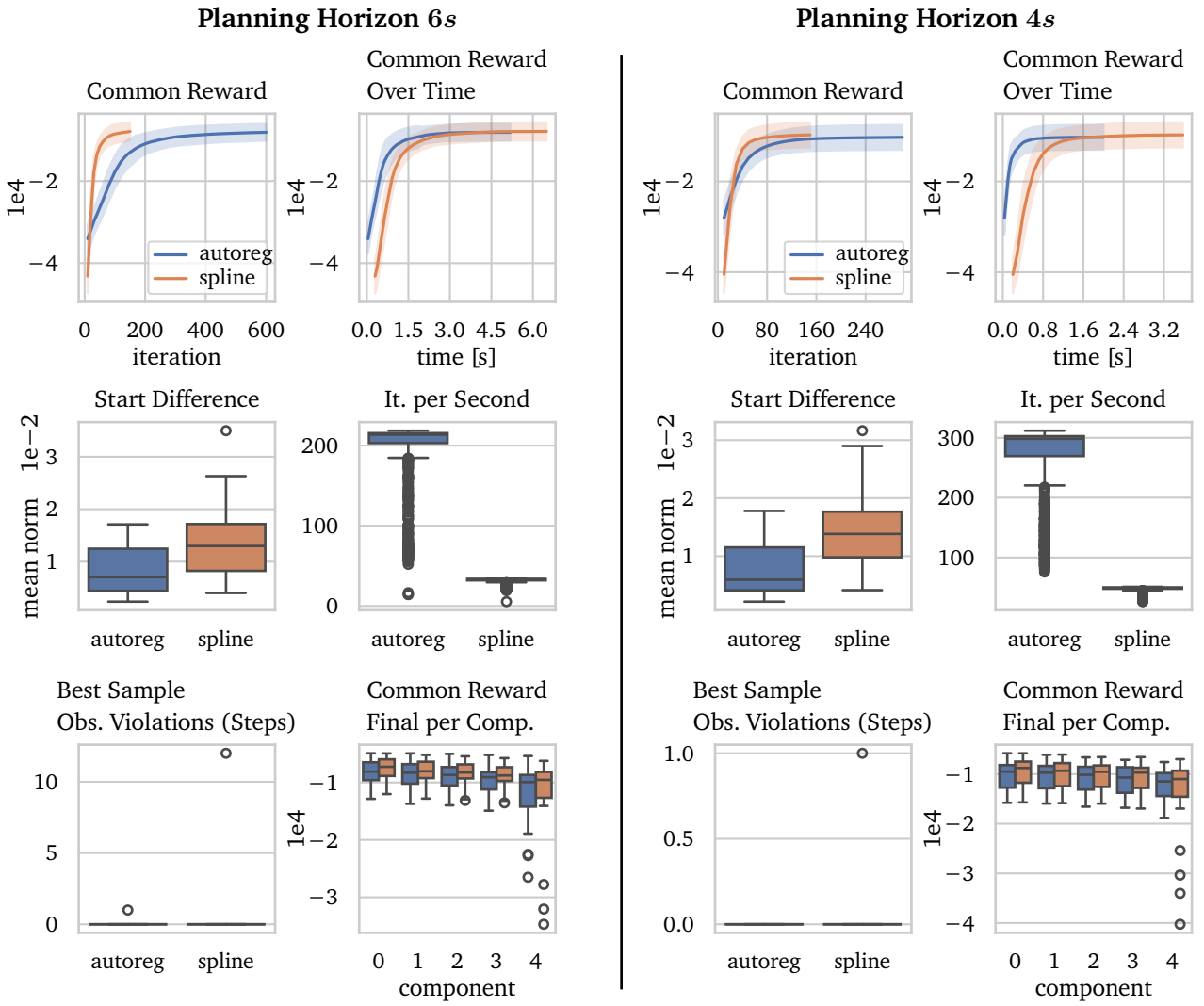


Figure 21: GMM trajectory optimization for the *Step Crosswalk* scenario. We show metrics for the 6s planning horizon on the left and metrics for the 4s planning horizon on the right. The plots in the first row show the common rewards sum over iterations and time. In the second row we show the total start state difference between the given start context and the first prediction time step as mean over the base position, base rotation angle and absolute feet positions. Here, we also show the iterations per second. In the third row, the obstacle violations Figure (Best Sample Obs. Violations) displays the distribution over the number of time steps of each run where obstacle boundaries are violated. Here, we also display the final common reward sum distribution per component (sorted by the mean component reward). For the start state difference and obstacle violations plot, we consider the final best sample. The line plots show the mean as a line and the standard deviation as shaded areas.

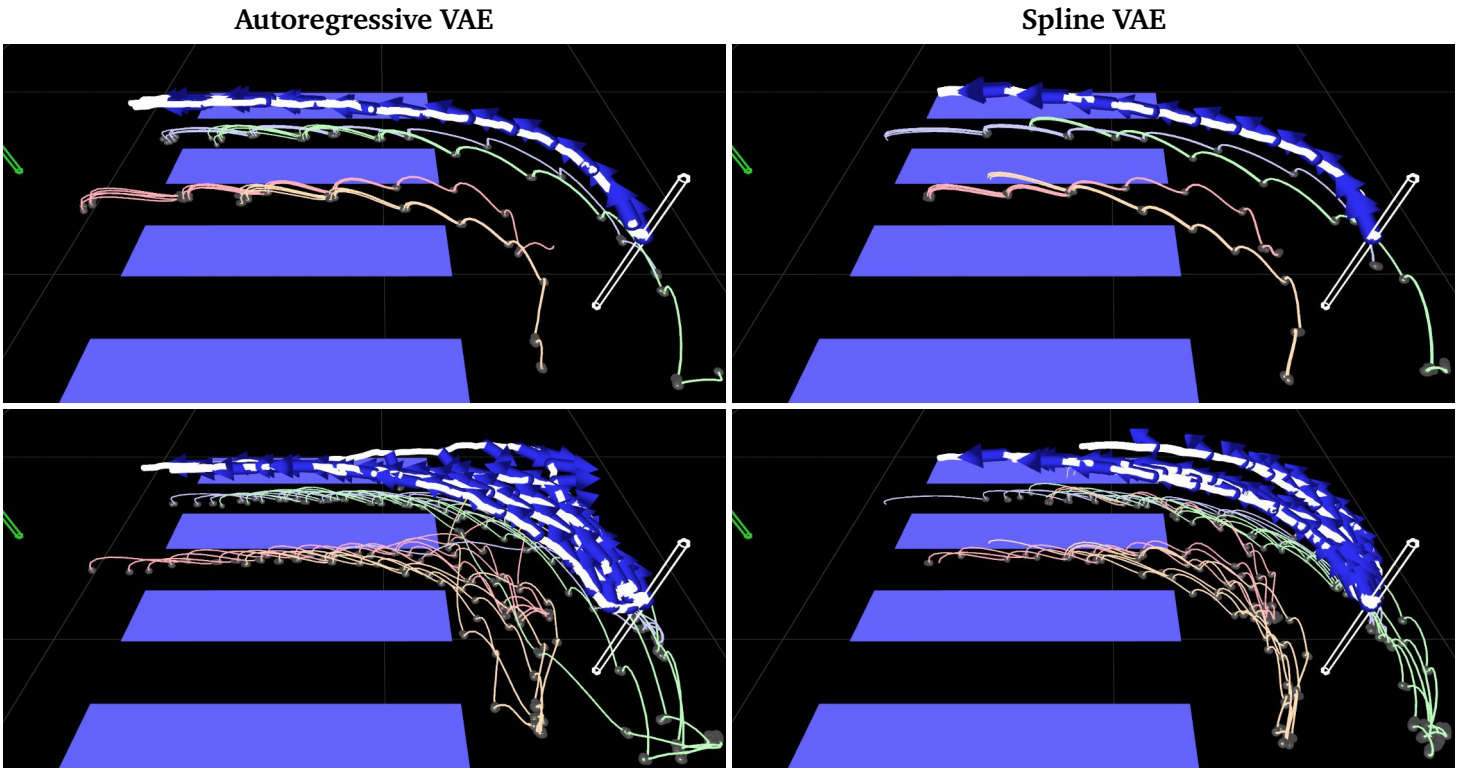


Figure 22: We show the reconstructions of 5 samples of the final GMM absolute trajectories for one run in the *Step Crosswalk* scenario ($6s$ horizon). On the left, we show the samples for the autoregressive VAE and on the right for the spline VAE. The top plots show 5 samples from the full GMM. The bottom plots illustrate one best sample from each component. Normal obstacles are shown in gray, while foot stepping obstacles (non allowed ground-contact areas) are shown in blue.

Wall Gap Scenario

In this scenario, the convergence rate over time behaves similarly to the previous *Step Crosswalk* scenario since the autoregressive VAE is generally faster. The same holds for the start state mismatching and final component reward distributions. Note that here the spline VAE optimization leads to significantly higher number of runs where obstacle boundaries are violated, especially for the shorter horizon. For this scenario further tuning of the obstacle rewards may be required.

In the top row in Figure 24, we can observe that for both VAEs, the optimization finds good solutions where the body rotates around the z-axis in order to fit through the gap. Again, the final GMM versatility of the sample is low (slightly higher for the autoregressive VAE at the end of the trajectories). For the spline VAE, the top samples from the components (bottom row) all correspond to good solutions leading to walking through the gap. At the same time the autoregressive component samples look more versatile, but multiple samples are of lower quality (not walking through the gap).

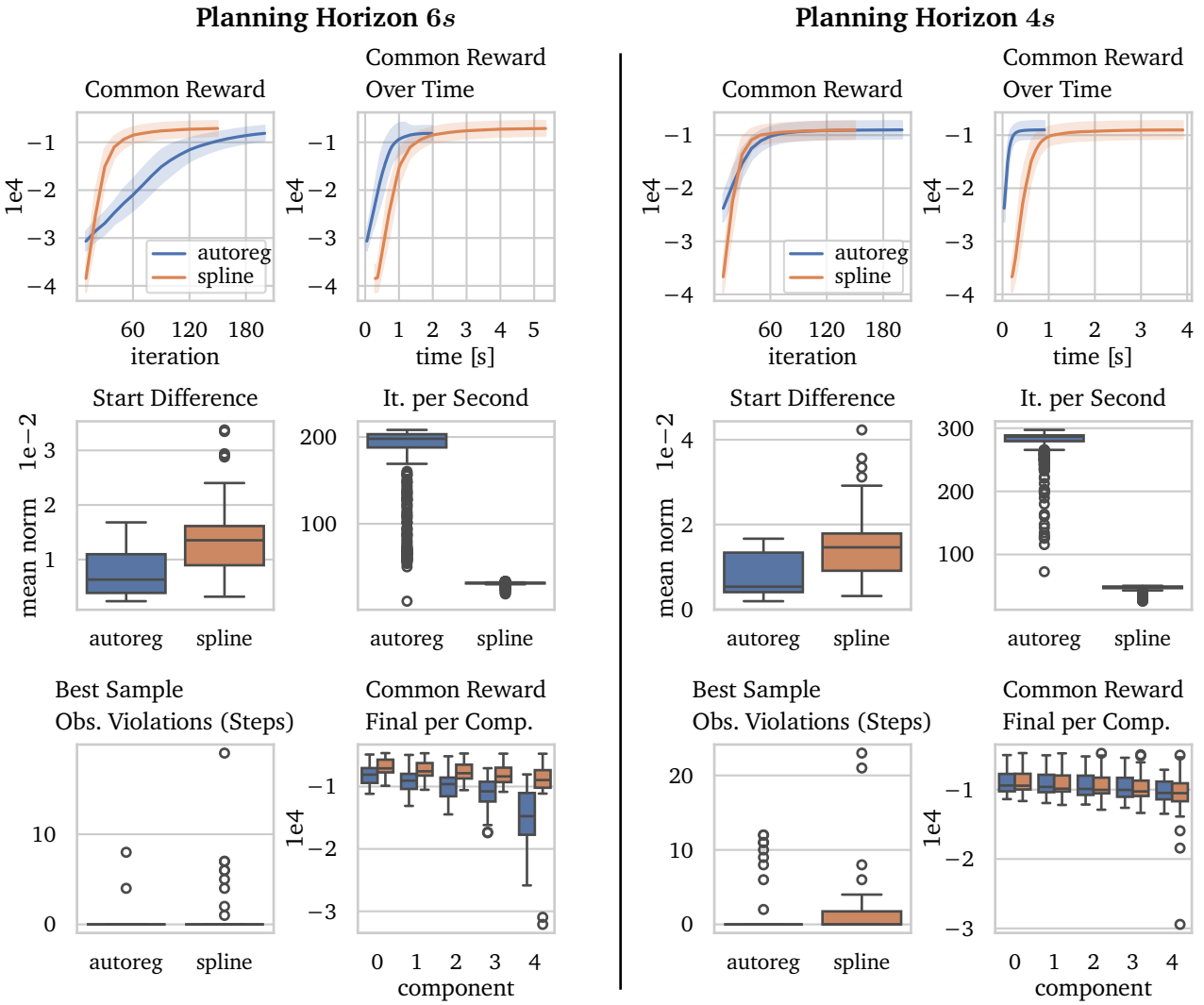


Figure 23: GMM trajectory optimization for the *Wall Gap* scenario. We show metrics for the 6s planning horizon on the left and metrics for the 4s planning horizon on the right. The plots in the first row show the common rewards sum over iterations and time. In the second row we show the total start state difference between the given start context and the first prediction time step as mean over the base position, base rotation angle and absolute feet positions. Here, we also show the iterations per second. In the third row, the obstacle violations Figure (Best Sample Obs. Violations) displays the distribution over the number of time steps of each run where obstacle boundaries are violated. Here, we also display the final common reward sum distribution per component (sorted by the mean component reward). For the start state difference and obstacle violations plot, we consider the final best sample. The line plots show the mean as a line and the standard deviation as shaded areas.

