Interpretable Robot Locomotion by Monte Carlo Tree Search and Self-Supervised Learning

Yian Wong
The University of Texas at Austin
yian@utexas.edu

Aashish Gottipati
The University of Texas at Austin
agottipati@utexas.edu

Nathaniel Plaxton
The University of Texas at Austin
nateplax@utexas.edu

***Abstract*—Robot locomotion is a fundamental task in robotics. It is a challenging problem from a control perspective, as the robot must operate in real-time, maintaining balance and a forward trajectory in a dynamic environment. Recent approaches have used a combination of reinforcement learning (RL) and generative methods to train agents. However, these methods often lack interpretability, making it difficult to understand and analyze the learned models. MuZero, an algorithm based on Monte Carlo Tree Search (MCTS), has demonstrated super-human performance in games such as Go, shogi, Atari, and chess. To improve interpretability in locomotion agents, we present a variant of MuZero that leverages latent imagination and MCTS for continuous control. We evaluate our trained model on three control tasks in the MuJoCo simulator: Hopper, HalfCheetah, and Humanoid. While we were unable to learn locomotion behavior with our approach, we show that we can improve the interpretability of the learned model by enabling visualization and analysis of latent action-control sequences via MCTS. Furthermore, we show that our approach is able to properly reconstruct and model the dynamics of the MuJoCo environment. Finally, we identify what we believe to be the source of our lower performance, and propose promising potential avenues for future work to amend those issues.**

I. Introduction

Robot locomotion is a fundamental task in robotics that involves the ability of a robot to move from one position to another. It is a challenging problem from a control perspective, as the robot must operate in real-time and maintain balance and a forward trajectory in a dynamic environment. Training these behaviors can be costly and time-consuming, as it often involves many interactions with the environment and sparse rewards.

To overcome these challenges, recent works have used a combination of reinforcement learning (RL) and generative models to train agents within latent space [7, 10]. This approach allows for learning long-horizon behaviors with fewer interactions with the environment, reducing the cost and complexity of training. The use of latent space also allows for more interpretable models, as it enables visualization and analysis of the learned behaviors.

While these methods have improved learning approaches for continuous control, the learned models still suffer from a lack of interpretability. Interpretability in deep learning refers to the ability to understand and explain the behavior of a learned model. It is a key challenge in the field, as models are often treated as black boxes, making it difficult to understand why they fail in specific scenarios and to assess the risks associated with deploying them in the real world. Improving model interpretability can help to build trust in the use of deep learning models and enable researchers to better understand and analyze their behavior.

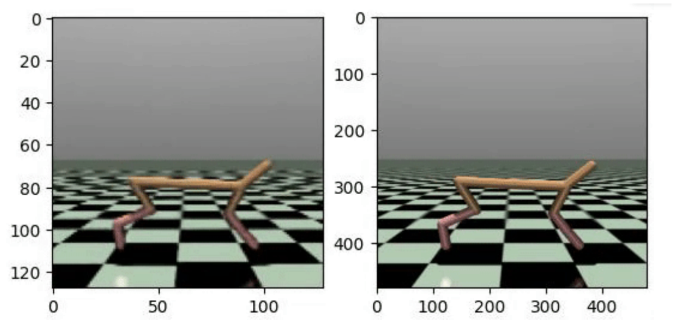

Fig. 1. Depicts two views of the half-cheetah MuJoCo environment. The left image is a latent view our model interprets while the right image is the true view of the environment.

In addition to improving model robustness, transparency helps to mitigate tail-end risk and avoid adverse consequences. For example, many safety-critical systems are transitioning to learning-based methods. A well-known example is Tesla's cars, which come equipped with computer vision models that assist with semi-autonomous driving. If the model fails to identify an obstacle or makes an incorrect control decision, it could lead to accidents and potential harm to passengers. Understanding the circumstances that led to the error and analyzing why the model made the incorrect decision can only be done effectively with improved model transparency. Ensuring that learned models are transparent and interpretable is essential for the safe and effective deployment of these systems in the real world.

