

High-Dimensional Controller Tuning through Latent Representations

Alireza Sarmadi¹, Prashanth Krishnamurthy¹, and Farshad Khorrami¹

Abstract—In this paper, we propose a method to automatically and efficiently tune high-dimensional vectors of controller parameters. The proposed method first learns a mapping from the high-dimensional controller parameter space to a lower dimensional space using a machine learning-based algorithm. This mapping is then utilized in an actor-critic framework using Bayesian optimization (BO). The proposed approach is applicable to complex systems (such as quadruped robots). In addition, the proposed approach also enables efficient generalization to different control tasks while also reducing the number of evaluations required while tuning the controller parameters. We evaluate our method on a legged locomotion application. We show the efficacy of the algorithm in tuning the high-dimensional controller parameters and also reducing the number of evaluations required for the tuning. Moreover, it is shown that the method is successful in generalizing to new tasks and is also transferable to other robot dynamics.

I. INTRODUCTION

Control design for nonlinear dynamical systems in complex environments poses a challenging problem since the control parameter space is high-dimensional. From several solutions proposed in the literature, three approaches in particular have gained significant attention. Some of the applications have been in the robotics context for legged locomotion in varying terrains and these three approaches have been proposed to overcome the high-dimensionality curse. The first approach is the heuristic-based control in which an expert with a deep knowledge of the system designs the algorithm [1]–[4] based on some heuristic. These algorithms suffer since it is difficult to know all conditions encountered during system operation.

The second approach is learning-based utilizing data-driven control policies. More specifically, Deep Reinforcement Learning (DRL) [5] methods learn a controller without any prior knowledge about the system dynamics by training an end-to-end controller (i.e., the input to the controller is the observed state of the system, the output is the actuation vector). These learning-based methods typically need large numbers of samples (e.g., tens of thousands [6]) to achieve a reasonable performance. However, collecting large amounts of data on real-world systems is challenging due to the physical wear and tear and safety issues in addition to the time required. Also, these algorithms can experience

stability/performance degradations when adapting to new tasks [7] and are very sensitive to hyperparameters [8].

The third approach is optimization-based control where the problem is formulated into an optimization problem. Model Predictive Control (MPC) is a well-known online framework for controlling various systems under uncertainties [9]–[14]. The performance of the MPC framework is very sensitive to cost function, constraints, and system dynamics. These hyperparameters should be tuned carefully to achieve the desired behavior. Usually, these hyperparameters are tuned manually or using grid search, which require many trial and error system evaluations and might not find the optimal set of hyperparameters. However, performing large numbers of trials on systems cause physical wear/tear and safety issues.

One widely used solution for tuning hyperparameters with smaller numbers of system evaluations is Bayesian Optimization (BO) [15]. BO is a black box optimization method to find the optimum of an unknown function when the function evaluation is expensive. BO has two main components: 1) a surrogate model for the objective function (usually Gaussian Process), and 2) an acquisition function to pick the next sample. One role of the acquisition function is to trade off exploration against exploitation. This makes BO a very interesting tool that can be used to adapt the policies designed in simulation with a few real-world experiments. However, BO's inference time grows cubic in the number of observations and exponentially with the dimension of the search space, therefore limiting its application in high-dimensional systems. Also, a high dimensional space results in an often heterogeneous function making the task of fitting a global surrogate model challenging. Due to these reasons, BO is typically practical for dimensions about 10-20.

Our proposed method utilizes an off-line simulation environment to learn an underlying mapping of the high-dimensional controller parameter space to a lower dimensional space. This is achieved by utilizing a Variational Auto Encoder (VAE) structure [16] to efficiently tune the high-dimensional controller parameters. To generate training data for learning the mapping, a sample-efficient BO-based search is utilized in an actor-critic framework. The encoder part of the trained VAE then acts as the desired mapping reducing the dimensionality of the search space for the controller parameters. With this learned mapping, the sample-efficient BO-based method is applied in combination with the VAE decoder to tune the parameters on the system with a significantly smaller number of system evaluations compared to if the BO-based search was applied directly to the system with the original high-dimensional controller parameter space. The proposed method is applied to quadrupeds.

¹The authors are with the Control/Robotics Research Laboratory (CRRL), Department of Electrical and Computer Engineering, NYU Tandon School of Engineering, Brooklyn NY, 11201. E-mail: {as11986@nyu.edu, prashanth.krishnamurthy@nyu.edu, khorrami@nyu.edu}. This work was supported in part by ARO under Grant W911NF-21-1-0155 and Grant W911NF-22-1-0028, and in part by the New York University Abu Dhabi (NYUAD) Center for Artificial Intelligence and Robotics (CAIR), funded by Tamkeen under the NYUAD Research Institute Award CG010.