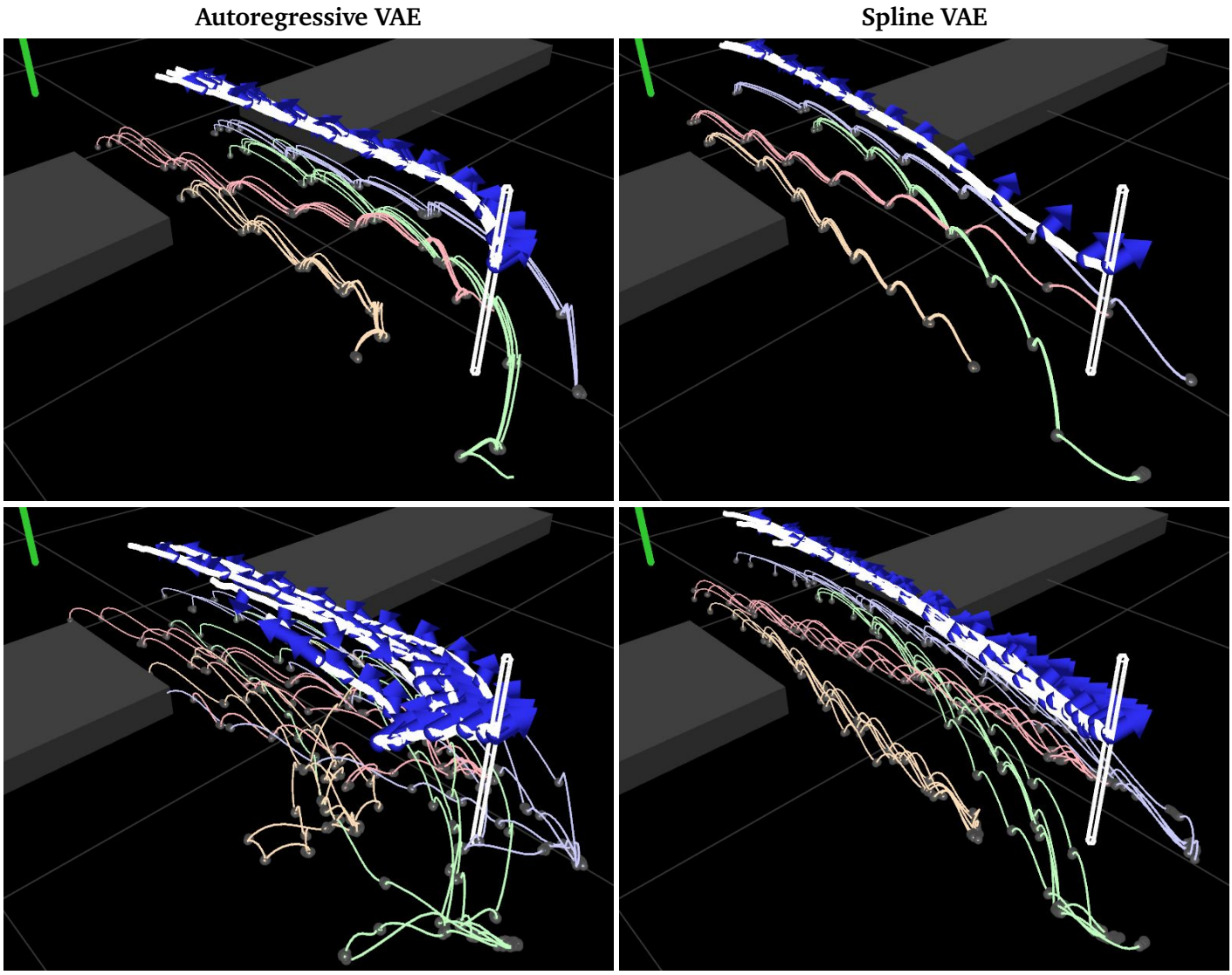


Figure 24: We show the reconstructions of 5 samples of the final GMM absolute trajectories for one run in the *Wall Gap* scenario ($6s$ horizon). On the left, we show the samples for the autoregressive VAE and on the right for the spline VAE. The top plots show 5 samples from the full GMM. The bottom plots illustrate one best sample from each component. Normal obstacles are shown in gray, while foot stepping obstacles (non allowed ground-contact areas) are shown in blue.

2 Goal Scenario

Here, the convergence rate over time behaves analog to the previous *Step Crosswalk* scenario, see Figure 25. The same holds for the start state mismatching and component weight distribution. Note that for this scenario, the worst component of the spline VAE has a few outliers with very low rewards.

The final GMM samples, shown in the top row of Figure 26, successfully pass through both goals and show slightly higher versatility than in previous scenarios, especially towards the end of the trajectories. Again, the samples from the different components show higher variance.

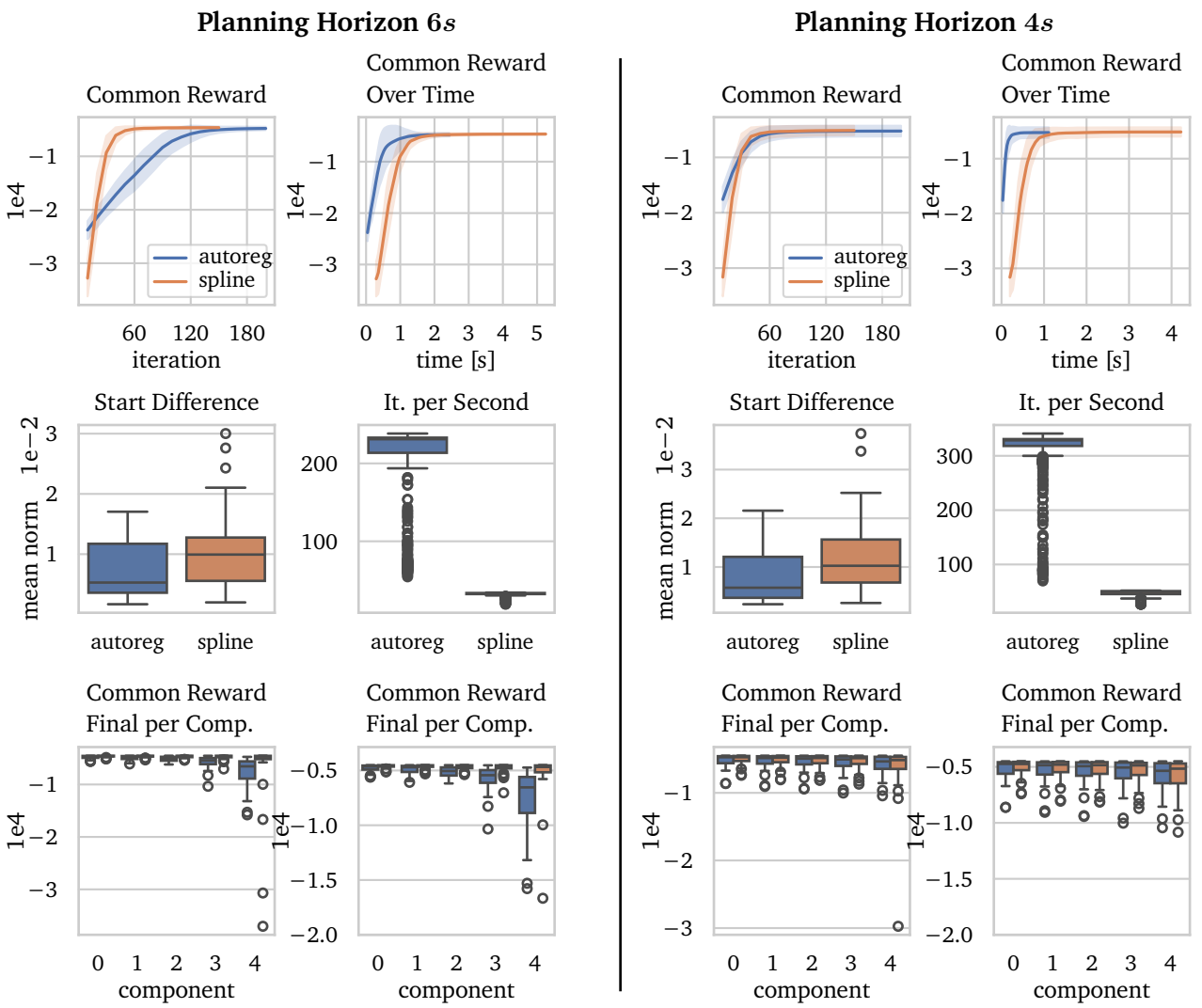


Figure 25: GMM trajectory optimization for the *2 Goal* scenario. We show metrics for the 6s planning horizon on the left and metrics for the 4s planning horizon on the right. The plots in the first row show the common rewards sum over iterations and time. In the second row we show the total start state difference between the given start context and the first prediction time step as mean over the base position, base rotation angle and absolute feet positions. Here, we also show the iterations per second. In the last row, we display the final common reward sum distribution per component (sorted by the mean component reward). We show this plot a second time in the last row with the y range clampled to a minimum of $-2 \cdot 10^4$ for better visiblity. For the start state difference plot, we consider the final best sample. The line plots show the mean as a line and the standard deviation as shaded areas.

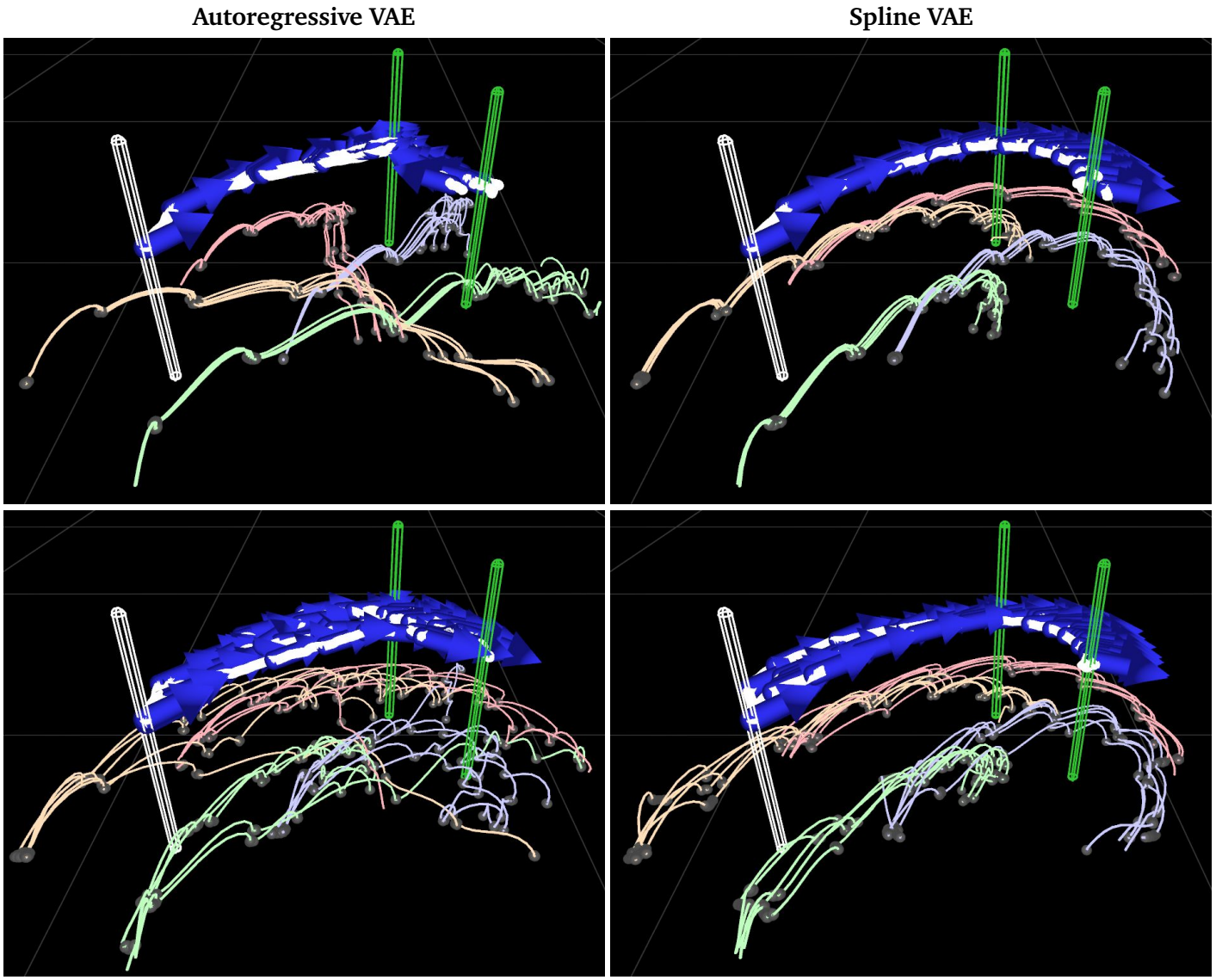


Figure 26: We show the reconstructions of 5 samples of the final GMM absolute trajectories for one run in the *2 Goal* scenario ($6s$ horizon). On the left, we show the samples for the autoregressive VAE and on the right for the spline VAE. The top plots show 5 samples from the full GMM. The bottom plots illustrate one best sample from each component. Normal obstacles are shown in gray, while foot stepping obstacles (non allowed ground-contact areas) are shown in blue.

4.3.2 Low-Level Optimized Trajectories Tracking Results

When looking at the low-level policy tracking errors in Figure 27, we can observe that the mean tracking errors are in a low range ($< 3\text{cm}$ for the base and feet and $< 0.03\text{rad} = 1.72°$ for the orientation). We can see that the majority of the distributions is within this low range error region, indicating that the policy can generally track the references well. When comparing the tracking of trajectories produced with the different VAE variants, we can see that the distributions look very similar. But we can also observe that there are some large outliers in all error categories. Thus, for a few time steps, the policy strongly deviates from the reference. This is especially noticeable for the

more complex task with obstacles. One potential reason for this may be a mismatch between the distribution of the references used in training the tracking policy and the distribution of references produced by the optimizations. In this context, the optimization may produce trajectories that differ more from the dataset for the more difficult scenarios to solve the task, leading to higher tracking error outliers for the *Step Crosswalk* and *Step Checkerboard* scenarios when looking at the base tracking. This potential issue could be addressed by further fine-tuning the tracking policy on trajectories produced with the optimization by including these optimized references in the policy training.

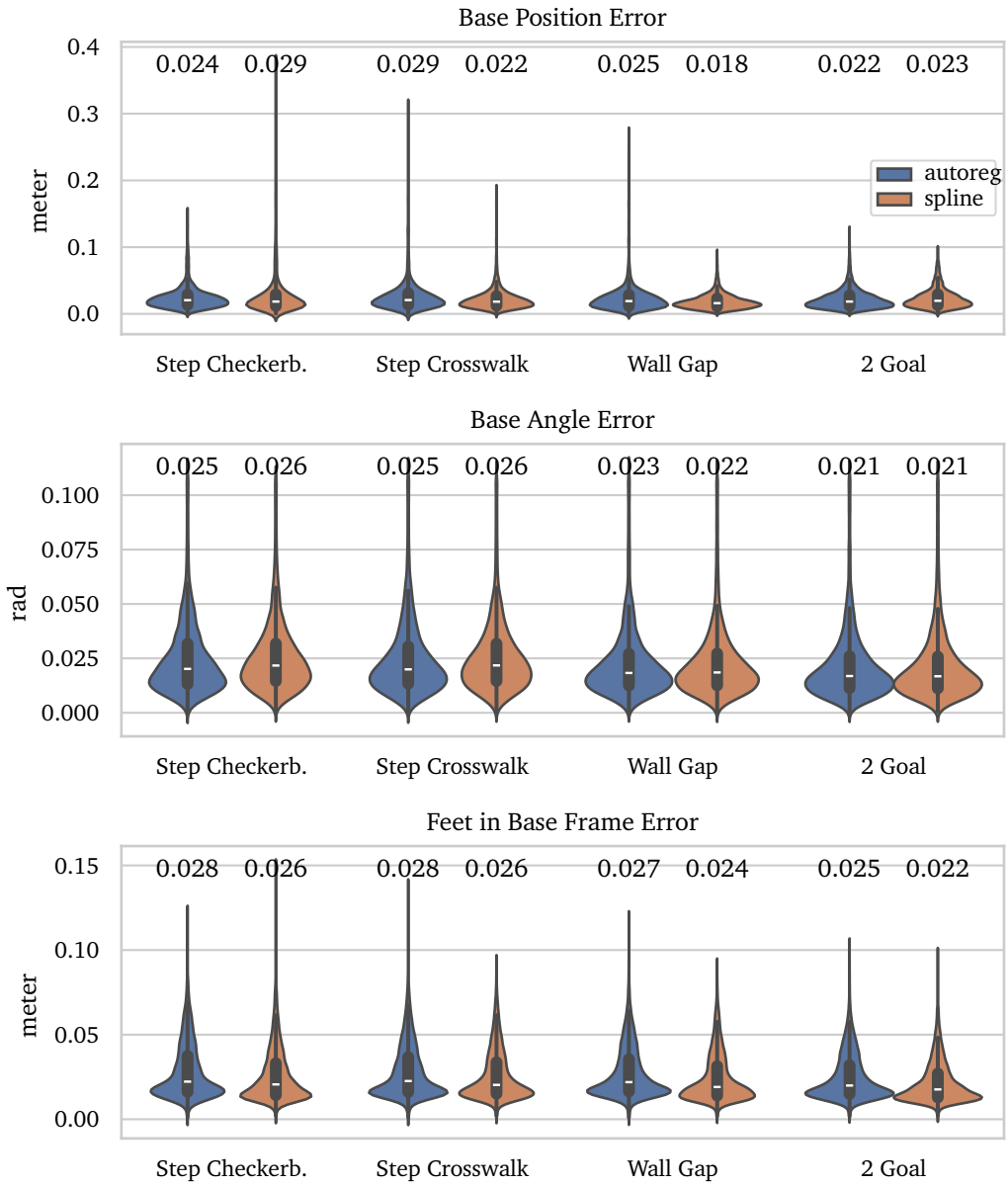


Figure 27: Policy tracking error per time step for the $6s$ planning horizon. We show the error distribution over all time steps and runs of each scenario. Inside each violin plot, a box plot spans from the first to the third quartile, while the small horizontal bars inside the boxes denote the median. Above each violin plot, we present the mean value. Note that for the feet tracking error, we show the distribution of the mean error over all feet. The base angle error refers to the angular difference in radians between the target and the actual base orientation.