In this paper, we present a variant of MuZero that leverages latent imagination and Monte Carlo Tree Search (MCTS) for continuous control [30]. We choose to use MuZero because it has demonstrated superior performance in complex games like Go, Shogi, and Chess as well as realtime games like that of the Atari Learning Environment. Our approach improves the interpretability of the learned model by enabling visualization and analysis of latent action-control sequences through MCTS. We evaluate our trained model on three control tasks in the MuJoCo simulator: Hopper, HalfCheetah, and Humanoid. Our main contributions are:

1) The design and prototype of a more transparent version of MuZero for robot locomotion.
2) A quantitative evaluation of our model on various locomotion control tasks within the MuJoCo simulator.

The remainder of the paper is organized as follows. In Section II, we introduce the necessary background information and related work to contextualize our project. Section III provides an overview of our method and design decisions. Section IV discusses our experimental setups and results for various locomotion tasks. We analyze our findings and suggest potential directions for future work in Section V. Finally, we conclude in Section VI

II. Background and Related Work

In this section, we provide the necessary background information and related work to contextualize our project.

A. Reinforcement Learning

RL has garnered a great deal of attention within the past decade; most notably, in 2016, AlphaGo defeated a professional human Go player [31], a feat that had been viewed as "far off" due to the complexity of Go. RL diverges from previous deep learning approaches in that it iteratively learns to solve a problem through trial-and-error rather than optimizing a specific loss objective [12], which makes RL a prime candidate for tasks where loss functions are ill-defined. At a high level, an RL agent seeks to explore an environment, learning a mapping between environment states and actions, i.e., a transition model. The transition model is heavily influenced by the reward function which encodes the task objective and helps steer the agents towards more favorable states [19]. Thus, by seeking to maximize its reward, the agent learns a transition model (or policy) to achieve a given task in the environment.

However, while RL has demonstrated serious potential [21, 26, 30], there are still aspects to improve. Mainly, RL suffers from high sample complexity, meaning that to learn a robust transition model, an agent typically needs millions of state observations to achieve convergence, i.e., the reward saturates and does not increase significantly. While for software-defined tasks such as video games [26], interacting with the environment millions of times is relatively cheap and fast; however, training in the real world is not due to factors such as setup, safety, and resources. Consequently, RL control research has migrated to training within simulated environments and testing in real environments [18].

Although simulated environments are much "cheaper" than real-world environments, training time can still be extremely expensive due to the computational cost of simulating interactions with the environment. A recent trend has been model-based reinforcement learning which seeks to learn a dynamics model of the agent's environment. Rather than directly sample from the actual environment, an agent samples from the learned model of the environment, enabling both computational and sample complexity benefits [27]. Ha and Schmidhuber propose a generative model, enabling an agent to train almost entirely within latent space [7]. Similarly, Hafner et al. propose Dreamer, an RL agent that learns to solve control tasks via latent imagination [10]. Both methods demonstrate a strong improvement in sample efficiency; thus, in our work, we leverage model-based approaches to reduce costly interactions with the simulated MuJoco environment and enhance sample efficiency.

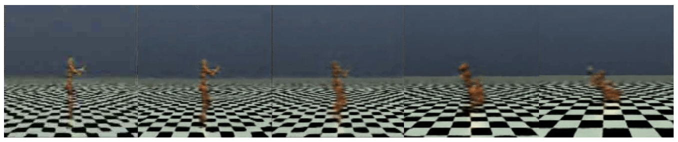
Fig. 2. Sample trajectory generated on Humanoid. The top images are visualizations of the dynamics model, starting with an representation encoded from the first image on the bottom left. The dynamics model empirically produces interpretable predictive trajectories.

B. Reinforcement Learning for Robot Locomotion

Constructing good, differentiable loss functions for robot locomotion tasks tends to be extremely difficult due to task complexity; hence, many have turned towards RL [11]. Specifically, [38] focuses on locomotion for tensegrity robots, leveraging a variant of mirror descent guided policy search (MDGPS) to learn a locomotion policy in simulation. [17] learns a quadrupedal locomotion policy in simulation and evaluates the learned policy on a real quadrupedal robot, demonstrating zero-shot generalization from a simulated to a real world setting. [32] shows that incorporating a modest amount of real-world training (e.g., fine-tuning in the real world) can significantly improve policy performance. [9, 14, 33, 37] explore learning a quadrupedal locomotion policy on a real robot. [37] learns a locomotion policy from 4.5 minutes of real-world data; however, the model leverages prior knowledge of leg trajectories, embedding the prior leg trajectories in action space.