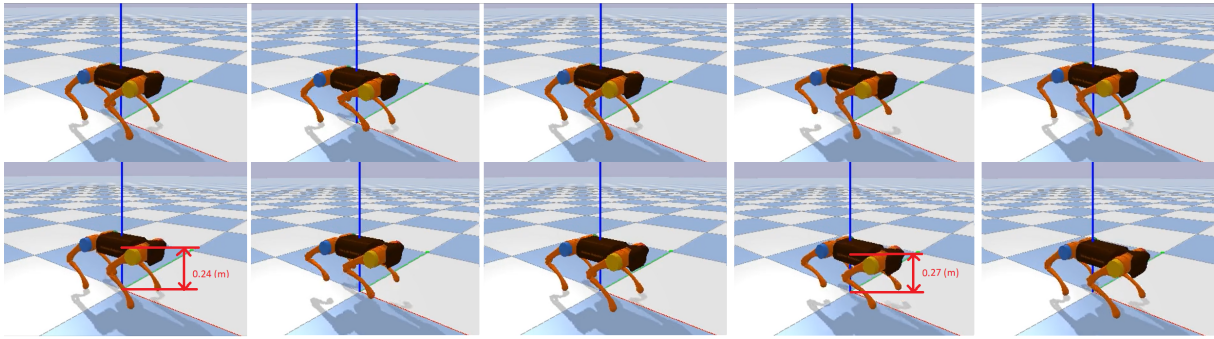


Fig. 1. Unitree A1 trotting (top row) and jumping (bottom row) motions in the PyBullet simulation environment. The maximum and minimum height of the robot during the jumping are shown.

II. RELATED WORKS

Performance of learning-based MPC has been improved recently [17]. Additionally, data-driven algorithms have been proposed to alleviate this problem by improving the system’s model [18]–[21]. Another trend in learning-based MPC is to learn the design of the controller (e.g., cost function and constraints). [22] proposed automatic LQR tuning using BO to tune the cost function parameters. In [23], the linear model of a system is learned using BO. In [24], [25], the authors used BO to tune cost function parameters of the trajectory optimization problem. An Inverse Optimal Control (IOC) algorithm was proposed in [26] that learns hyperparameters of a defined cost function from human demonstration data for path tracking control problems. In [27], the authors used BO for a jumping quadruped with motor current constraints. [28] utilized first-order information based acquisition functions to improve gradient estimates for a black-box reward function using Gaussian Process model gradients. [29] proposed a method for solving MPC where a neural network derives cost function parameters from sensory data, employing an encoder for sensory information dimensionality reduction.

Utilizing evolutionary algorithms [30] is another approach for solving the optimization problem for high-dimensional spaces when the gradient of the cost function is not available (i.e., black-box function). However, these methods require a lot more samples. One of the fundamental work is Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [31] that forms a parametric distribution over the solution space. Then, it chooses candidates to be evaluated by a black-box function from a parameterized search distribution. The first few best candidates are selected to update the parametric distribution. Other papers in the literature use simulation to speed up BO. In [32], trajectories are generated during simulations to build a feature transformation. Then, instead of passing the samples to the kernel function, they pass the transformed controller parameters.

To implement the BO component in our work, we use Trust Region BO (TuRBO) [33] instead of the standard BO that is only suitable for low-dimensional problems. This method uses a collection of simultaneous local optimization runs using independent probabilistic models to model the

whole function by a combination of local surrogate models. These local surrogates allow for heterogeneous modeling of the objective function and do not suffer from over-exploration. To optimize globally, an implicit multi-armed bandit strategy is leveraged at each iteration to allocate samples between these local areas to determine which local optimization runs to continue.

III. METHOD

We consider the general problem of tuning a vector of controller parameters corresponding to a parameterized controller for an agent interacting with an environment. For example, the set of parameters could include parameters in a cost function for an MPC controller. The objective is to minimize the cumulative cost of performing a control task. For brevity, we will simply use the term “parameters” below to refer to the vector of controller parameters. The agent interacts with the environment via the action $u_i \in A$ generated by the policy $\pi_\theta(x_i)$ in which $x_i \in S$, θ is the parameter vector, S is the set of all states, and A is the set of all the actions. The agent seeks to find the optimal θ by solving the optimization problem:

$$\theta^* = \arg \min_{\theta} L^{\pi_\theta} \quad (1)$$

where the path cost of the policy (i.e., L^{π_θ}) is defined as

$$L^{\pi_\theta}(O^{des}, O) = \sum_{i=0}^{N_H} q(O_i^{des}, O) \quad (2)$$

with $N_H \in [0, +\infty)$ being the termination horizon of the problem, $q(\cdot, \cdot) : S \times S \rightarrow \mathbb{R}$ the running cost, O a task-specific vector of observed quantities from the environment that quantify performance, and O_i^{des} the desired task behaviour in terms of the observed variables. For example, in the quadruped application, O could be the velocity of the center of mass (CoM) while O_{des} would be the desired value of the CoM velocity.