4.3.3 GMM Optimization Variations

We show the results for varying the number of components and samples per component for both VAE types in Figure A.5 and Figure A.6. We can observe that, in general, a total higher number of samples per iteration leads to higher final rewards. This is expected, since the total number of iterations is fixed per scenario and VAE variant, which leads to the optimization receiving more total samples when the number of samples per iteration is higher. Note that a higher number of samples per iteration also corresponds to longer total optimization time. We choose our default configuration of 5 components and 50 samples per component as trade-off between runtime performance and final achieved reward.

4.3.4 Conclusion

We can generally say that the spline VAE is more suited for longer planning horizons with more complex tasks where it can compete with the optimization time of the autoregressive VAE. Here, the spline VAE shows better reward distributions of final components. On the other hand, the autoregressive VAE achieves better matching of the given start state. Across all tasks and VAE variants, the diversity of the final GMM distribution is low and one component dominates the distribution. Also the versatility of reconstructed samples of this final dominant component is low, showing more variance toward the final time steps of the reconstructed trajectories. We see two potential reasons for these issues. First, the reconstructed trajectories may be very sensitive to small latent value changes. On one side, this may come from the fact that the decoders produce the base trajectory (for position and z-rotation) as velocity over time, which is integrated afterward to get the absolute trajectory. This way, small changes of the base velocity in the beginning of the trajectory may drastically change the final position. Thus, the optimization algorithm may be forced to keep the variance of especially early latent values small to solve the task (e.g. to not collide with an obstacle towards the end of the trajectory). The autoregressive nature of the autoregressive VAE may additionally increase the sensitivity of latent values corresponding to earlier time steps[10]. Second, to achieve good final trajectories, high scaling of the inductive bias reward terms is required. Increasing the scaling of rewards generally reduces the effect of the entropy regularization term in the optimization objective, since the maximization of rewards will dominate, see Equation 17. Both of these potential explanations require further experimental investigation.

As potential improvement, other reference trajectory representation models may be explored that directly produce absolute trajectories and perform the reconstruction not autoregressively. Additionally, these models may consider the start state as conditional input, such that no additional start state matching reward is required. Better reconstruction quality across all potential latent values may reduce the need for the inductive bias reward terms.

[10] E.g. the first latent value will indirectly affect the reconstruction outputs of all following decoder passes.

4.4 Simulation Closed-Loop MPC Trajectory Optimization

In this experiment we want to evaluate the performance of our planning framework in the closed-loop MPC deployment setting. Therefore we repeatedly run our optimization algorithm with the current state as start context. We then execute the resulting reference trajectory for 1s (50 time steps) in simulation with the tracking policy, before replanning from the new current state, see Section 3.4.6.1. We consider 3 different scenarios, where our MPC planning loop is executed for a scenario-specific number of time steps. Each scenario specifies a list of multiple goal positions, where we only consider one goal at a time. When the robot reaches a goal, i.e. its base is in a distance range of $< 15\text{cm}$ to the goal, we switch to the next goal in the goal list. After reaching the last goal we go back to the first one. We execute each scenario 3 times from the same start state, where the robot stands upright in its default pose, with different GMM optimization seeds.

Since in reality, the optimization requires some runtime (see Section 3.4.6.1), we simulate different optimization delays ($0, 7, 15$ time steps). We thereby execute all scenarios for all delay variants. For each delay, we perform one execution with and without the $r^{\text{match old ref abs}}$ reward to measure its effect.

Her,e we adapt the additional collision points that approximate the knee by changing their position offset from the default -0.175m (see Table A.11) to -0.2m, in order to ensure proper collision avoidance. We also change the rectangle obstacle tolerance value $s_{\text{obs rect tol}}$ to 0.2.

In the following we describe the individual scenarios. Here, we do not consider a similar scenario as the previous *Step Checkerboard* since we found that the longer required optimization duration ($> 300\text{ms}$) for this scenario leads to unstable behaviors when deployed closed-loop.

Slalom: In this scenario, we consider three normal rectangular box obstacles arranged in a line, with a gap of 0.75m between neighboring obstacles. The first two boxes have a size of $0.5\text{m} \times 0.5\text{m}$, while the last box is smaller ($0.4\text{m} \times 0.4\text{m}$). Here, 6 goal positions are placed right and left of the boxes, encouraging slalom-like trajectories. See Figure 28 for an illustration. Here, we execute our framework for 45s.

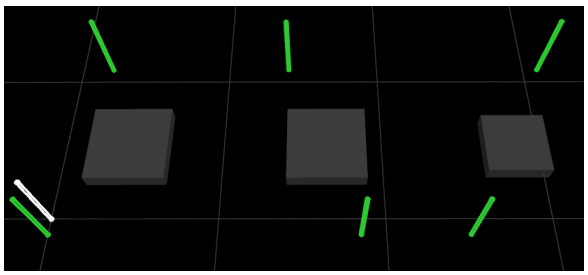


Figure 28: Slalom scenario obstacles and goals. The robot starts at the white pole position. It then walks to the green goals in the following order: top-middle, bottom-right, top-right, bottom-middle, top-left, bottom-left.

Crosswalk: This scenario is mostly equivalent to the *Step Crosswalk* scenario in Section 4.3. Here, we define two goals, the first is positioned on the other side of the crosswalk obstacles at $x = 2, y = 0$. Note that here we walk towards the positive x direction. The second goal matches the start position at $x = 0, y = 0$. We execute our framework for 45s in this scenario.

2 Goal: Similar to the *2 Goal* scenario in Section 4.3, we sample two goal positions that are at minimum 0.45m apart, such that the total distance is $\leq 1.5\text{m}$. We run each seed 10 times, each time with different sampled goal positions. We execute our framework for 10s.

Algorithm Configuration

We consider an optimization planning horizon of 100 time steps (2s), which ensures sufficiently fast optimization performance[11], see Figure 42 in Section 4.5. Only the autoregressive VAE can achieve this optimization runtime requirement, thus we only consider it for the MPC deployment setup. For this horizon length, 70 iterations are sufficient for convergence across all scenarios. Since we want to closely match a real-world setup and the real robot has no working foot contact sensors, we consider the autoregressive VAE version without the foot contact state as condition input. We use the default GMM optimization parameter from Table A.13. We apply some adaptations to the default reward weights for this MPC setup, see Table A.14. Note that only the end goal is used, and the mid goal is deactivated.

4.4.1 Results

First, we can look at one example execution for each scenario in Figure 29. Here, we consider the 15 time steps delay configuration with the $r^{\text{match old ref abs}}$ reward enabled. The robot successfully walks around the obstacles for the *Slalom* scenario. Planned foot trajectories do not collide with the obstacles. Figure 30 displays the trajectory from the top view. We can observe that the robot passes through all goals[12] while navigating between the obstacles. The policy also tracks the base xy trajectory closely, except for some small deviations. Note that we can see small position jumps in the base reference in some places, e.g. at $x = 2.1, y = 0.7$. These are caused by the optimization delay when the reference is updated to a new result from the optimization. Here, the policy continues to follow the old reference while the optimization is performed. The $r^{\text{match old ref abs}}$ reward can only counteract this to a certain degree. This can also be observed in Figure 31, where at some time steps, the reference command slightly "jumps", e.g. the "FR Foot Pos" at 5.5s. Figure 31 also shows that, in general, the tracking policy tracks the reference well with some smaller deviations. We further investigate this later.

For the *Crosswalk* scenario the robot walks successfully over the stepping obstacles, see Figure 29. The optimization produces valid trajectories that keep the required base orientation for not stepping onto the obstacles. Figure 30 shows that the robot walks over both stepping obstacles closest to the two goals. The reference is again tracked closely. The robot walks over the horizontal center of the obstacles, which is required to avoid collisions during foot contact.

With the *2 Goal* scenario heading to the first goal then the second goal, the robot reaches both goals[12], see Figure 29 and Figure 30. In Figure 30 we can again observe jumps in the reference when it is updated causing larger tracking mismatches for some sections of the execution.

In Figure 32 we compare the different optimization delay configuration. We align the plots at the time step where the new reference is applied to the policy. As expected, for the no delay variant, the tracking error reduces after applying the new reference, since the new reference start will match the current state well. Afterward, the tracking error increases. For the 7 step delay, the

[11] Optimization delay $\leq 300\text{ms} \triangleq 15$ time steps

[12] Within the specified 15cm tolerance

tracking error does not reduce after switching to the new reference. For the base angle, the error increases. This effect is even stronger for the 15 steps delay, where both the position and base orientation tracking errors increase after applying the new reference. This matches the previously observed “jumps” in the reference commands, see Figure 31. The policy first needs some time to keep up with the discontinuous change, thus the tracking error decreases over time. We can observe that adding the $r^{\text{match old ref abs}}$ reward only improves the tracking for the 15 step delay configuration. Here, it helps to reduce the initial peak in the position tracking error.

Finally we show the tracking error distributions for the 15 step delay configuration, where the $r^{\text{match old ref abs}}$ reward is enabled, see Figure 33. We consider this configuration as a pessimistic estimation of a real-world scenario. It can be observed that the mean tracking errors are generally higher than in the open-loop deployment, see Figure 27. We primarily attribute this to the already discussed mismatch between the new applied reference and the actual current state caused by the optimization delay. Again, there are also some outliers where, at some time steps, the tracking error is large.

Conclusion

We demonstrate that our framework deployed as closed-loop MPC can successfully solve the presented scenarios, while optimization delays lead to discontinuities in the reference command causing larger tracking errors. The $r^{\text{match old ref abs}}$ reward helps with larger optimization delays, thus we use it in the following real-world experiments, see Section 4.5. For future work, reducing the optimization runtime may enable better tracking performance.

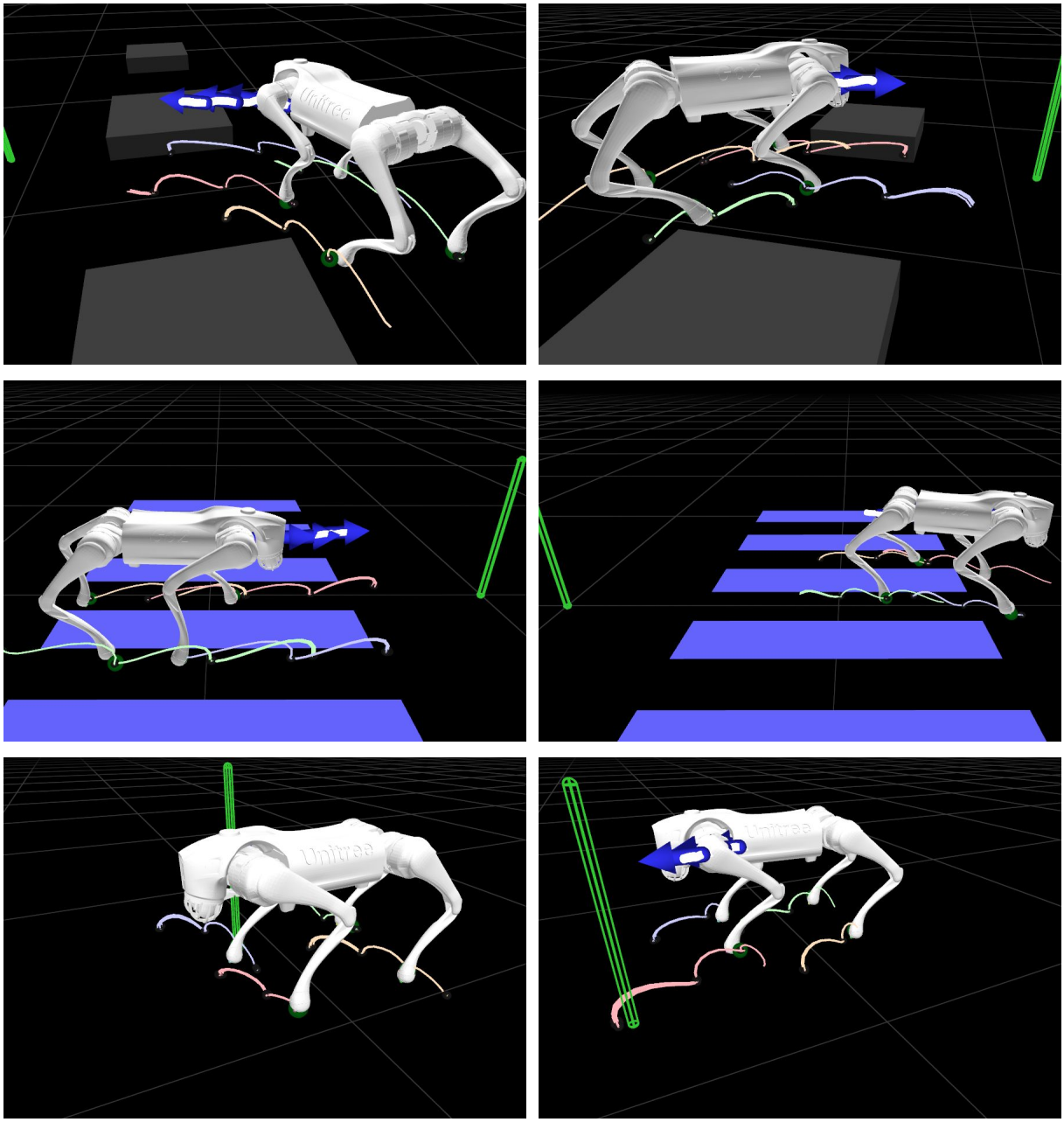


Figure 29: Example execution pictures for the closed-loop MPC deployment of the trajectory optimization and tracking policy for one seed per scenario (15 time steps delay with $r^{\text{match old ref abs}}$ enabled). Each row shows one scenario, from top to bottom: *Slalom*, *Crosswalk* and *2 Goal*. For each scenario we show two time steps. The robot visualization shows the current state in simulation. We show the current reference trajectory plan resulting from the last optimization as 5 samples from the final GMM. Green spheres denote the current foot target position given to the policy. In the top-left the robot walks from the start state to the second goal for the *Slalom* scenario. In the second picture in the first row, the robot walks towards the second goal. For the *2 Goal* scenario in the last row, the robot walks from the start state towards the first goal in the first picture. Subsequently, it walks to the second goal in the second picture.