On the other hand, while the previous works have demonstrated learning locomotion policies is feasible, the deployment scenarios are heavily constrained; hence, other methods are necessary to ensure robust generalization. [20] proposes a model-free RL approach for bipedal locomotion that aims to reduce the sim-to-real gap through domain randomization. [25, 3] focus on incorporating exteroceptive perception for robust locomotion. [3] adopts an off-policy approach for real-world training and shares experience between agents in a distributive manner, improving sample complexity. [15] proposes Rapid Motor Adaptation (RMA) for rapid locomotion generalization. RMA learns a base policy for locomotion and incorporates an adaption module for fine-tuning the base policy for the current setting, enhancing the generalization of locomotion policies. [6] incorporates structured noise to boost policy exploration, forcing the model to learn a more general policy for locomotion. Lastly, [8] focuses on the safety implications of training in real-world scenarios, employing a safety-constrained RL framework to learn a locomotion policy with minimal human intervention and oversight. Like prior work, we restrict the scope of our work to a subset

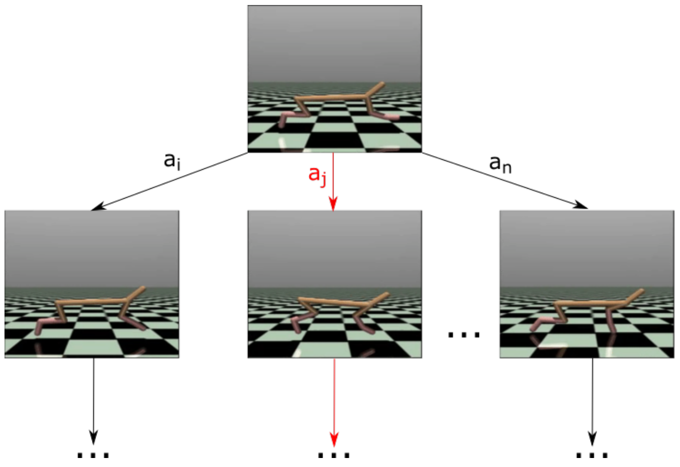

Fig. 3. Example of Monte Carlo Tree Search for interpreting model decisions. The red path indicates the path that maximizes the reward. By visualizing the tree, we are able to gain insight into "why" specific action sequences lead to increased reward.

of locomotion tasks; specifically, we focus on the hopper, half-cheetah, and humanoid locomotion tasks in the MuJoCo simulator.

C. *Model Interpretability for Reinforcement Learning*

Model interpretation for reinforcement learning (RL) is a highly desirable and active area of research [5, 28, 35]. However, model interpretation often raises epistemological questions, so we restrict ourselves to the definition explored in [22]. Lipton surveys the literature on interpretability and identifies five distinct properties: trust, causality, transferability, informativeness, and fairness. In this context, [39] surveys different methods for improving the informativeness of convolutional neural network (CNN) visualizations. [29] explores transferability by arguing that learned models should be viewed as black boxes, i.e., our interpretability methods should be agnostic to the type of model.

In contrast, our work focuses on the causality aspect, i.e., can we better understand our model by visualizing the sequence of actions that maximize the reward along multiple trajectories? Similar to [23], we seek to leverage tree structures to enhance the interpretability of our model. Specifically, we use MCTS [**?**] to visualize how sequences of actions maximize the model's reward across multiple trajectories within a finite horizon. This allows us to better understand the decisions made by the model and the factors that influence its behavior.

III. Approach

A. *MuZero*

Our algorithm is closely based on the MuZero algorithm [30], so we will provide a brief summary of the algorithm here. MuZero is a model-based reinforcement learning algorithm that is based on the AlphaGo algorithm [31]. At each timestep t, MuZero predicts the predictive policy distribution $p_t(a|s)$ and the state value v_t. In addition, MuZero also learns an internal Markov Decision Process (MDP) that closely represents the actual MDP it is learning from. This internal MDP is a learned model μ_θ that is parameterized by θ and learns a dynamics function $g_\theta(s_{t+1}, r_t|s_t, a_t)$ (which encapsulates the transition and reward function), and a representation function $h_\theta(s_t|o_t)$, which encodes an observation o_t from the real MDP into a latent representation s_t. In MuZero, the transition function and representation function are modeled deterministically.