Our method approaches this problem in a gradient-free actor-critic manner in which the path cost is used to measure the performance of an MPC as an actor. A set of parameters is suggested by the critic. Then, the MPC actor generates the policy by solving Eq. 6. The interaction between actor and

critic is depicted in Fig. 2. In the following, the critic and the actor are described in more detail.

Critic: The critic in our method evaluates the actor’s performance using the path cost defined in Eq. 2. The actor rolls out the policy and collects the (state, action) pairs to compute the path cost along the trajectory. Then, the critic suggests another θ in order to solve Eq. 1 after some iterations. Note that Eq. 1 can not be solved using gradient based methods, since we do not have access to the gradient of the L^{π_θ} w.r.t. to θ . While a black-box optimization method such as BO would potentially be applicable, BO by itself is not suitable for high-dimensional parameter spaces as discussed in Sec. I. To overcome this limitation, TuRBO [33] is utilized in which multiple independent local surrogate models represent the path cost instead of only one surrogate model. For each local model, a Gaussian Process (GP) is utilized to construct the local surrogate model

$$L_l^{\pi_\theta} \sim GP_l(\mu_l(\theta), K_l(\theta, \theta)) \quad (3)$$

where $\mu_l(\cdot)$ is the prior mean, and $K_l(\cdot, \cdot)$ corresponds to the kernel matrix ($K_l \in \mathbb{R}^{n \times n}$; where n is the length of $\mu_l(\cdot)$) of the l^{th} GP model ($l \in \{1, \dots, m\}$ where m is the number of the regions, i.e., the number of local models). The element at the r^{th} row and t^{th} column of the kernel matrix is the Matern kernel function [34] between θ_r and θ_t defined as

$$k_{Matren}(\theta_r, \theta_t) = \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2\nu d}) \kappa_\nu(\sqrt{2\nu d}) \quad (4)$$

where $d = (\theta_r - \theta_t)^T \Theta^{-2} (\theta_r - \theta_t)$ with length scale parameter Θ , ν is a smoothness parameter, Γ is the gamma function, and κ_ν is the modified Bessel function. The local surrogate models are optimized on a Trust Region (TR) centered at the best solution. The assumption here is that the local model can accurately model the function in the corresponding region. TuRBO considers hyperrectangles for the trust regions with an initial set of side lengths. During the optimization, these side lengths are rescaled based on the improvement. At each iteration, a set of candidates are chosen from the union of all the trust regions, then the i^{th} candidate is chosen in a way that minimizes the function value across all the trust regions as

$$s_t = \arg \min_s \arg \min_{\theta \in TR_s} GP_s^t(\mu_s(\theta), k_s(\theta, \theta')) \quad (5)$$

where s_t is the optimal trust region at iteration t , μ_s is the mean vector, k_s is the kernel matrix for trust region s , and GP_s^t is the GP model of the trust region s at iteration t . The critic improves the GP model by obtaining more samples that are suggested by an acquisition function trading off the exploration vs. exploitation.

Actor: The actor is the trajectory optimizer that solves an

optimal control problem in an MPC framework [35]

$$\begin{aligned} \pi_\theta = \min_{u_k, \dots, u_{k+N_1}} & J_T(x_{N_1}, u_{N_1}) + \sum_{i=0}^{N_1} J(x_{k+i}, u_{k+i}) \\ \text{s.t. } & x_{k+1} = f_d(x_k, u_k); x_0 = \hat{x}_0 \\ & u_{lb} \leq u_k \leq u_{ub} \\ & x_{lb} \leq x_k \leq x_{ub} \end{aligned} \quad (6)$$

where x_k and u_k are states and control input at the k^{th} time step, J is the cost function, J_T is the terminal cost function, $f_d(\cdot, \cdot)$ is the model of the dynamic system, and N_1 is the MPC time horizon. u_{lb} , u_{ub} and x_{lb} , x_{ub} are input and state lower and upper bound vectors, respectively. \hat{x}_0 is the current estimate of the system’s state.

A. Latent Representation

If the aforementioned actor-critic framework is directly applied to the system with a high-dimensional parameters, a large number of system evaluations would be required. Moreover, for each new task, the framework needs to be repeated from scratch. To overcome these limitations, we learn a mapping from the high-dimensional controller parameter space to a lower-dimensional space in a data-driven manner. As in Section I, the intuition is that the well-performing parameters form a subset in a lower-dimensional space can be learned in an unsupervised manner without assumptions on the mapping. For this purpose, the general actor-critic approach is applied and the set of controller parameters that provide reasonable performance (also referred to here as “stable parameters”) is stored in a replay buffer called the training set. This set is utilized to learn the mapping by a VAE using a loss function of the structure