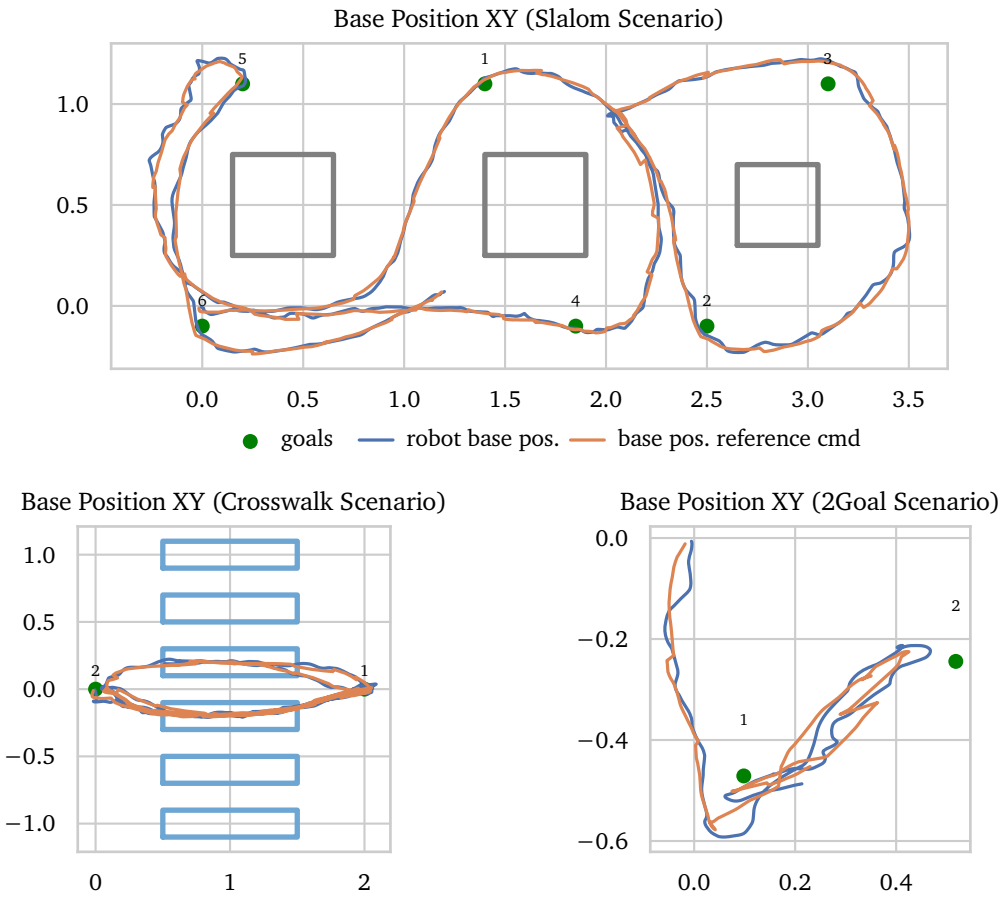


Figure 30: Base position over time for the same example executions shown in Figure 29. For each scenario we show the reference base position passed to the tracking policy, as well as the real robot base position. Normal obstacles are shown in gray and stepping obstacles (holes where the robot is not allowed to step onto, but can walk over) are shown in light blue. The number above each goal denotes its index in the goal sequence. The x and y axes show the x and y position coordinates in meters.

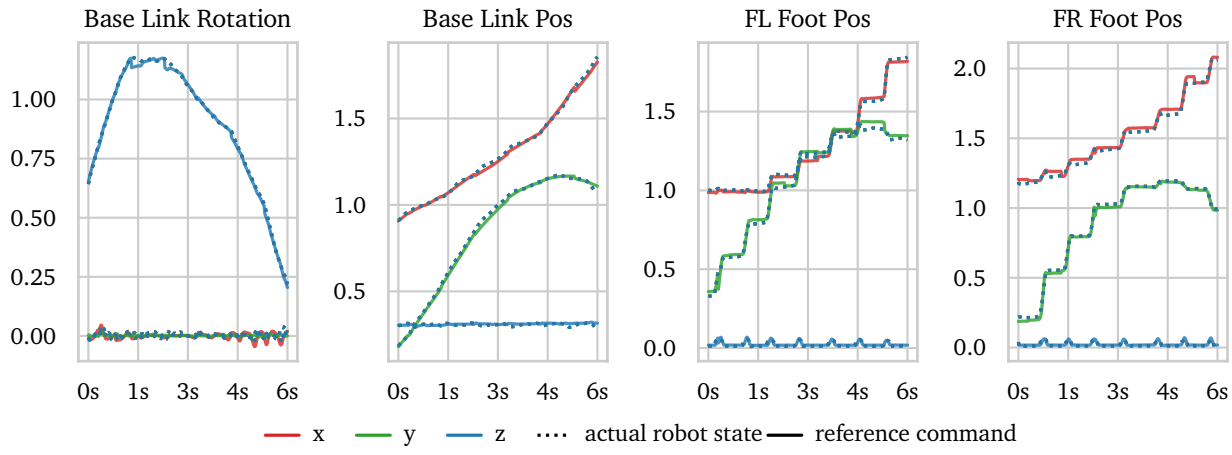


Figure 31: Execution of one seed for the *Slalom* scenario (same as in Figure 29 and Figure 30). We show the base pose and absolute front feet positions over time for 6s, beginning after the first 4s of the execution. The y-axis corresponds to radians for the rotation plot and to meters for all other plots.

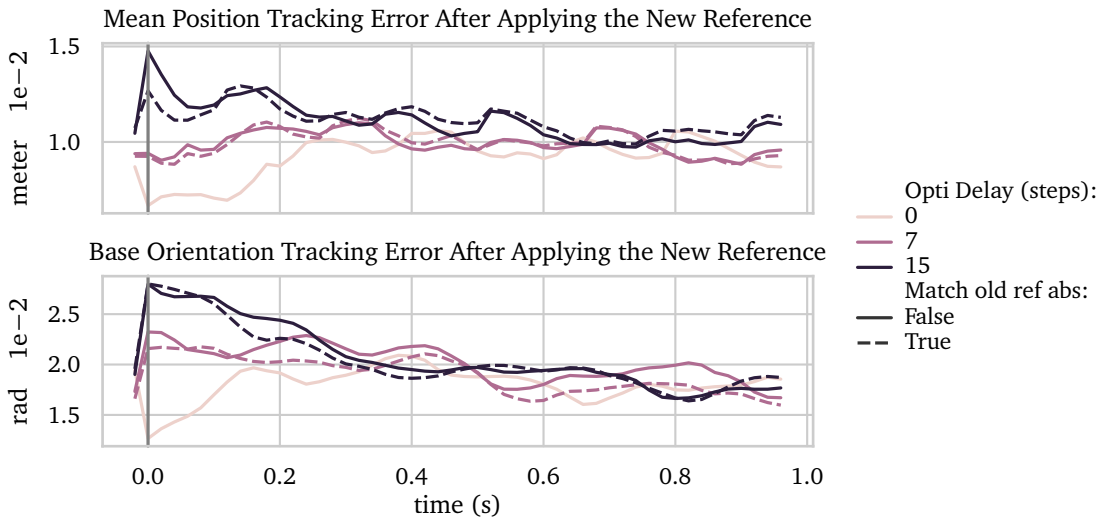


Figure 32: Total mean tracking error over all feet (in base frame) and the base position (top Figure) and separately the base orientation angle mean tracking error (bottom Figure) for MPC in simulation. We align the plots at the point in time where the new reference is applied to the policy (at $t = 0$). The time step prior to applying the new reference is shown at $t = -0.02$.

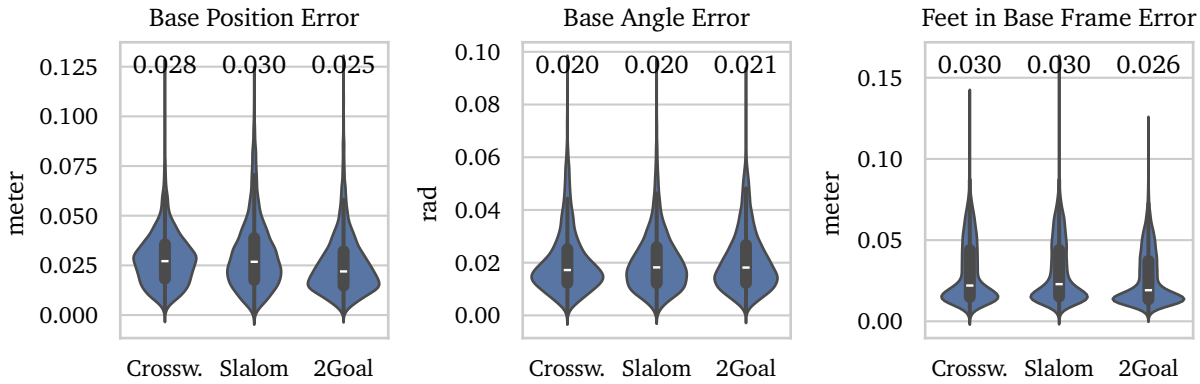


Figure 33: Policy tracking error per time step for MPC in simulation with delay of 15 steps and $r^{\text{match old ref abs}}$ enabled. Above each violin plot, we show the mean value. Note that for the feet tracking error we show the mean error over all feet.

4.5 Real-World Trajectory Optimization

In this experiment, we want to evaluate our framework in a real-world setting, where we execute planned trajectories on a real Go2 quadrupedal robot. We consider the previously described open-loop configuration with both VAE variants and the 6s planning horizon, see Section 4.3. We also execute the closed-loop variant with the autoregressive VAE with the configuration described in Section 4.4. The evaluation is performed on the same 3 scenarios as in Section 4.4. We also use the same reward hyperparameter adaptations as in Section 4.4. For the *Slalom* and *Crosswalk* scenarios we performed 3 executions with different GMM optimization seeds. With the *2 Goal* scenario, 4 random goal position configurations are considered, where we execute each configuration with just one seed. For every execution of a scenario, we use the same start position[13] and pose where the robot stands upright in the default joint configuration. For the closed-loop deployment, the execution of each scenario is analog to Section 4.4 running our framework for the specified duration.

With the open-loop configurations, we plan from the fixed starting pose towards the first goal for the *Slalom* and *Crosswalk* scenarios. Also the mid goal is considered for the *2 Goal* scenario, where the first goal position is set to the mid goal target and the second goal position is considered as end goal. During optimization the robot keeps its default pose via PD control. After obtaining the optimization result, the low-level policy is activated, which tracks the reference trajectory for the full 6s duration. For the *Slalom* scenario, we execute each seed twice, the second time starting from a second position, which is beside the second obstacle box. Although the real robot does not provide contact state information, we assume that at the start state all feet are in contact for the open-loop configurations since we always start from the default pose. The algorithm configuration is analog to Section 4.3, where the autoregressive VAE variant uses 600 iterations for the *Crosswalk* scenario and 200 iterations for the other scenarios. The spline VAE always performs 150 iterations.

Deployment

To estimate the absolute base pose, we employ an optical tracking system [41], where multiple cameras track reflective markers fixed on the robot's base body. The position and orientation data coming from the tracking system is transformed to the origin of the base frame. For the *Slalom* scenario we consider real box obstacles. The position and orientation of each box is also monitored by the optical tracking via markers on each box, see Figure 34. We use two large boxes of the size $0.45\text{m} \times 0.5\text{m}$ $(\text{x} \times \text{y})$, and one small box $(0.35\text{m} \times 0.4\text{m})$. For the *Crosswalk* scenario the stepping obstacles are only considered virtually at fixed positions. To account for the geometry of the room, we add one rectangular obstacle to model the back wall and one circular obstacle to represent a pillar in the middle of the room. These two fixed obstacles are considered in every scenario to avoid collisions. For the current state context provided to the reference optimization, we require the absolute feet positions. We obtain these by employing forward kinematics with the current joint angles and transform the result into the global frame with the robot base pose from the optical tracking system.

[13] We mark the base frame outline on the floor to repeatedly match the same start xy-position and z-rotation.

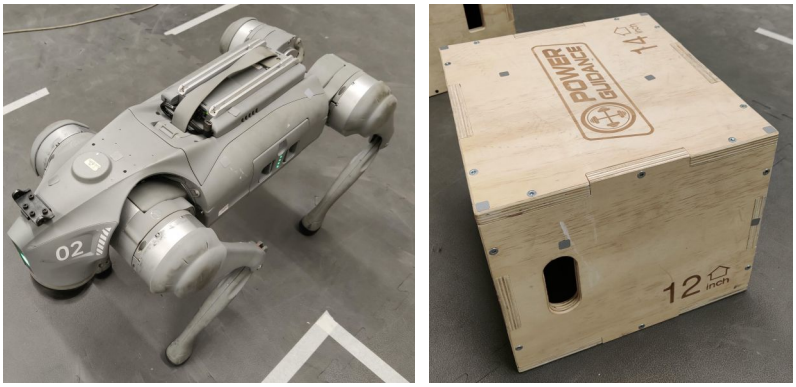


Figure 34: Real-world Go2 robot and box obstacle with tracking markers.

We execute both the low-level policy and trajectory optimization on a separate computer that is connected to the robot via an Ethernet cable. The optimization is executed on the GPU (NVIDIA RTX 3090) and the policy on the CPU. This way, the policy execution does not affect the performance of the optimization. Running the policy on the CPU is sufficient to achieve the required 50Hz update rate. Reference trajectories resulting from the optimization are copied from the GPU memory to the main memory to be accessible by the policy. The policy then sends joint commands via the network connection to the robot.

4.5.1 Results

We first consider the closed-loop setup, where we show qualitative examples in Figure 35, Figure 36 and Figure 37. For the *Slalom* scenario, we can observe that our framework successfully navigates the robot around the obstacles and passes all goals, see Figure 35 and Figure 38. Just as in simulation, the reference trajectories produce "jumps" when switching to a new optimization result, see Figure 38. For the base xy-position, the policy tracking shows larger deviations for some time periods. With the *Crosswalk* scenario, the robot avoids the stepping obstacles while reaching both goals, see Figure 36 and Figure 38. During returning to the first goal, the optimization chooses to walk over the obstacles sideways which is a valid solution, just as the previously forward walking approach. In the *2 Goal* scenario, the robot successfully reaches both goals in the available time frame (Figure 37 and Figure 38). In Figure 42 (left side), we can observe that the optimization delay is generally below 13 time steps. The *Crosswalk* scenario has the largest mean delay since it needs to consider the largest number of obstacles.

With the open-loop deployment setup, both VAE variants manage to walk from the start position through the gap between the obstacles and get close to the final goal for the *Slalom* scenario, see Figure 39. The final autoregressive VAE GMM shows slightly higher variance of the samples. The stepping obstacles are crossed successfully by both VAE optimizations in the *Crosswalk* scenario (Figure 40). Here, each VAE variant converges to a different solution. In the final *2 Goal* scenario, the GMM samples show higher variation, see Figure 41. The spline VAE result reaches the mid goal more closely, while both variants closely match the final goal at the end of the trajectory. When we look at the optimization time in Figure 42 (right side), we observe that only the autoregressive VAE achieves lower optimization duration than the considered planning horizon. For the *Crosswalk* scenario, both VAE variants need more optimization time than duration of the planned trajectory. Similar to previous experiments, the autoregressive VAE is generally faster.

In Figure 43, we show the tracking errors when executing the reference from the optimization. Generally, the tracking errors for the closed-loop MPC deployment are the highest, except for

some outliers of the open-loop spline VAE variant. We primarily attribute this to the previously discussed mismatches caused by the optimization delay. Both open-loop variants show similar tracking error distributions, except for some outliers for the base position in the *Crosswalk* task for the spline VAE. When comparing the mean errors with previous simulation experiments, see Figure 27 and Figure 33, the real-world experiment generally shows higher values. This is likely a result of state estimation errors but also could hint at potential required improvements to the policy sim-to-real transfer.

Conclusion

We showed successful deployment of our framework in a real-world setting with both, closed-loop and open-loop configurations. As already shown in the simulation experiments, see Section 4.4, tracking performance could potentially be improved by reducing the delay caused by the optimization.