At each timestep t, MuZero uses the learned internal MDP and the predicted policy, value, and rewards to perform a Monte-Carlo Tree Search (MCTS). The MCTS looks ahead in the predictive game tree and uses the predicted outcomes to refine the predictive policy p_t to a posterior policy distribution π_t.

During training, MuZero trains its learned MDP, value, and reward functions using supervised learning to make them more similar to the true MDP.

B. *World Models and Dreamer*

In this work, we are heavily influenced by previous studies that have used image reconstruction to learn a latent space through a variational autoencoder (VAE) [13]. Specifically, we draw inspiration from the works of [7] and [10], which have demonstrated the effectiveness of using a VAE for this purpose. We adopt their approach and use a VAE to learn a compact representation of the environment's image observations.

Furthermore, we incorporate the use of Mixture Density Networks (MDN) [2] in our approach, as demonstrated in World Models [7]. By using an MDN to express the continuous distribution across the latent space of the transition function, we are able to make the transition function stochastic and use the log-likelihood as the supervised learning objective instead of the euclidean distance. This allows for a more flexible and accurate model of the environment dynamics. Overall, our approach builds upon the work of previous studies by combining the use of VAE and MDN for more effective learning of the latent space and transition function.

C. *Our Modifications*

We describe our algorithm in more detail below. Our approach is based on the MuZero algorithm [30], which uses Monte Carlo Tree Search (MCTS) to generate predictive plans from the agent. However, this approach lacks interpretability, which is crucial in the field of robot locomotion where agents make decisions that can have significant impacts.

To improve interpretability, we propose using a variational autoencoder (VAE) [13] to encode the environment's image observations into a compressed latent space. This allows us to reconstruct the observations for any sample in the latent space, providing a more interpretable plan. Additionally, the use of a VAE allows us to compress the high-dimensional image data into a lower-dimensional representation, making it more efficient to process. Overall, our approach improves upon the base MuZero implementation by providing a more interpretable and efficient way to generate predictive plans from the agent.

One major challenge of model-based reinforcement learning is that it can be difficult to learn the model efficiently, and approximation errors can lead to poor policy behavior. To address this issue, we model the dynamics function as a probability distribution, as shown in previous studies [7, 10]. This is implemented using a Mixture Density Network, which allows the model to express uncertainty about certain transitions in the Markov Decision Process (MDP).

In the MCTS, we account for the uncertainty of the model by using the Open-Loop MCTS approach for stochastic dynamics functions from [16]. In child nodes of the MCTS, instead of storing the predicted latent state, the node stores the root node latent state encoded by h and the actions taken to reach the node. During selection on the node, the dynamics function is given the root node and the action sequence to sample a new latent state. This allows the MCTS to pick actions based on the expected value of the latent state distribution, taking into account the uncertainty of the model. Overall, we believe this approach allows the MCTS to make more informed and robust decisions based on the uncertainty of the dynamics function.

Drawing from this, our loss for the dynamics function (transition and reward function) is:

$$l_{g_\theta} = -log(g_\theta(s_{t+1}, r_t | s_t, a_t))$$

The value objective trained using the bootstrapped n-step return:

$$l_v = x(v_t, z_t)$$
$$z_t = r_t + \gamma r_{t+1} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n v_{t+n}$$

where j is the Huber distance between the two values.

The policy objective is described as the distance between the MCTS posterior distribution and predictive prior policy:

$$l_p = y(p_t, \pi_t)$$

where y is the cross entropy loss between the two distributions.