$$\begin{aligned} \min_{\psi, \phi} & \frac{1}{N_t} \sum_{i=1}^{N_t} \|D(E(\theta_i; \phi); \psi) - \theta_i\|_2^2 + \\ & KL(N(\mu_{\bar{\theta}, \phi}, \sigma_{\bar{\theta}, \phi}), N(0, I)) \end{aligned} \quad (7)$$

where $E(\cdot; \phi)$ is the encoder network with parameter vector ϕ , $D(\cdot; \psi)$ is the decoder network with parameters ψ , N_t is the number of samples in the training set, KL is the KL (Kullback–Leibler) divergence between a Gaussian distribution (with $\mu_{\bar{\theta}, \phi}$ as mean and $\sigma_{\bar{\theta}, \phi}$ as standard deviation) and a normal distribution $N(0, I)$. Here, $\mu_{\bar{\theta}, \phi}$ and $\sigma_{\bar{\theta}, \phi}$ denote the mean and standard deviation of the latent vector encodings (i.e., $E(\theta_i; \phi)$, $i = 1, \dots, N_t$) of the training set $\bar{\theta} = \{\theta_i, i = 1, \dots, N_t\}$.

The proposed method is described in Algorithm 1 in which three procedures are defined. The *AC_GEN* uses *TuRBO* to generate N_t number of samples. This procedure takes the function to be optimized (f) as an input and the other input is the mapping function from the lower-dimensional space to the higher-dimensional space (D). Then, it first generates N_{init} number of samples (which could include some manually tuned samples), followed by calls the *TuRBO* algorithm to suggest new samples to be evaluated given previously seen samples. The *TRAIN_VAE* procedure trains a VAE with D and E as the decoder and the encoder neural

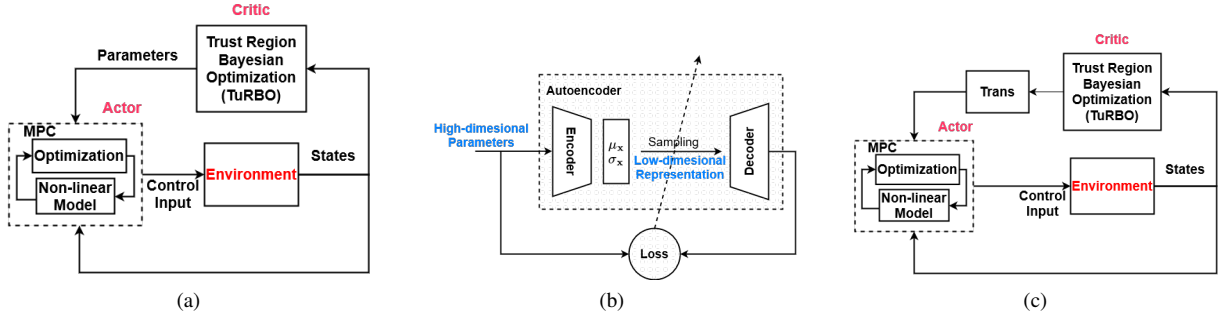


Fig. 2. (a) General Actor-Critic interaction in high-dimensional space; (b) Training the autoencoder to learn a mapping from the high-dimensional parameter space to a lower-dimensional space; (c) Actor-Critic interaction in the lower-dimensional space.

networks, respectively. The parameters of the VAE (i.e., $D \circ E$ in which E is the encoder neural network and D is the decoder) are updated using Stochastic Gradient Descent (SGD) represented by the function GU in the algorithm. The number of batches and the number of epochs are denoted by $n_{batches}$, and n_{epochs} , respectively. The *MAIN* procedure corresponds to the overall execution of the algorithm in which f is the path cost (i.e., the function defined in (2) for the application considered here) and the training samples $((X_s, Y_s))$ are generated by *AC_GEN* called initially with no learned mapping (i.e., Id which denotes “Identity” passed as the mapping function indicating that the parameter search is in the higher-dimensional space). Given the training samples, the decoder D is trained by the *TRAIN_VAE* procedure. In the last phase, a new set of parameters is found (X_r) by taking advantage of the decoder.

Algorithm 1 Proposed Method

Input: BO Cost

Output: X_r, Y_r

```

1: procedure AC_GEN( $f, D$ )
2:    $X \leftarrow [x^i]_{i=1}^{N_{init}}$ 
3:    $Y \leftarrow f(D(X))$ 
4:   for  $i \leftarrow 1$  to  $N_t - N_{init}$  do
5:      $x_{new} \leftarrow TuRBO(X, Y)$ 
6:      $Append(X, x_{new})$ 
7:      $Append(Y, f(D(x_{new})))$ 
8:   return  $X, Y$ 
9: procedure TRAIN_VAE( $X$ )
10:  Initialize  $D$   $\triangleright$  Initializing the Decoder Network  $D$ 
11:  Initialize  $E$   $\triangleright$  Initializing the Encoder Network  $E$ 
12:  for  $i \leftarrow 1$  to  $n_{epochs}$  do
13:    for  $j \leftarrow 1$  to  $n_{batches}$  do
14:       $GU(D \circ E, X^j)$   $\triangleright$  Updating the weights of
the model  $M = D \circ E$  using SGD
15:  return  $D$ 
16: procedure MAIN
17:   $f \leftarrow BO\ Cost$ 
18:   $X_s, Y_s \leftarrow AC\_GEN(f, Id)$ 
19:   $D \leftarrow TRAIN\_VAE(X_s)$ 
20:   $X_r, Y_r \leftarrow AC\_GEN(f, D)$ 