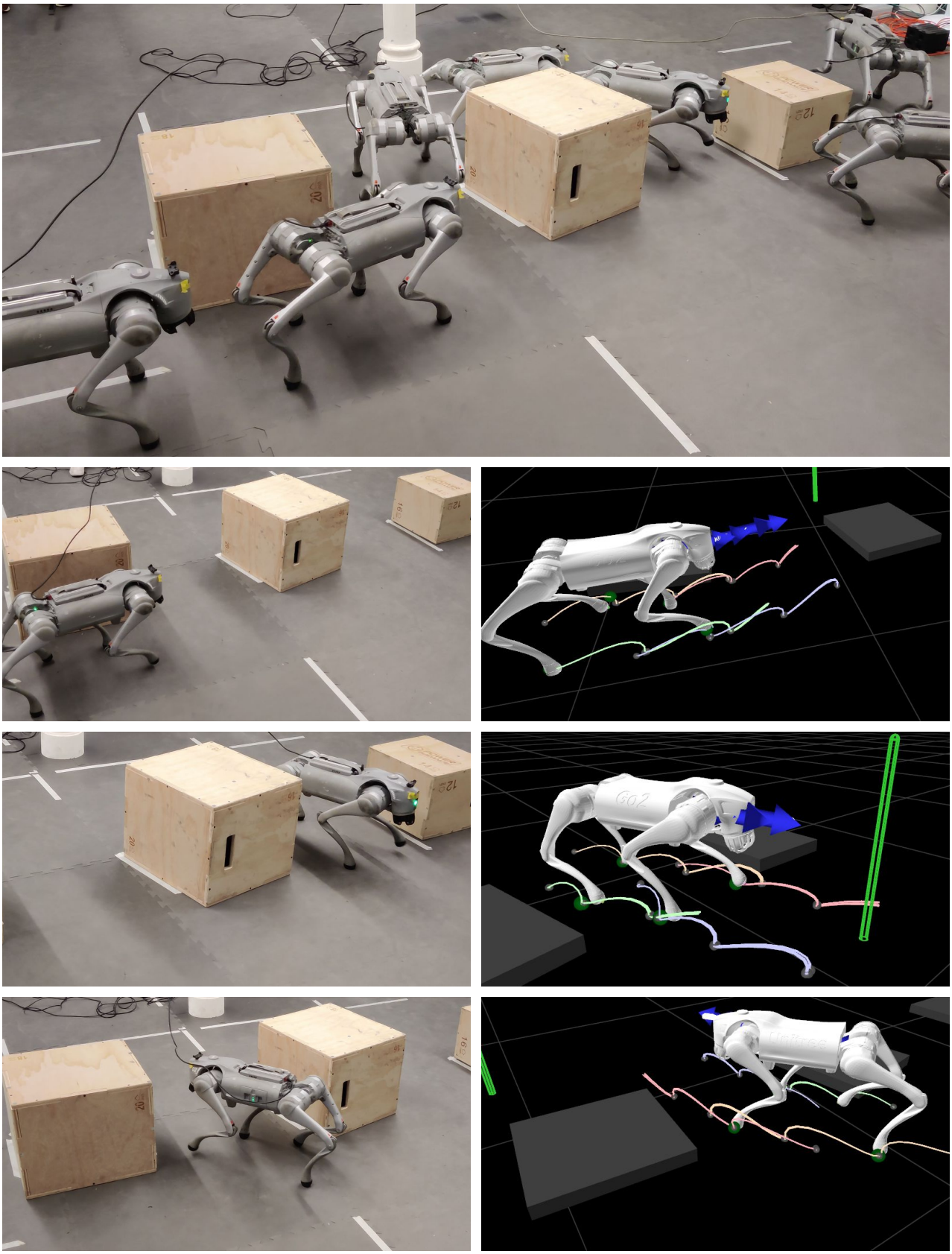


Figure 35: Real-world closed-loop trajectory optimization for the *Slalom* scenario for one seed. The top image overlays multiple time steps of the first half of the execution. Below, each row of pictures corresponds to one time step in the temporal order from top to bottom. The right column shows the current robot state and planned trajectory as 5 samples from the final GMM.

The top picture shows the full scene, the other pictures are zoomed in for better visibility.

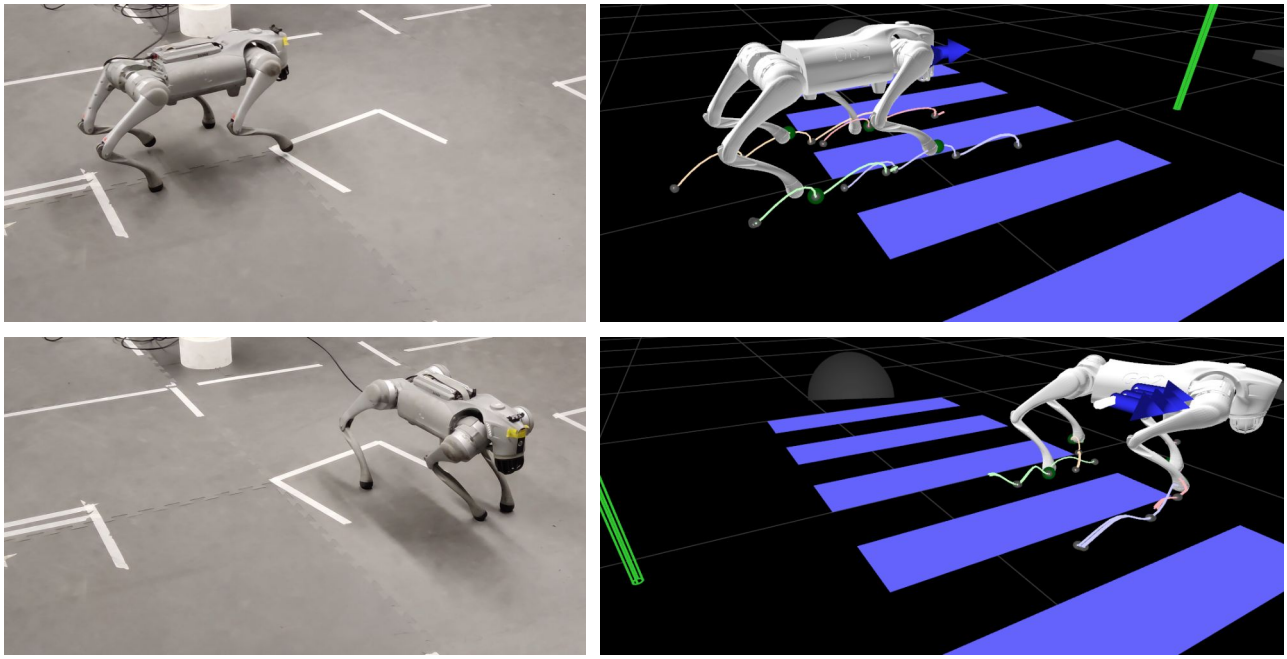


Figure 36: Real-world closed-loop trajectory optimization for the *Crosswalk* scenario. We show 2 time steps of one seed, where each row of pictures corresponds to one time step in the temporal order from top to bottom. The right column shows the current robot state and planned trajectory as 5 samples from the final GMM. Note that the stepping obstacles are only virtual and thus not visible in the real photos.

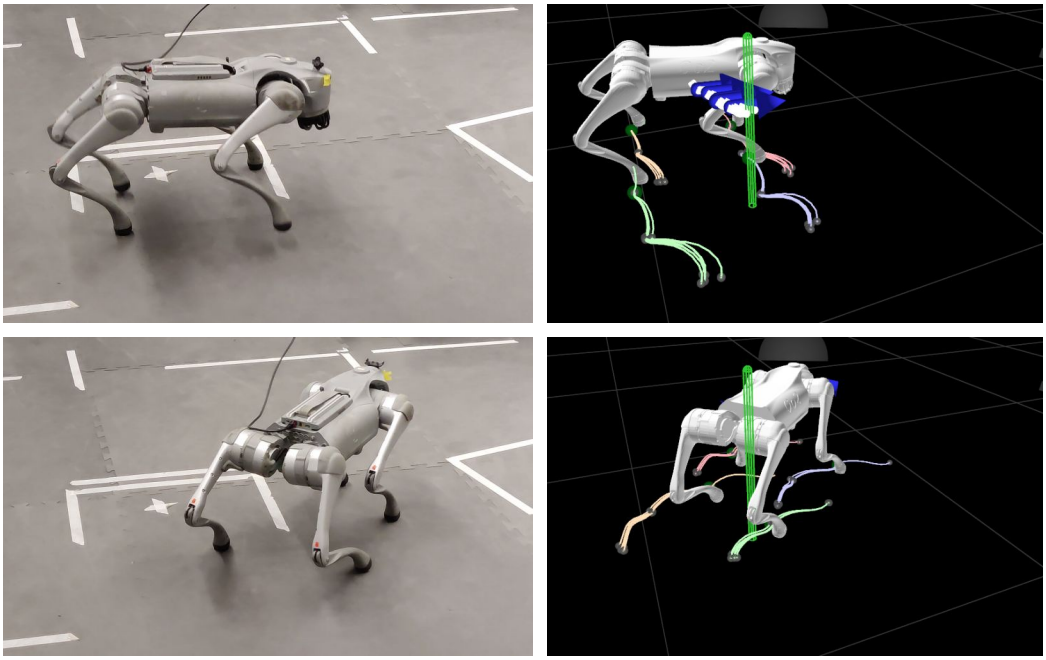


Figure 37: Real-world closed-loop trajectory optimization for the *2 Goal* scenario. We show 2 time steps of one seed, where each row of pictures corresponds to one time step in the temporal order from top to bottom. The right column shows the current robot state and planned trajectory as 5 samples from the final GMM.

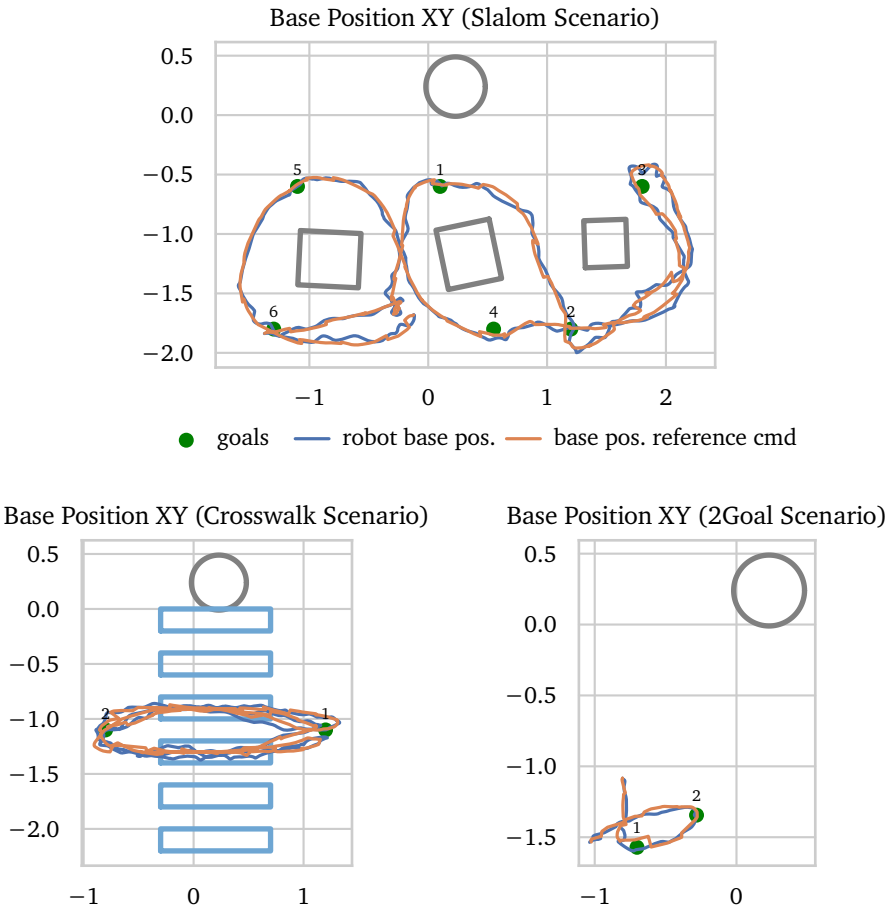


Figure 38: Real-world closed-loop trajectory optimization, showing base position over time for the same example executions displayed in Figure 35, Figure 36 and Figure 37. For each scenario we show the reference base position passed to the tracking policy, as well as the real robot base position. Normal obstacles are shown in gray and stepping obstacles (holes where the robot is not allowed to step onto, but can walk over) are shown in light blue. The number above each goal denotes its index in the goal sequence. The x and y axes show the x and y position coordinates in meters. The fixed back wall obstacle is outside of the plots, but the pillar room obstacle is visible.

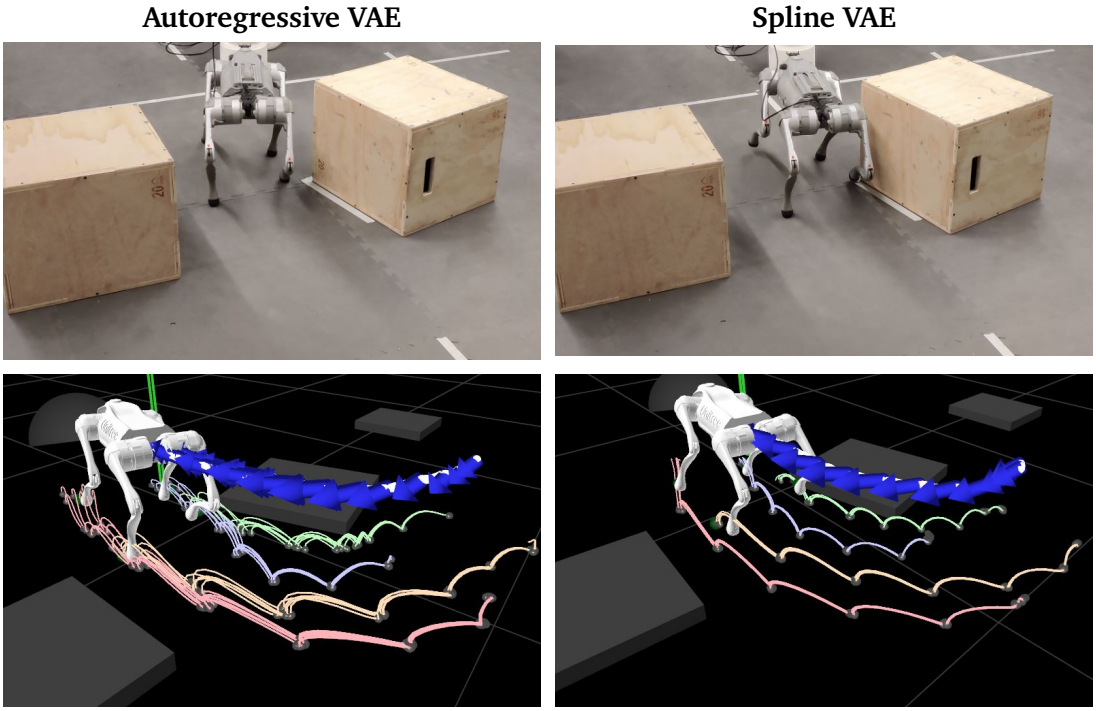


Figure 39: Real-world open-loop trajectory optimization for the *Slalom* scenario (6s planning horizon). We show the final time step of on seed with the second starting position, which is beside the center obstacle. Each column corresponds to one VAE variant. The bottom row shows the final robot state and planned trajectory as 5 samples from the final GMM.