Lastly, we train our VAE using Huber Loss as our estimated lower bound, and use a KL divergence from $Normal(0, 1^2)$ as our regularization prior:

$$l_h = j(\hat{s_t}, s_t) + KL(h_t(o_t), Normal(0,1)),\ \hat{s_t} \sim h_t(o_t) \quad .$$

We thus train our models using backpropagation-through-time, using the combined loss function:

$$l = l_{g_\theta} + l_v + l_p + l_h$$

IV. Experiments

We leverage the open source MuJoCo physics engine [34] to train and evaluate our approach, and OpenAI's gym [4] wrapper to interface with the MuJoCo simulator. MuJoCo offers a variety of built-in locomotion tasks to train from; specifically, we tested on three built-in locomotion tasks: Hopper, HalfCheetah, and Humanoid. These tasks involve continuous action spaces, which we discretized to simplify our implementation. While MuZero has been successfully adapted to the continuous case, we chose to discretize the action space for the sake of staying true to the original MuZero paper. We divided the action space into equal intervals and enumerated each joint combination of these intervals as separate actions, as in previous work.

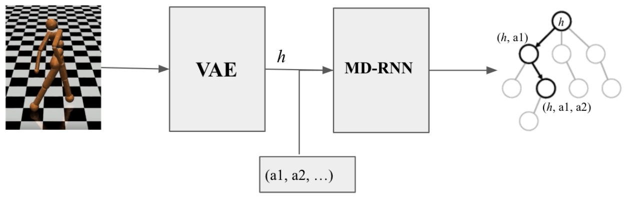

Fig. 4. Architecture of our approach. The agent is given an image for the current state, which is then passed through a variatonal auto-encoder to produce a latent-space representation. This representation, combined with a sequence of actions, is used by the MD-RNN to create a latent representation of the predicted new state. Our MCTS uses this process to pick actions based on the expected value of the latent state distribution.

However, discretizing the action space presents several challenges specific to MuZero and MuJoCo. As the number of actuators in the environment increases, the discretized action space grows exponentially. For example, Hopper has 3 actuator controls, which leads to an action space of 108 when each actuator is divided into 4 increments. Humanoid, on the other hand, has 17 actuators, resulting in an action space of 19652. Compared to other real-time environments like the Atari Learning Environment, these action spaces are significantly larger. This can greatly impact our algorithm, as larger action spaces increase the branching factor of the Monte-Carlo search tree.

We implemented MuZero according to the original paper, using residual networks and the open-loop approach for Monte-Carlo Tree Search. We based our implementation on the MuZero General GitHub repository [36], modifying the representation function with a symmetrical deconvolutional neural network and the stochastic dynamics function with a Mixture Density network.

In our experiments, we trained our model using the Adam optimizer with a learning rate of 0.0001 and a batch size of 32. We used a discount factor of 0.999 and a replay buffer size of 100000. We trained asynchronously until we reached 600000 training steps on the policy. At each iteration, we performed 20 Monte Carlo Tree Search simulations with a maximum rollout length of 50. We used a temperature parameter of 0.9 for the tree policy. These hyperparameters were chosen based on previous work on MuZero [30] in the Atari case.

We also used bootstrapped our dynamics and representation models using 3000 random samples from each task to train the dynamics model, and we collected additional samples during training for the replay buffer.

As shown in 5, our model does not appear to learn effective behaviors in the three MuJoCo tasks. However, we found that the model was able to accurately learn the underlying Markov Decision Process (MDP) of the tasks, as depicted in 2 and 1. As such, we attribute the shortcomings of our model to the RL problem setup, and not the learning of the model