```

IV. RESULTS

We implement the proposed method on a quadruped robot for legged locomotion tasks. To show the efficacy of our proposed approach, we leverage two off-the-shelf MPC frameworks for legged locomotion: a state-of-the-art nonlinear MPC framework called BiConMP [36] and the method proposed in [37]. We have evaluated our method on the Unitree A1, B1, and the Solo12 [38] quadruped robots in the PyBullet simulation environment [39]. We consider the trotting and jumping motions (Fig. 1). In both motions, the robot moves forward with a desired speed of 0.5 m/s in the forward direction. For BiConMP on Solo12 robot, the controller parameter vector to be tuned is of length 77 while for Unitree A1 and B1, the controller parameter vector is of length 25. We consider the running cost of the form

$$q = \begin{cases} C_1, & \text{if the robot falls} \\ \|v_{CoM} - v_{Desired}\|^2, & \text{otherwise} \end{cases} \quad (8)$$

where C_1 is a constant value, v_{CoM} is the velocity vector of the CoM of the robot, and $v_{Desired}$ is the desired velocity vector that the user wants the robot to follow. It should be noted that falling is detected when the height of the CoM is outside the desired boundaries. We run 10000 iterations of the actor-critic method without dimensionality reduction (phase 1) for both trotting and jumping. The number of trust regions is set to 10. Once the parameter vectors are generated, those with a cost of less than 100 are chosen as training samples for VAE. The threshold 100 comes from the fact that the constant C_1 is chosen to be 100 (i.e., we pick parameters that do not at least result in falling). In this step, the chosen samples are used to train the VAE (phase 2). The architecture of the VAE is reported in Table I in which d_{high} and d_{low} are the dimensions of the high-dimensional and low-dimensional parameter spaces, respectively. These dimensions are $d_{high} = 77$ and $d_{low} = 5$ for Solo12, and $d_{high} = 25$ and $d_{low} = 10$ for the Unitree robots. The VAE is trained with a 0.001 learning rate and 64 batch size. Mean Squared Error (MSE) loss is utilized as the loss function and optimized using Adam optimizer. Once the VAE is trained, its decoder is used to find the vector of controller parameters in a lower-dimensional space using the approach presented in Section III. In the last phase of the algorithm, TuRBO is run for only 220 iterations with random initialization. It should

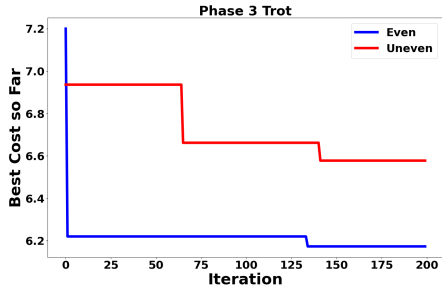


Fig. 3. Unitree A1’s phase 3 best cost so far for 220 iterations of running the algorithm for trotting motion on an even and uneven surface. The final cost for the even surface is 6.17, and 6.57 for uneven surface.

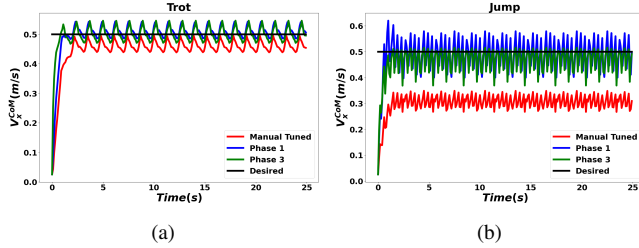


Fig. 4. Comparison of Unitree A1 CoM velocity along x axis for manually tuned set of parameters after phases 1 and 3 for trot (a) and jump (b) motions. The desired velocity of CoM of the robot is 0.5 m/s.

be mentioned that the cost function is sparse meaning the cost value is constant in most of the search space. Therefore, instead of standard BO, TuRBO is utilized for the last phase. In Fig. 3, the accumulative minimum value is depicted for trotting motion. It can be seen that the algorithm successfully converges to a solution that has a relatively low-cost value. We removed initial unstable points from figures 3 and 5 for better visual comparison. Also, the velocity of the CoM of the robot along x axis is depicted in Fig. 4. It can also be seen that the set of parameters found in phases 1 and 3 are more successful in tracking the velocity compared to the manually tuned set of parameters [36], [37].