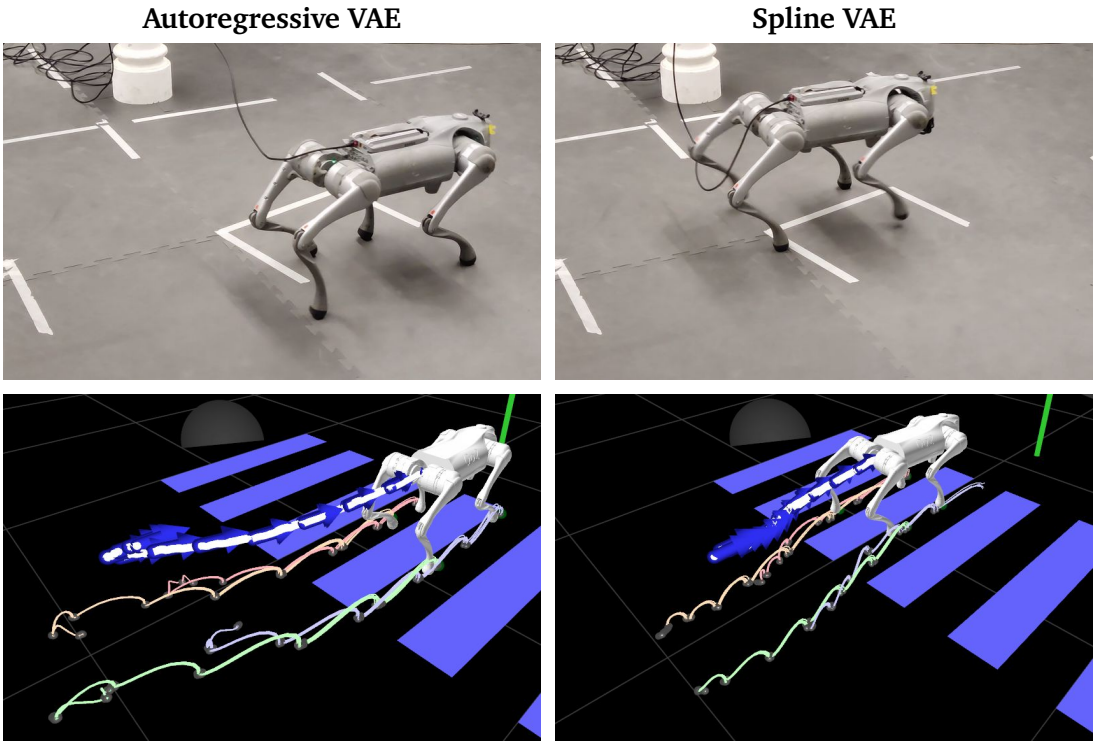


Figure 40: Real-world open-loop trajectory optimization for the *Crosswalk* scenario (6s planning horizon). We show the final time step of on seed. Each column corresponds to one VAE variant. The bottom row shows the final robot state and planned trajectory as 5 samples from the final GMM. Note that the stepping obstacles are only virtual and thus not visible in the real photos.

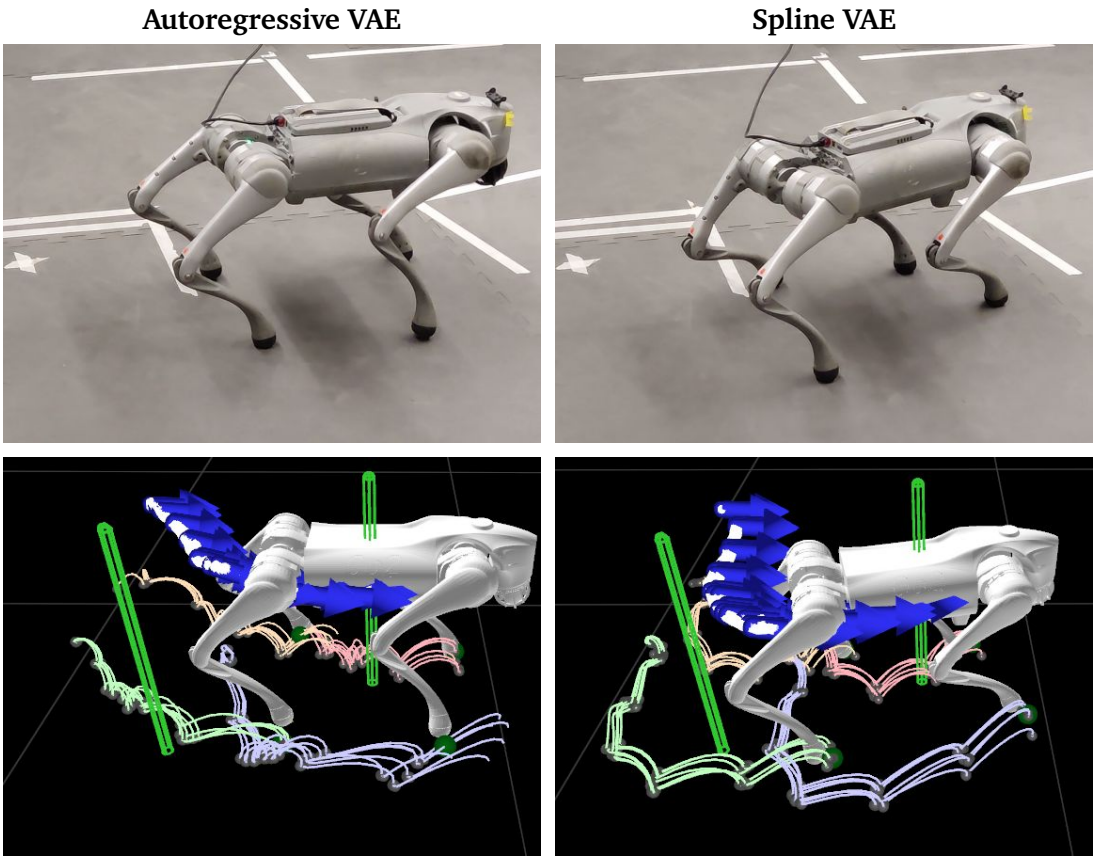


Figure 41: Real-world open-loop trajectory optimization for the *2 Goal* scenario ($6\mathrm{s}$ planning horizon). We show the final time step of one seed. Each column corresponds to one VAE variant. The bottom row shows the final robot state and planned trajectory as 5 samples from the final GMM.

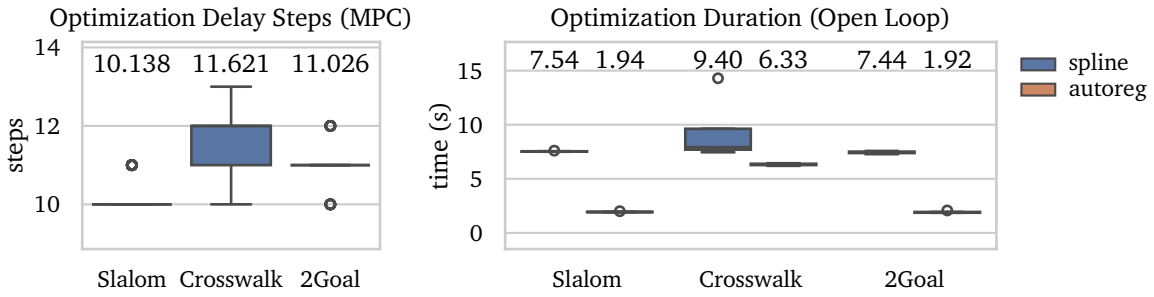


Figure 42: Optimization duration for the real-world experiments. The left Figure shows the optimization delay in policy execution steps (at $50\mathrm{Hz}$) for the MPC execution. In the right plot, the optimization duration for the open-loop execution is displayed (here for each scenario the left box plot corresponds to the spline VAE and the right to the autoregressive VAE). Above each box plot, we show the mean value.

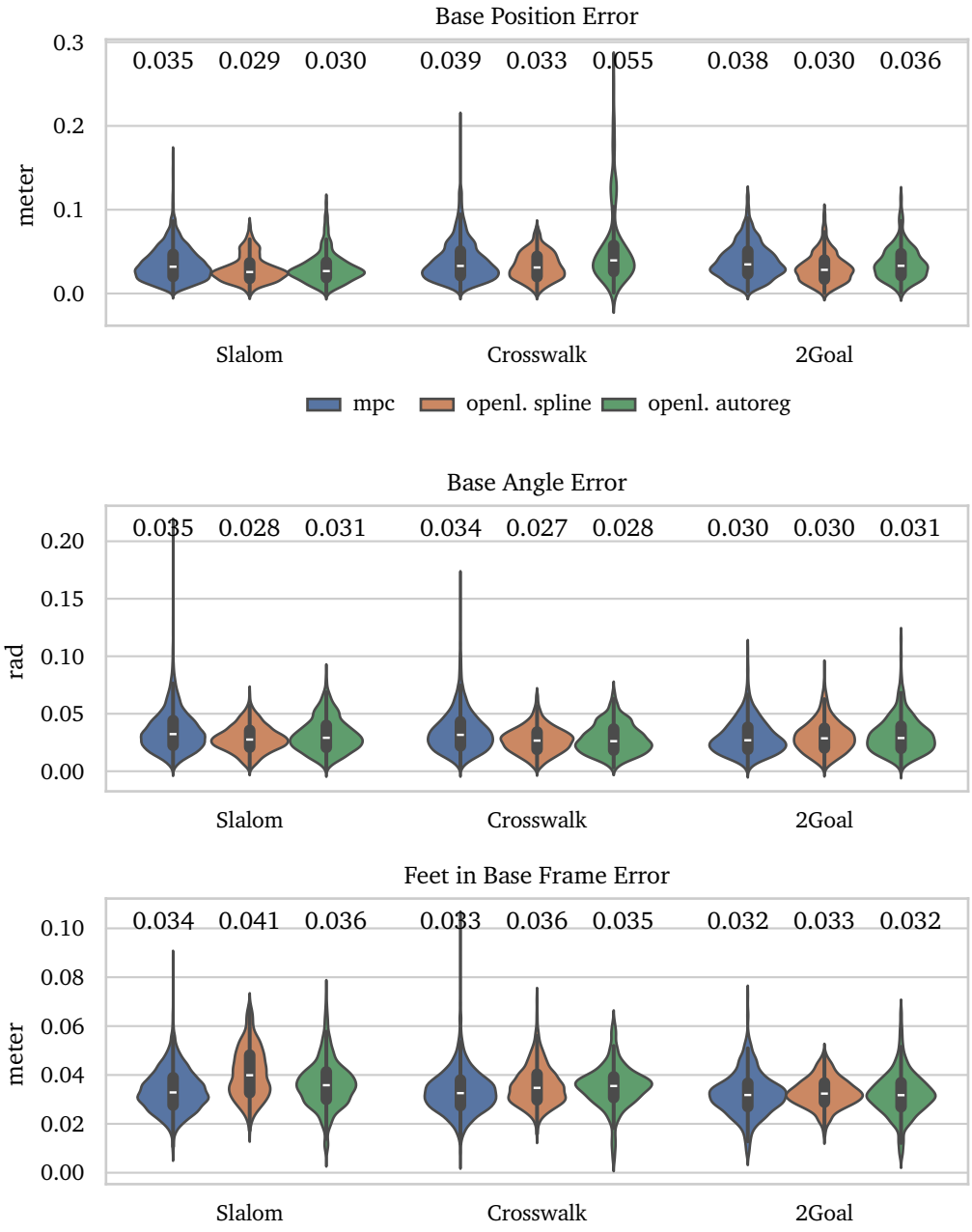


Figure 43: Policy tracking errors for the real-world experiments for all scenarios and deployment variants. Note that for the feet tracking error, we show the mean error over all feet. Above each violin plot we show the mean value.

5 Conclusion

In this work, we presented a framework for task-adaptive imitation-based locomotion planning. We implemented our approach based on VI with GMMs and demonstrated its capabilities with the quadrupedal robot Go2. Two different latent motion representation models were evaluated in combination with our trajectory optimization algorithm. Therefore, we considered a reference trajectory imitation dataset generated from an optimal control locomotion policy. We evaluated our framework in three different task settings. 1) We showed that we can perform goal reaching tasks. 2) Given planar 2D obstacles, obstacle avoidance could be achieved while heading for a goal location. 3) Areas where the robot is not allowed to step onto could be incorporated, leading to valid footstep location planning. Our results show that these tasks can be performed by our framework for open-loop long-horizon planning with both considered motion representation models. Here, the spline VAE is generally more suited for more complex scenarios and longer planning horizons, where its slower optimization performance is similar to the generally faster autoregressive VAE. Our framework also works for closed-loop MPC with shorter planning horizons. Here, only the autoregressive VAE fulfills the runtime requirements. We show successful deployment together with the learned low-level tracking policy in simulation and with a real-world robot for both closed- and open-loop configurations.

When looking at the diversity of different GMM component samples, our approach is capable of considering multiple solutions of a given task. However, the diversity within each GMM component is low, and final distributions only focus on one component. Despite our demonstration of the capabilities for obstacle avoidance, the optimization sometimes produces final trajectories that violate the obstacle collision constraints. Additionally, delays caused by the optimization runtime lead to non-continuous reference trajectory commands forwarded to the low-level policy in the closed-loop setting. This leads to larger tracking errors.

5.1 Outlook

To address some of the current limitations of our approach, we propose the following ideas for future work. Fine-tuning the low-level policy on trajectories produced by the optimization may further improve the tracking performance. This could even happen iteratively, where first, the optimization produces references, which are then incorporated into fine-tuning the policy. Afterward, the actual motions produced by the tracking policy may be added to the dataset, which is subsequently used to fine-tune the motion representation model. Then again, new reference optimizations for given tasks produce new references used to further fine-tune the policy.

To address the low variance of the final GMM samples, new motion representation models may be explored that directly decode latent values to absolute trajectories. Ideally, these models may not be autoregressive and may directly incorporate the start state as conditional input to the decoding process. Additionally, new models may ensure general validity of decoded trajectories, which would reduce the need for high reward weights for inductive bias terms. More efficient

architectures could also improve the optimization runtime performance, enabling lower delays with longer planning horizons in the closed-loop setting.

Our framework could be extended to non-flat terrain scenarios. Here, matching of the terrain height for stepping could be considered as an additional reward term. Additionally, other reference dataset sources, such as recorded real-world data from animals, may be considered for training the motion representation model.