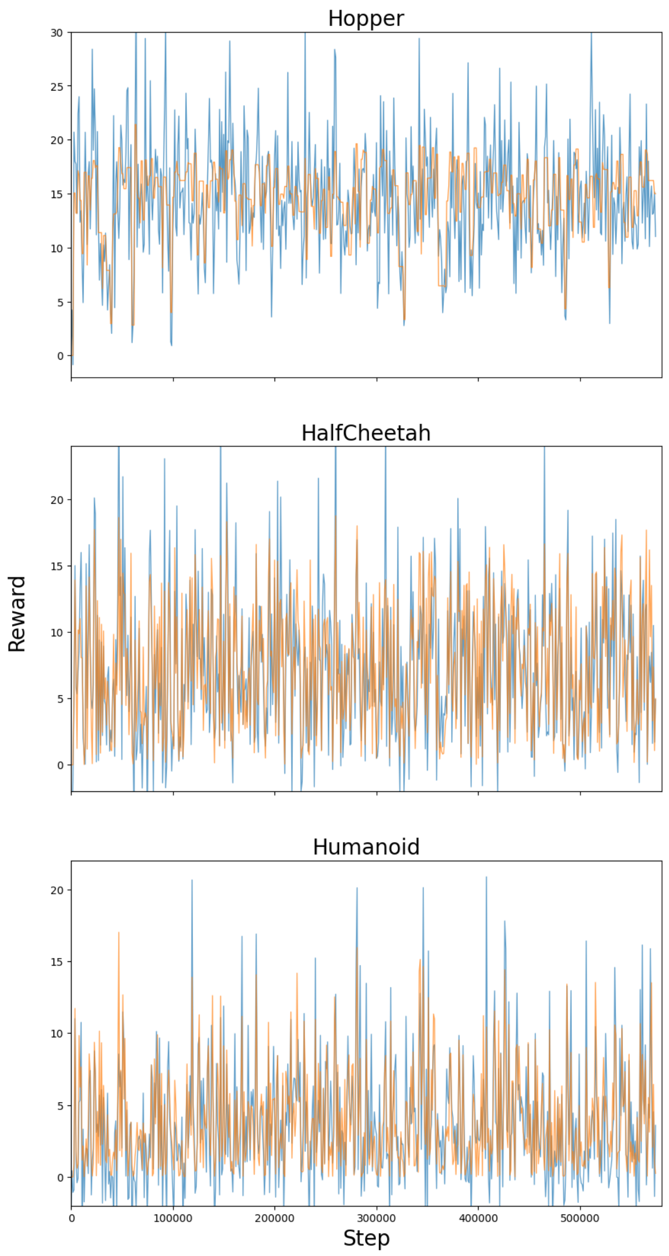

Fig. 5. Performance approach on Hopper, HalfCheetah, and Humanoid, measured by maximum test reward (y-axis) achieved at the given timestep (x-axis). Blue lines are results using our approach, orange are results using MuZero. The learning curve for both MuZero and our approach are similar, and generally show hardly any learning. Emperically, both models do not seem to solve the tasks (ie reach the end). While simpler, model-free algorithms like PPO and DDPG are able to solve MuJoCo tasks consistently [12], we were not able to find such results.

dynamics- specifically the discretization of continuous tasks in MuJoCo. Continuous algorithms like DDPG and PPO are able to solve this problem by directly acting in the continuous space. However, even the generalized MuZero implementation fails in this space. We surmise that the action space is too large for MuZero, leading to excessively high branching factors in the game tree that make it difficult for the algorithm to effectively learn. It is unclear whether our modifications to the image and stochastic dynamics modeling have improved performance. Extending MuZero and our other modifications to the continuous action space is an area for future work, which we will discuss further in the next section.

V. Discussion and Future Work

We believe that the bottleneck for performance of our algorithm is the primitive discretization of the action space which we use in our MCTS implementation. Although there exist MCTS variations that have been made for use in continuous spaces, we decided to use a simple discretization approach given the limited timeframe of our project. After examining our results, we believe the decision to primitively discretize the action space linearly and then use traditional MCTS was the main inhibitor of our performance.

Another possible explanation for MuZero's shortcomings is that the *way* we discretize may not be fine-grained enough or that uniformly incrementing the values may not happen to land on the optimal subset of the continuous action space. We hypothesize that we can potentially tackle the discretization problem of robot locomotion by adding an expert demonstration layer. By collecting expert demonstrations, we have rich data as to what are the most common configurations picked by the expert. We propose that in order to find an effective discretization of the environment, we use a set of expert demonstrations $\{s_t, a_t\}_{t=0}^{N}$, and identify clusters within the actions $\{a_t\}_{t=0}^{N}$ of the demonstrations, using an algorithm like K-Means [24]. We then use the cluster centers as a discrete proxy to the real action space, allowing MuZero to pick from a finite subset of the continuous action space. While this idea will likely enable MuZero to effectively learn in these tasks, the high-level idea of learning from the behavior of an expert goes against the idea of MuZero's ”learning from nothing” concept [31]. Using this form of discretization is left for future work.