A. Real-World Simulation

The proposed algorithm enables efficient parameter tuning on a quadruped by learning the mapping between the higher-dimensional parameter space and a lower-dimensional space. The last phase of the algorithm utilizes the learned mapping for performing the controller parameter tuning on the real

TABLE I
THE VALUES d_{high} AND d_{low} FOR THE VAE ARCHITECTURE.

Model	Number of Nodes	Activation Function
Encoder	$\lfloor \frac{d_{high}}{2} \rfloor$	ReLU
	$\lfloor \frac{d_{high}}{4} \rfloor$	ReLU
	d_{low}	Sigmoid
Decoder	d_{low}	ReLU
	$\lfloor \frac{d_{high}}{4} \rfloor$	ReLU
	$\lfloor \frac{d_{high}}{2} \rfloor$	Sigmoid

TABLE II

RESULTS FOR COMPARING THE PROPOSED ALGORITHM WITH THE STATE OF THE ART ALGORITHMS AS CITED.

	Parameters’ Max Dimension	# of Trials	Predefined Features	Transfer- ability	Task Generali- zation
Ours	77	75	✗	✓	✓
[25]	8	50	✗	✗	✗
[32]	50	100	✓	✗	✗
[40]	16	25	✓	✗	✗
[41]	16	100	✓	✗	✗

system (i.e., to transfer the simulation-learned knowledge as encoded in the VAE to the real system). To evaluate the transferability robustness of the algorithm from the simulation to the real world, external perturbations are added to the simulation environment. We considered uneven surface as a source of perturbation. A hilly surface (using Perlin noise with a maximum height of 0.02 m) is added during phase 3 of the algorithm while in the first phase (which is based on the nominal simulation), there is no perturbation when generating the training set. Fig. 3 shows the accumulative minimum for 220 evaluations of the robot in the presence of surface perturbations while performing the trotting motion. Our approach is successful at finding an optimal solution even in the presence of the external perturbation.

B. Task Generalization

In this section, we refer to task generalization as utilizing simulation-learned knowledge (i.e., the learned mapping from high-dimensional controller parameter space to a lower-dimensional space) corresponding to one task to then tune the controller for a different task. For the legged locomotion task, the decoder is trained on a specific motion (e.g., trotting) and then used to find the parameters for another motion (e.g., jumping). The intuition in this task generalization study is that while the motion (i.e., specific control objective) is different, the reasonable subset of controller parameters will still be similar since it is the same underlying dynamics and therefore the learned mapping will still be somewhat valid even with the new motion (albeit requiring a bit more search during phase 3 tuning on the real system on the actual desired task). We run phase 1 of the algorithm for trotting motion, then the VAE is trained in phase 2. Finally, the VAE is used for the third step of the proposed method to find the parameters for the jumping task. In Fig. 5, the cumulative cost is depicted for this case. It can be seen that the algorithm successfully found a parameter vector that results in a stable motion. Moreover, it is to be noted that when phase 1 of the algorithm is directly run for jumping motion, the error is improved after 2000 iterations while in this case, less than 200 iterations are needed. A comparison between different cases is reported in Table III for Unitree A1 and Solo12. In the table, the cost value (considering a horizon of 5000 time steps in all results in this section) for a set of manually tuned parameters is reported in the third column. The best cost found in phases 1 and 3 and then phase 3 in presence of perturbation are given in forth, fifth, and sixth columns,

TABLE III
PATH COST COMPARISON FOR THE ROBOTS.

Robot	Method	Manual Tuned	Phase1	Phase3 (Even)	Phase3 (Uneven)	Phase3 (Even)
Unitree A1	Trot	8.96	5.96	6.17	6.57	9.61 (jump)
	Jump	13.81	8.21	9.52	9.15	20.57 (Trot)
Solo12	Trot	17.73	7.59	7.38	12.72	18.33 (Jump)
	Jump	12.24	8.41	10.26	10.88	10.17 (Trot)

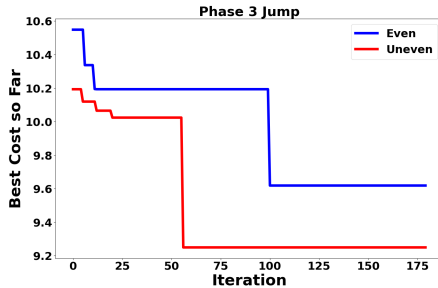


Fig. 5. Unitree A1’s phase 3 best cost so far for 180 iterations of running the algorithm for jumping task using VAE trained on the trotting task. For even surface the final cost is 9.61, and for uneven surface is 9.25.

respectively. The seventh column shows the best cost in phase 3 using a VAE trained on a different task in the fifth column. It can be seen that for both motions, the minimum cost for both phases is less than the manually tuned set of parameters. It is seen that for all the cases except for task generalization, the tuned parameters using our approach results in less error than manually tuned ones. Furthermore, even for task generalization, our approach successfully yields parameters that achieve stable motion.