Bibliography

[1] Luo, Z.; Cao, J.; Merel, J.; Winkler, A.; Huang, J.; Kitani, K.; Xu, W. (2023). Universal humanoid motion representations for physics-based control, *Arxiv Preprint Arxiv:2310.04582*

[2] Han, L.; Zhu, Q.; Sheng, J.; Zhang, C.; Li, T.; Zhang, Y.; Zhang, H.; Liu, Y.; Zhou, C.; Zhao, R.; others. (2024). Lifelike agility and play in quadrupedal robots using reinforcement learning and generative pre-trained models, *Nature Machine Intelligence*, Vol. 6, No. 7, 787–798

[3] Tevet, G.; Raab, S.; Cohan, S.; Reda, D.; Luo, Z.; Peng, X. B.; Bermano, A. H.; Panne, M. van de. (2024). Closd: Closing the loop between simulation and diffusion for multi-task character control, *Arxiv Preprint Arxiv:2410.03441*

[4] Mitchell, A. L.; Merkt, W.; Papatheodorou, A.; Havoutis, I.; Posner, I. (2024). Gaitor: Learning a unified representation across gaits for real-world quadruped locomotion, *Arxiv Preprint Arxiv:2405.19452*

[5] Li, Z.; Peng, X. B.; Abbeel, P.; Levine, S.; Berseth, G.; Sreenath, K. (2025). Reinforcement learning for versatile, dynamic, and robust bipedal locomotion control, *The International Journal of Robotics Research*, Vol. 44, No. 5, 840–888

[6] Kingma, D. P.; Welling, M. (2013). Auto-encoding variational bayes, *Arxiv Preprint Arxiv:1312.6114*

[7] Ling, H. Y.; Zinno, F.; Cheng, G.; Van De Panne, M. (2020). Character controllers using motion vaes, *ACM Transactions on Graphics (TOG)*, Vol. 39, No. 4, 40–41

[8] Arenz, O.; Dahlinger, P.; Ye, Z.; Volpp, M.; Neumann, G. (2023). A unified perspective on natural gradient variational inference with gaussian mixture models

[9] Sutton, R. S.; Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (Second.), The MIT Press

[10] Murphy, K. (2024). Reinforcement learning: an overview, *Arxiv Preprint Arxiv:2412.05265*

[11] Rummery, G. A.; Niranjan, M. (1994). *On-Line Q-Learning Using Connectionist Systems* (Vol. 37), University of Cambridge, Department of Engineering Cambridge, UK

[12] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. (2013). Playing atari with deep reinforcement learning, *Arxiv Preprint Arxiv:1312.5602*

[13] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Reinforcement Learning*, 5–32. doi:10.1007/978-1-4615-3618-5_2

[14] Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. (2015). Trust region policy optimization, *International Conference on Machine Learning*, 1889–1897

[15] Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. (2017). Proximal policy optimization algorithms, *Arxiv Preprint Arxiv:1707.06347*

[16] Higgins, I.; Matthey, L.; Pal, A.; Burgess, C.; Glorot, X.; Botvinick, M.; Mohamed, S.; Lerchner, A. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework, *International Conference on Learning Representations*

[17] Burgess, C. P.; Higgins, I.; Pal, A.; Matthey, L.; Watters, N.; Desjardins, G.; Lerchner, A. (2018). Understanding disentangling in $b\eta$-VAE

[18] Blessing, D.; Jia, X.; Esslinger, J.; Vargas, F.; Neumann, G. (2024). Beyond ELBOs: A large-scale evaluation of variational methods for sampling, *Arxiv Preprint Arxiv:2406.07423*

[19] Arenz, O.; Neumann, G.; Zhong, M. (2018). Efficient Gradient-Free Variational Inference using Policy Search, J. Dy; A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning* (Vol. 80), PMLR, 234–243

[20] Lin, W.; Schmidt, M.; Khan, M. E. (2020). Handling the positive-definite constraint in the Bayesian learning rule, *International Conference on Machine Learning*, 6116–6126

[21] Zhang, H.; Starke, S.; Komura, T.; Saito, J. (2018). Mode-adaptive neural networks for quadruped motion control, *ACM Transactions on Graphics (Tog)*, Vol. 37, No. 4, 1–11

[22] Starke, S.; Zhang, H.; Komura, T.; Saito, J. (2019). Neural state machine for character-scene interactions, *ACM Transactions on Graphics*, Vol. 38, No. 6, 178

[23] Kim, J.; Li, T.; Ha, S. (2023). Armp: autoregressive motion planning for quadruped locomotion and navigation in complex indoor environments, *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2731–2737

[24] Lucas, T.; Baradel, F.; Weinzaepfel, P.; Rogez, G. (2022). Posegpt: Quantization-based 3d human motion generation and forecasting, *European Conference on Computer Vision*, 417–435

[25] Starke, S.; Mason, I.; Komura, T. (2022). Deepphase: Periodic autoencoders for learning motion phase manifolds, *ACM Transactions on Graphics (Tog)*, Vol. 41, No. 4, 1–13

[26] Starke, P.; Starke, S.; Komura, T.; Steinicke, F. (2023). Motion in-betweening with phase manifolds, *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, Vol. 6, No. 3, 1–17

[27] Li, C.; Stanger-Jones, E.; Heim, S.; Kim, S. (2024). FLD: Fourier latent dynamics for structured motion representation and learning, *Arxiv Preprint Arxiv:2402.13820*

[28] Watanabe, R.; Li, C.; Hutter, M. (2025). DFM: Deep Fourier Mimic for Expressive Dance Motion Learning, *Arxiv Preprint Arxiv:2502.10980*

[29] Mitchell, A. L.; Merkt, W. X.; Geisert, M.; Gangapurwala, S.; Engelcke, M.; Jones, O. P.; Havoutis, I.; Posner, I. (2023). Vae-loco: Versatile quadruped locomotion by learning a disentangled gait representation, *IEEE Transactions on Robotics*, Vol. 39, No. 5, 3805–3820

[30] Peng, X. B.; Abbeel, P.; Levine, S.; Panne, M. Van de. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills, *ACM Transactions on Graphics (TOG)*, Vol. 37, No. 4, 1–14

[31] Bin Peng, X.; Coumans, E.; Zhang, T.; Lee, T.-W.; Tan, J.; Levine, S. (2020). Learning agile robotic locomotion skills by imitating animals, *Robotics: Science and Systems (RSS), Virtual Event/corvalis, July*, 12–16

[32] Peng, X. B.; Ma, Z.; Abbeel, P.; Levine, S.; Kanazawa, A. (2021). Amp: Adversarial motion priors for stylized physics-based character control, *ACM Transactions on Graphics (Tog)*, Vol. 40, No. 4, 1–20

[33] Wu, J.; Xin, G.; Qi, C.; Xue, Y. (2023). Learning robust and agile legged locomotion using adversarial motion priors, *IEEE Robotics and Automation Letters*, Vol. 8, No. 8, 4975–4982. doi:10.1109/LRA.2023.3290509

[34] Castillo, G. A.; Weng, B.; Zhang, W.; Hereid, A. (2024). Data-Driven Latent Space Representation for Robust Bipedal Locomotion Learning, *2024 IEEE International Conference on Robotics and Automation (ICRA)* (Vol. 0), 1172–1178. doi:10.1109/ICRA57147.2024.10610978

[35] Stark, F.; Middelberg, J.; Mronga, D.; Vyas, S.; Kirchner, F. (2025). Benchmarking Different QP Formulations and Solvers for Dynamic Quadrupedal Walking

[36] Tedrake, R.; Drake Development Team. (2019). Drake: Model-based design and verification for robotics

[37] Zakka, K.; Tabanpour, B.; Liao, Q.; Haiderbhai, M.; Holt, S.; Luo, J. Y.; Allshire, A.; Frey, E.; Sreenath, K.; Kahrs, L. A.; Sferrazza, C.; Tassa, Y.; Abbeel, P. (2025). MuJoCo Playground

[38] Todorov, E.; Erez, T.; Tassa, Y. (2012). MuJoCo: A physics engine for model-based control, *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. doi:10.1109/IROS.2012.6386109

[39] Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs

[40] Freeman, C. D.; Frey, E.; Raichuk, A.; Girgin, S.; Mordatch, I.; Bachem, O. (2021). Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation

[41] OptiTrack - Motion Capture Systems. www.optitrack.com

A. Appendix

Parameter	Value
Sample base linear xy-velocity command ($\frac{m}{s}$)	$U(-0.35, 0.35)$
Sample base angular z-velocity command ($\frac{\text{rad}}{s}$)	$U(-0.8, 0.8)$
Sample base z-height command (m)	$U(0.28, 0.32)$
Fixed base xy-angles (rad)	0
Propabilty to sample new command at time step	$0.06 \triangleq 0.6\frac{1}{s}$
Number of interpolation steps from old to new command	$13 \triangleq 1.3s$
Number steps before starting to sample new a command	$14 \triangleq 1.4s$
Command resample loop frequency	10Hz

Table A.1: Go2 reference trajectory dataset generation parameters.

Parameter	Value
Encoded time steps N	20
Context length l_{ctx}	2
Latent dimensions D_z	16
Autoregressive training rollout length M_{rollouts}	8
Initial learning rate	10^{-4}
Final learning rate	10^{-7}
Loss KL-term weight β	0.015
Loss contact reconstruction weight α_{contact}	0.05
Training phases fractions for $[p_{\text{ctx,autoreg}} = 0, p_{\text{ctx,autoreg}} = \text{inc}, p_{\text{ctx,autoreg}} = 1]$	$[15\%, 25\%, 60\%]$
Normalization	z-score
Update batch size	1024
Training Epochs	400

Table A.2: Autoregressive VAE training parameters for the quadruped dataset.

Fully Connected - 1024
ELU
Fully Connected - 256
ELU
Fully Connected - 128
ELU
Fully Connected - 64
ELU
Fully Connected - $16 \cdot 2$
ELU
Fully Connected - $16 \cdot 2$ $\mu, \log(\sigma^2)$ output

(a) Encoder

Fully Connected - 64
ELU
Fully Connected - 128
ELU
Fully Connected - 256
ELU
Fully Connected - 1024
ELU
Fully Connected - 1024
ELU
Fully Connected - 440 delta $\hat{\boldsymbol{o}}_{t=0,\dots,20}$ output

(b) Decoder

Figure A.1: Autoregressive VAE model structure. For each layer we denote the output size.

Parameter	Value
Encoded time steps N	25
Latent dimensions D_z	10
Learning rate	$2.5 \cdot 10^{-4}$
Loss KL-term weight β	0.15
Loss contact reconstruction weight α_{contact}	0.1
Loss spline reconstruct weight $\alpha_{\text{recon,spline}}$	1
Spline basis knot points per reference duration	$20/15s$
Polar min radius $r_{\min}$	0.01
Spline regression regularization weight λ_{reg}	0.01
Sampled train reference slice time steps $l_{\text{stepstrain}}$	100
Normalization	MinMax
Update batch size	16
Training Epochs	$14 \cdot 10^3$

Table A.3: Spline VAE training parameters for the quadruped dataset.

Fully Connected - 256
relu
Fully Connected - 256
relu
Fully Connected - 64
relu
Fully Connected - 32
relu
Fully Connected - $10 \cdot 2$ $\mu, \log(\sigma^2)$ output

(a) Encoder

Fully Connected - 32
ELU
Fully Connected - 64
ELU
Fully Connected - 256
ELU
Fully Connected - 256
ELU
Fully Connected - 450 $\hat{\boldsymbol{o}}_{\text{no contact, }(t=0,\ldots,25)}$ output

(b) Decoder

Fully Connected - 32
ELU
Fully Connected - 64
ELU
Fully Connected - 256
ELU
Fully Connected - 256
ELU
Fully Connected - 100 $\hat{\boldsymbol{c}}_{t=0,\ldots,25}$ output

(c) Contact Decoder

Figure A.2: Spline VAE model structure. For each layer we denote the output size.

	Observation	Dimensions
Proprioception	Base-frame angular velocity (noisy)	3
	Joint Angles relative to default pose (noisy)	12
	Joint Velocities (noisy)	12
	Gravity Vector (noisy)	3
Reference Cmd. for $(t+1, t+2, t+3)$	Target base-frame angular velocity	$3 \cdot 3$
	Target z-height	$1 \cdot 3$
	Target z-height error	$1 \cdot 3$
	Target base-frame rotation error vector (as rad)	$3 \cdot 3$
	Target base-frame xy-position error (clipped to length 1)	$2 \cdot 3$
	Target feet contact state	$4 \cdot 3$
	Target feet positions in base-frame	$3 \cdot 4 \cdot 3$
	Last Action	12

(a) Policy observations.

	Observation	Dimensions
Proprioception	Base-frame angular velocity	3
	Joint Angles relative to default pose	12
	Joint Velocities	12
	Gravity Vector	3
	Base-frame linear velocity	3
	Base-frame linear acceleration	3
	Actuator torques	12
	Global velocities for each foot	$3 \cdot 4$
	Feet contact states	$1 \cdot 4$
	Push force applied to the base-frame	3
	Time steps since last push $\geq$ time steps to next push	1
	Feet air time	$1 \cdot 4$

(b) Privileged observation that the value function receives in addition to the policy observations.

Table A.4: Tracking policy observations for policy and value function.

Parameter	Value
Ground friction	$U(0.3, 1.0)$
Joint friction loss (scaling of default)	$U(0.8, 1.2)$
Joint armature inertia (scaling of default)	$U(1.0, 1.05)$
Base-frame body center of mass position (xyz offset to default)	$U(-0.05, 0.05)$
Base-frame body mass (offset to default)	$U(-1.0, 1.0)$
All links body mass (scaling of default)	$U(0.9, 1.1)$
Joint zero angle offset (offset to default)	$U(-0.05, 0.05)$
Joint stiffness (scaling of default)	$U(0.8, 1.2)$
PD-controller k-gain (scaling of default)	$U(0.9, 1.1)$
PD-controller d-gain (scaling of default)	$U(0.8, 1.7)$
Gravity value	$U(-8.81, -10.81)$
Probability action delay $p_{\text{a\_delay}}$	0.05

Table A.5: Tracking policy domain randomization parameters.

Parameter	Value
Base-frame angular velocity	$U(-0.2, 0.2)$
Joint Angles	$U(-0.05, 0.05)$
Joint Velocities	$U(-2, 2)$
Gravity Vector	$U(-0.05, 0.05)$

Table A.6: Tracking policy observation noise parameters. The sampled noise values are added to the observations.

	Parameter	Value
Controller	P-Gain	45
	D-Gain	0.5
Base Pushes	Push magnitude $v_{\text{push,max}}$ ($\frac{m}{s}$)	$U(0.0, 1.5)$
	Push angle around z-axis (rad)	$U(0.0, 2\pi)$
	Push duration t_{push} (s)	$U(0.05, 0.2)$
	Waiting time between pushes (s)	$U(1.0, 3.0)$
Reference Jumps	Wait time between jums (s)	$U(0.4, 1.2)$
	Jump in time length (s)	$U(-0.2, 0.2)$
Start State	Base-frame initial position xy (offset to default, m)	$U(-0.02, 0.02)$
	Base-frame initial rotation z (offset to default, °deg)	$U(-8, 8)$
	Base-frame initial linear velocity (offset to default, $\frac{m}{s}$)	$U(-0.1, 0.1)$
	Base-frame initial angular velocity (offset to default, $\frac{\text{rad}}{s}$)	$U(-0.2, 0.2)$

Table A.7: Tracking policy enviroment paramters.

	Reward Term	Weight
Tracking	$r^{\text{track base pos xy}}$	5
	$r^{\text{track base pos z}}$	0.5
	$r^{\text{track base linvel}}$	0.5
	$r^{\text{track base rot}}$	2
	$r^{\text{track base angvel}}$	0.5
	$r^{\text{track feet pos}}$	1
	$r^{\text{track feet contact}}$	1
Regularization	$r^{\text{q limit}}$	1
	$r^{\text{q default}}$	0.5
	$r^{\text{q vel}}$	0.5
	$r^{\text{q acc}}$	0.02
	r^{torque}	0.0002
	$r^{\text{action rate}}$	0.05
	r^{energy}	0.001
	$r^{\text{feet slip}}$	0.6
	$r^{\text{base angvel}}$	0.05
	$r^{\text{base rot}}$	5
	$r^{\text{terminate}}$	1
	r^{survival}	5

Table A.8: Tracking policy reward weights.

Fully Connected - 32
Layer norm
swish
Fully Connected - 32
swish
Fully Connected - 32
swish
Fully Connected - 32
swish
Fully Connected - $2 \cdot 12$

(a) Policy. The output is a Gaussian distribution over actions (mean and variance per dimension)

Fully Connected - 256
Layer norm
swish
Fully Connected - 256
swish
Fully Connected - 256
swish
Fully Connected - 256
swish
Fully Connected - 256
swish
Fully Connected - 1

(b) Value function

Figure A.3: Policy and value function NN structure. For each layer we denote the output size.

Parameter	Value
Total environment steps	$0.8 \cdot 10^9$
Discount factor γ	0.97
Normalize observations	yes
Number of minibatches	32
Update batch size	256
Learning rate	$3 \cdot 10^{-4}$
Entropy cost	10^{-2}
Number parallel environments	4096
Maximum gradient norm	1.0
GAE lambda	0.95
Policy loss clipping ε	0.3
Episode length (steps)	$750 \triangleq 15s$

Table A.9: Tracking policy training PPO parameters.