A recent paper by Abidi et al. which was released midway through our project proposes a new algorithm, Continuous Monte Carlo Graph Search (CMCGS) [1]. In this algorithm, the authors leverage the sharing of policies between states during planning. They group similar states together into nodes, then combine those nodes to form a graph rather than a traditional search tree. They demonstrate that even traditional approaches for continuous versions of MCTS (specifically using progressive widening, henceforth refered to as MCTS-PW) are unable to perform well on most tasks in the DeepMind Control Suite, including those most similar to the tasks on which we are evaluating our solution. In their findings, MCTS-PW was only able to achieve a low reward and exhibited minimal improvement in performance with more training time, similarly to our solution. In contrast, they show that CMCGS is able to perform relatively well on such tasks. On the Cheetah-Run and Walker-Walk tasks, for example, CMCGS demonstrates a 10x improvement in performance relative to MCTS-PW. Additionally, CMCGS is shown to be able to successfully improve performance given more training time. Incorporating this framework into our solution in place of traditional MCTS would be a natural first step towards increasing the effectiveness of our approach.

It is worth noting that CMCGS [1] is shown to only be able to achieve about 0.5x the overall performance of Dreamer [10] on the same tasks. However, Dreamer was given about 40x as much training time before being evaluated. Due to this discrepancy, we still believe that it would be worth exploring combining the two algorithms, in the hope that interpretability can be significantly improved without sacrificing too much performance.

VI. Conclusion

In this paper, we presented a new approach to learning for locomotion tasks. Our algorithm combined recent works in latent imagination with Monte Carlo Tree Search in an effort to produce a more interpretable model. While our approach was unable to demonstrate good performance for learning the locomotive tasks overall, we showed that we could properly reconstruct and model the dynamics of the MuJoCo environment, enabling visualization of latent control sequences within the search tree. After further evaluation, we believe that we have identified the primary cause of our performance bottleneck and present ideas for future work which could lead to achieving good results.

References

[1] Amin Babadi, Yi Zhao, Juho Kannala, Alexander Ilin, and Joni Pajarinen. Continuous monte carlo graph search. 2022. URL https://arxiv.org/abs/2210.01426.

[2] Christopher M Bishop. Mixture density networks. 1994.

[3] Michael Bloesch, Jan Humplik, Viorica Patraucean, Roland Hafner, Tuomas Haarnoja, Arunkumar Byravan, Noah Yamamoto Siegel, Saran Tunyasuvunakool, Federico Casarini, Nathan Batchelor, et al. Towards real robot learning in the wild: A case study in bipedal locomotion. In *Conference on Robot Learning*, pages 1502–1511. PMLR, 2022.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

[5] Jianyu Chen, Shengbo Eben Li, and Masayoshi Tomizuka. Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[6] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *Advances in neural information processing systems*, 31, 2018.

[7] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018.

[8] Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*, 2020.

[9] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[10] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[11] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

[12] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[14] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2619–2624. IEEE, 2004.

[15] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[16] Erwan Lecarpentier, Guillaume Infantes, Charles Lesire, and Emmanuel Rachelson. Open loop execution of tree-search algorithms, extended version. *arXiv preprint arXiv:1805.01367*, 2018.

[17] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5 (47):eabc5986, 2020.

[18] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[19] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[20] Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2811–2817. IEEE, 2021.

[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel,

Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[22] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[23] Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 414–429. Springer, 2018.

[24] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[25] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[27] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86 (2):153–173, 2017.

[28] Erika Puiutta and Eric Veith. Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*, pages 77–95. Springer, 2020.

[29] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[30] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[31] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529 (7587):484–489, 2016.

[32] Laura Smith, J Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1593–1599. IEEE, 2022.

[33] Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. *arXiv preprint arXiv:2208.07860*, 2022.

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

[35] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018.

[36] Aurèle Hainaut Werner Duvaud. Muzero general: Open reimplementation of muzero. https://github.com/werner-duvaud/muzero-general, 2019.

[37] Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. Data efficient reinforcement learning for legged robots. In *Conference on Robot Learning*, pages 1–10. PMLR, 2020.

[38] Marvin Zhang, Xinyang Geng, Jonathan Bruce, Ken Caluwaerts, Massimo Vespignani, Vytas SunSpiral, Pieter Abbeel, and Sergey Levine. Deep reinforcement learning for tensegrity robot locomotion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 634–641. IEEE, 2017.

[39] Quan-shi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1): 27–39, 2018.