C. VAE’s Mapping of Stable Regions

A crucial part of the proposed approach is the application of a VAE to find the informative latent representation of the controller parameters, especially over the “good” or “stable” subset of the parameter space in which reasonable performance is achieved. To specifically study this aspect of the VAE, we run TuRBO to find a separate set of parameters that are not used for training the VAE. Then, we pass these parameters to the VAE to obtain the transformed set of parameters. The set of transformed parameters is evaluated on the robot to check if they result in a stable motion. Out of 677 stable points (i.e., parameter vectors that result in stable motion without falls) in the original space, 658 points are stable in the transformed domain (Fig. 6). It is therefore seen that over the stable part of the controller parameter space, the VAE achieves good reconstruction capability, therefore validating its use in focusing the search for controller parameters in the phase 3 of the proposed approach.

D. Transferability to Other Robots

In this section, we evaluate the transferability of the proposed algorithm. Specifically, we consider whether the mapping from high-dimensional to low-dimensional space learned for a specific robot can be potentially used for another robot. We train a VAE for the Unitree A1 robot

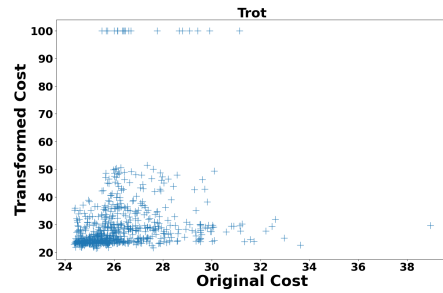


Fig. 6. The cost of transformed points found in phase 1 w.r.t. the cost of the same points is depicted.

while performing trotting. Then, this VAE is used for the Unitree B1 robot while performing jumping. The final cost is 12.98 after 220 iteration. The same procedure is done for finding parameters for the Unitree B1 robot while performing the trotting task. After 220 iterations, the final cost is 33.41. It should be noted that for this evaluation, we did not have access to a manually tuned set of parameters. This further illustrates the significance of the proposed method in that it can enable finding controller parameters for new robots on which even manual tuning has not been performed.

E. Comparative Study

We compare our proposed method with the state of the art algorithms reported in Table II. Unlike prior works, we do not consider predefined low-level features. We instead use a VAE to find a mapping from the high-dimensional controller parameter space to a lower-dimensional space using an approximate simulation model. Moreover, in our work, our mapping-based approach enables optimization of the actual high-dimensional parameter vector (e.g., a vector of length 77 for Solo12) while prior related works constrained the number of parameters to around a maximum of 50. Moreover, our method has the benefit of task generalization and transferability to other robots. These benefits comes from the fact that our method tunes the parameters in a low-dimensional space of extracted features. In comparison, other methods either need to define different low-level features [32], [40], [41], or they need to solve the problem from scratch for different tasks or robots [25].

V. CONCLUSION

An efficient controller parameter tuning algorithm is proposed that is applicable to high-dimensional controller parameter spaces. The method is evaluated on legged locomotion tasks. The method is shown to be effective in finding parameters that result in stable motions with reduced numbers of system evaluations compared to direct tuning of the controller parameters. Also, it has been shown that the approach provides task generalization (transferring learned knowledge from a motion to find parameters for new motion tasks) and generalizability (transferring knowledge learned on one robot to a different robot) properties.