Parameter	Value
$s_{\text{contact max total}}$	0.01
$s_{\text{slip max step}}$ $(\frac{m}{s})$	$1 \cdot 10^{-4}$
$s_{\text{slip max total}}$	$5 \cdot 10^{-4}$
$s_{\text{footz max step}}$ (m)	0.02
$s_{\text{footz max total}}$	$1 \cdot 10^{-6}$
$z_{\text{max autoreg vae}}$	3
$z_{\text{max spline vae}}$	3.5
δ_{lin}	0.4
$g_{\text{end steps frac}}$	0.2

Table A.10: Default reference trajectory optimization reward hyperparameters.

Parameter	Value
$\boldsymbol{s}_{\text{col base xy rel}}$ (m)	$\begin{bmatrix} -0.25 & -0.16 \\ 0.05 & -0.16 \\ 0.35 & -0.16 \\ -0.25 & 0.16 \\ 0.05 & 0.16 \\ 0.35 & 0.16 \\ -0.25 & 0 \\ 0.35 & 0 \end{bmatrix}$
$\boldsymbol{s}_{\text{col feet xy rel}}$ (m)	$\begin{bmatrix} -0.175 \\ 0.0 \end{bmatrix}$
$s_{\text{obs circle tol}}$ (m)	0.1
$\boldsymbol{s}_{\text{obs rect ells num}}$	3
$\boldsymbol{s}_{\text{obs rect ells expos}}$	$(2, 4, 6)^T$
$\boldsymbol{s}_{\text{obs rect ells weights}}$	$(1, 1, 3)^T$
$s_{\text{obs rect tol}}$	0.25
$\boldsymbol{s}_{\text{step obs rect ells num}}$	2
$\boldsymbol{s}_{\text{step obs rect ells expos}}$	$(2, 4)^T$
$\boldsymbol{s}_{\text{step obs rect ells weights}}$	$(1, 1)^T$
$s_{\text{step obs rect tol}}$	0.15

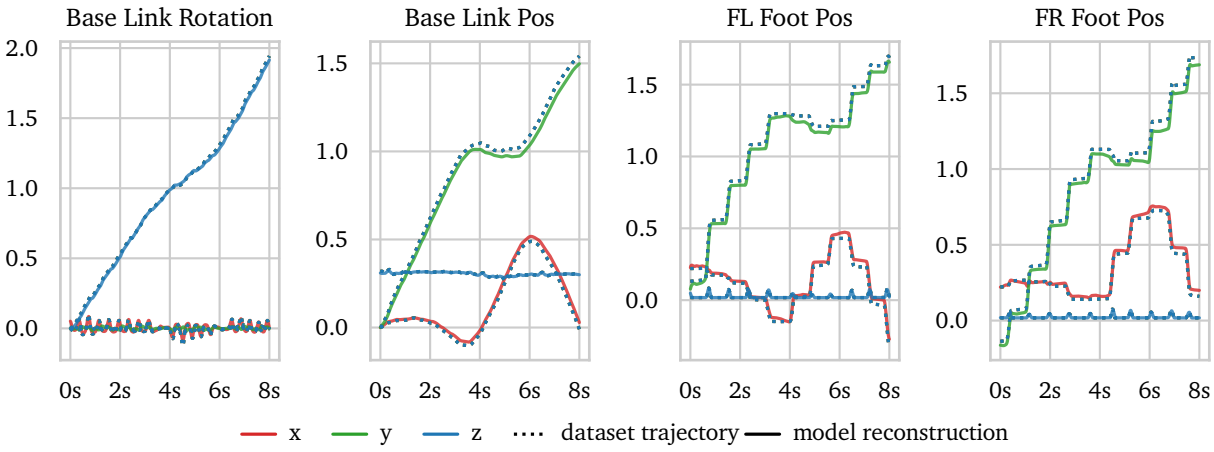
Table A.11: Default reference trajectory optimization obstacle avoidance parameters.

		Reward Term	Weight
Base Task		$r^{\text{z in range}}$	10000
	Ind. Bias	$r^{\text{indbias cont}}$	9
		$r^{\text{indbias slip}}$	9000000
		$r^{\text{indbias footz}}$	90000
	Reg.	$r^{\text{reg base angular z vel}}$	10
		$r^{\text{reg base linear xy vel}}$	10
Spline VAE		$r^{\text{spline dphase in range}}$	10000
		$r^{\text{spline start rel}}$	50000
		$r^{\text{spline start abs}}$	500000
Goal		$r^{\text{goal pos end}}$	10000
		$r^{\text{goal pos mid}}$	10000
		$r^{\text{goal pos end steps}}$	1000
Obstacles		$r^{\text{obs avoid feet}}$	10000
		$r^{\text{obs avoid additional}}$	20000
		$r^{\text{obs avoid base}}$	10000
		$r^{\text{step obs avoid feet}}$	13000

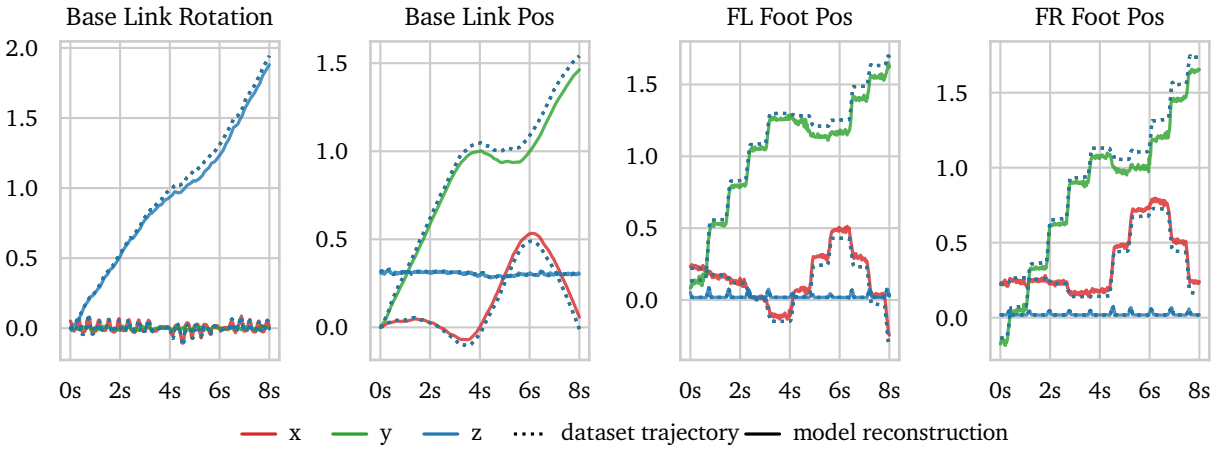
Table A.12: Default reference trajectory optimization reward weights.

Parameter	Value
GMM init standard deviation	2.0
GMM init prior mean	0.0
GMM init prior scale	1.0
GMM covariance matrix	diagonal
GMM number of components	5
Samples per component	50

Table A.13: Default reference trajectory optimization Algorithm 2 parameters.



(a) With appling the filtering described in Equation 27 with $\alpha_{\text{filter}} = 0.05$.



(b) Without appling the filtering, but only taking the first decoder output time step of each reconstruction.

Figure A.4: Reconstruction when passing one dataset trajecory through our spline VAE. Solid lines corresond to the reconstuction model output while dashed lines show the true dataset trajectory. We show the absolute representation, where the y-axis corresonds to radians for the rotation plot and to meters for all other plots.

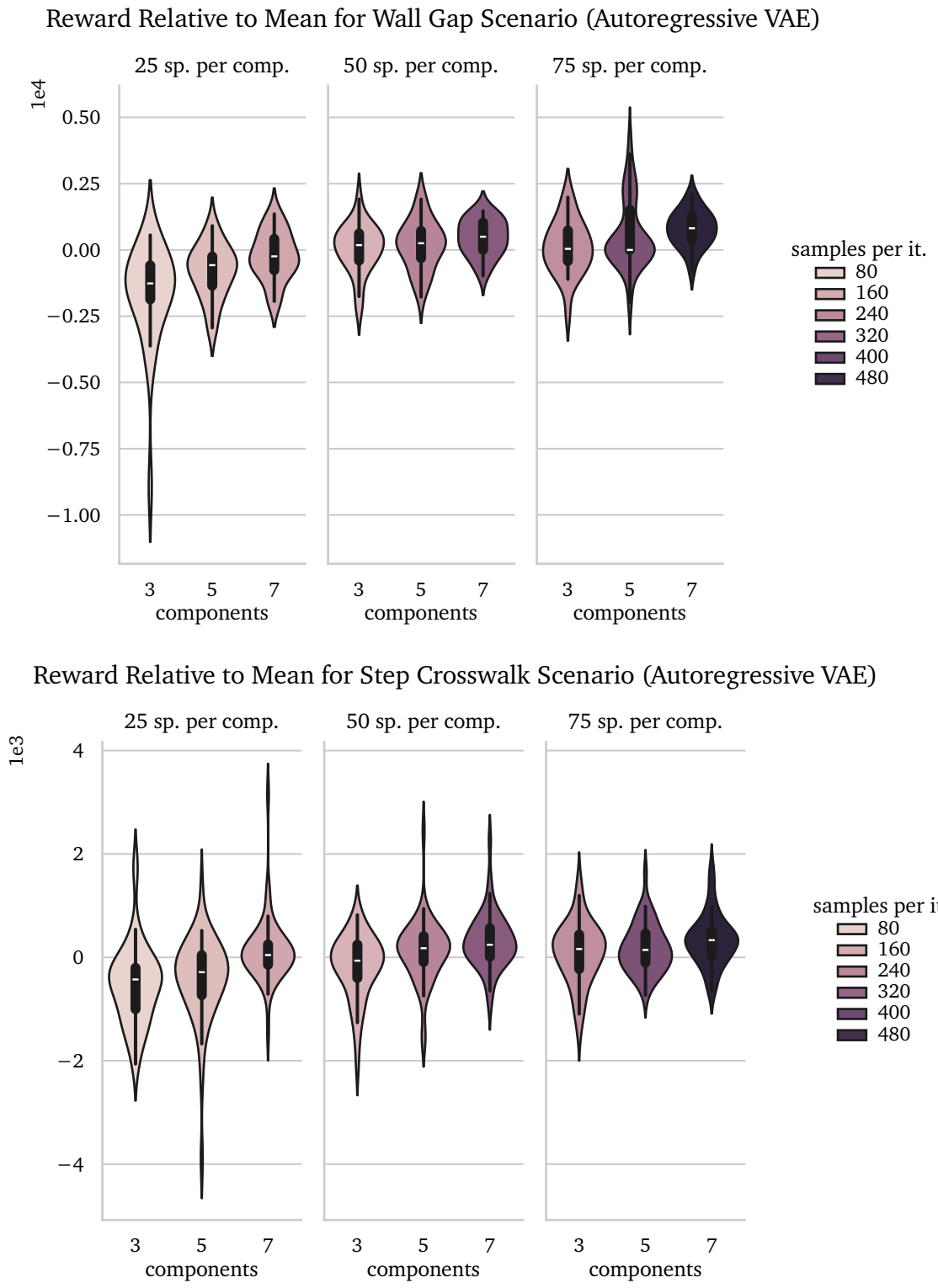


Figure A.5: GMM optimization parameter variation with the autoregressive VAE for the $6s$ horizon. We show the final reward (including all reward terms) of the optimization across different parameterizations relative to the mean reward of the scenario (the mean reward over all parameterizations). The color of each violin plot indicates the total number of samples per iteration resulting from the chosen number of components and samples per component.

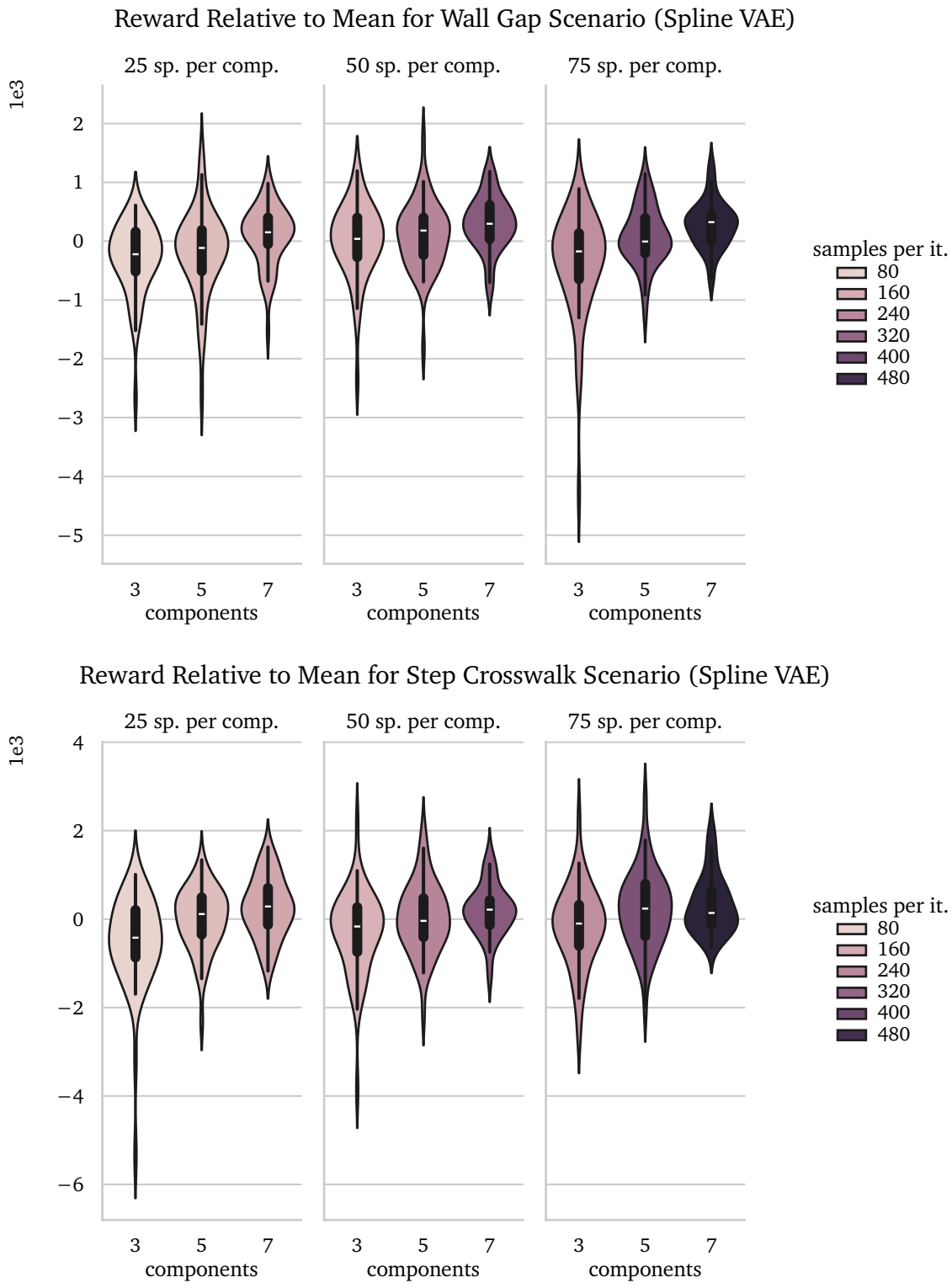


Figure A.6: GMM optimization parameter variation with the Spline VAE for the $6s$ horizon. We show the final reward (including all reward terms) of the optimization across different parameterizations relative to the mean reward of the scenario (the mean reward over all parameterizations). The color of each violin plot indicates the total number of samples per iteration resulting from the chosen number of components and samples per component.

Reward	Weight
$r^{\text{obs avoid additional}}$	10000
$r^{\text{step obs avoid feet}}$	8000
$r^{\text{indbias slip}}$	7200000
$r^{\text{goal pos end}}$	9600
$r^{\text{goal pos mid}}$	0
$r^{\text{goal pos end steps}}$	1200

Table A.14: Autoregressive VAE reward weight adaptations for the MPC deployment.