REFERENCES

- [1] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems*, vol. 86, pp. 13–28, 2016.
- [2] M. Focchi, R. Orsolino, M. Camurri, V. Barasuol, C. Mastalli, D. G. Caldwell, and C. Semini, "Heuristic planning for rough terrain locomotion in presence of external disturbances and variable perception quality," in *Advances in Robotics Research: From Lab to Market*. Springer, 2020, pp. 165–209.
- [3] G. Bleidt and S. Kim, "Extracting legged locomotion heuristics with regularized predictive control," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Virtual, August 2020, pp. 406–412.
- [4] H. Utku Unlu, V. Goncalves, D. Chaikalas, A. Tzes, and F. Khorrani, "A control barrier function-based motion planning scheme for a quadruped robot," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Yokohama, Japan, May 2024.
- [5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
- [6] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.
- [7] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, "Towards continual reinforcement learning: A review and perspectives," *arXiv preprint arXiv:2012.13490*, 2020.
- [8] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, New Orleans, LA, February 2018.
- [9] F. Allgöwer, T. A. Badgwell, J. S. Qin, J. B. Rawlings, and S. J. Wright, "Nonlinear predictive control and moving horizon estimation—an introductory overview," *Advances in control: Highlights of ECC'99*, pp. 391–449, 1999.
- [10] D. H. Shim, H. J. Kim, and S. Sastry, "Decentralized nonlinear model predictive control of multiple flying robots," in *Proceedings of the IEEE International Conference on Decision and Control*, vol. 4, Maui, HI: IEEE, December 2003, pp. 3621–3626.
- [11] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Algarve, Portugal: IEEE, October 2012, pp. 4906–4913.
- [12] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
- [13] L. T. Biegler, "A perspective on nonlinear model predictive control," *Korean Journal of Chemical Engineering*, vol. 38, no. 7, pp. 1317–1332, 2021.
- [14] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: an engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 5, pp. 1327–1349, 2021.
- [15] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.
- [16] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.
- [17] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 269–296, 2020.
- [18] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.
- [19] R. Soloperto, M. A. Müller, S. Trimpe, and F. Allgöwer, "Learning-based robust model predictive control with state-dependent uncertainty," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 442–447, 2018.
- [20] J. Berberich, J. Köhler, M. A. Müller, and F. Allgöwer, "Data-driven model predictive control with stability and robustness guarantees," *IEEE Transactions on Automatic Control*, vol. 66, no. 4, pp. 1702–1717, 2020.
- [21] R. G. Goswami, P. Krishnamurthy, and F. Khorrani, "Data-driven deep learning based feedback linearization of systems with unknown dynamics," in *Proceedings of the American Control Conference (ACC)*, San Diego, CA, June 2023, pp. 66–71.
- [22] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe, "Automatic lqr tuning based on gaussian process global optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation*. Stockholm, Sweden: IEEE, May 2016, pp. 270–277.
- [23] S. Bansal, R. Calandra, T. Xiao, S. Levine, and C. J. Tomlin, "Goal-driven dynamics learning via bayesian optimization," in *Proceedings of the IEEE Annual Conference on Decision and Control*. Melbourne, Australia: IEEE, December 2017, pp. 5168–5173.
- [24] M. H. Yeganegi, M. Khadiv, S. A. A. Moosavian, J.-J. Zhu, A. Del Prete, and L. Righetti, "Robust humanoid locomotion using trajectory optimization and sample-efficient learning," in *Proceedings of the 19th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Toronto, Canada: IEEE, October 2019, pp. 170–177.
- [25] M. H. Yeganegi, M. Khadiv, A. Del Prete, S. A. A. Moosavian, and L. Righetti, "Robust walking based on mpc with viability guarantees," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2389–2404, 2021.
- [26] M. Rokonzaman, N. Mohajer, S. Nahavandi, and S. Mohamed, "Learning-based model predictive control for path tracking control of autonomous vehicle," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. Toronto, Canada: IEEE, October 2020, pp. 2913–2918.
- [27] A. Marco, D. Baumann, M. Khadiv, P. Hennig, L. Righetti, and S. Trimpe, "Robot learning with crash constraints," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1439–1446, 2021.
- [28] S. Müller, A. von Rohr, and S. Trimpe, "Local policy search with bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 708–20 720, 2021.
- [29] A. Meduri, H. Zhu, A. Jordana, and L. Righetti, "MPC with sensor-based online cost adaptation," in *Proceedings of the IEEE International Conference on Robotics and Automation*. London, UK: IEEE, May 2023, pp. 996–1002.
- [30] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments—a survey," *IEEE Transactions on evolutionary computation*, vol. 9, no. 3, pp. 303–317, 2005.
- [31] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.
- [32] A. Rai, R. Antonova, F. Meier, and C. G. Atkeson, "Using simulation to improve sample-efficiency of bayesian optimization for bipedal robots," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1844–1867, 2019.
- [33] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, "Scalable global optimization via local bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [34] A. Melkumyan and F. Ramos, "Multi-kernel gaussian processes," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, Barcelona, Spain, July 2011.
- [35] L. Grüne, J. Pannek, L. Grüne, and J. Pannek, *Nonlinear model predictive control*. Springer, 2017.
- [36] A. Meduri, P. Shah, J. Viereck, M. Khadiv, I. Havoutis, and L. Righetti, "Biconmp: A nonlinear model predictive control framework for whole body motion planning," *arXiv preprint arXiv:2201.07601*, 2022.
- [37] J. Di Carlo, P. M. Wensing, B. Katz, G. Bleidt, and S. Kim, "Dynamic locomotion in the MIT Cheetah 3 through convex model-predictive control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Madrid, Spain: IEEE, October 2018, pp. 1–9.
- [38] F. Grimmering, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, T. Flayols *et al.*, "An open torque-controlled modular robot architecture for legged locomotion research," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3650–3657, 2020.
- [39] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.
- [40] A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. Atkeson, "Bayesian optimization using domain knowledge on the atrias biped," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Brisbane, Australia: IEEE, May 2018, pp. 1771–1778.
- [41] R. Antonova, A. Rai, and C. G. Atkeson, "Deep kernels for optimizing locomotion controllers," in *Proceedings of the Conference on Robot Learning*. Mountain View, CA: PMLR, November 2017, pp. 47–56.