A Comparison of Actor Critic and Policy Gradient Algorithms for Learning Mechanical Designs

Andrew Albright & Adam Smith
University of Louisiana at Lafayette
104 E. University Circle, Lafayette, LA 70503
andrew.albright1@louisiana.edu, adam.smith2@louisiana.edu

Todo list

Make a hyperameters table for PPO. 4

Abstract

Legged systems have many advantages when compared to their wheeled counterparts. For example, they can more easily navigate extreme, uneven terrain. However, there are disadvantages as well, particularly the difficulty seen in modeling the nonlinearities of the system. Research has shown that using flexible components within legged locomotive systems improves performance measures such as efficiency and running velocity. Tuning mechanical designs to define what might be an optimal performing system is challenging though, particularly when the system is nonlinear. Reinforcement learning can be tasked with learning mechanical parameters of a system to match a control input. It is shown in this work that when deploying reinforcement learning to find design parameters for a monopode jumping system, the designs the algorithms learn are optimal within the design space provided to the agents.

1. Introduction

The use of flexible components within legged locomotive systems has proved useful for both reducing power consumption and increasing performance [**?**,**?**,**?**]. However, designing controllers for these systems is difficult as the flexibility of the system generates nonlinear models. As such, employing series-elastic-actuators (SEA) instead of flexible links is an attractive and popular solution, since the models of the systems become more manageable [**?**, **?**, **?**]. Still, the use of SEAs do not represent the full capability of flexible systems. As a result, other methods that use flexible tendon-like materials meant to emulate more organic designs have been proposed [**?**]. These, however, are still not representative of fully flexible links, which have been shown to drastically improve locomotive performance measures such as running speed [**?**].

Control methods have been developed that work well for flexible systems like the ones mentioned [**?**, **?**]. However, as the systems increase in dimensionality, effects such as dynamic coupling between members make such methods challenging to implement. As such, work has been done that uses neural networks and methods such as reinforcement learning (RL) to develop controllers for flexible systems [**?**, **?**]. For example, RL has been used to create control strategies for both flexible-legged and rigid locomotive systems that when compared, show the locomotive systems outperform their rigid counterparts [**?**]. Furthermore, those controllers were shown to be robust to changes in design parameters.

In addition to the work done using RL to develop controllers for flexible systems, work has been completed which shows that this technique can be used to concurrently design the mechanical aspects of a system and a controller to match said system [**?**]. These techniques have even been used to define mechanical parameters and control strategies where the resulting controller and hardware were deployed in a sim-to-real process, validating the usability of the technique [**?**]. Using this technique for legged-locomotion has also been studied, but thus far has been limited to the case of rigid systems [**?**].

As such, this paper explores the use of RL for concurrent design of flexible-legged locomotive systems. A simplified flexible jumping system was used where, for the initial work, the control input was held fixed so that an RL algorithm was tasked with only learning optimized mechanical parameters. The rest of the paper is organized such that in the next section, similar work will be discussed. In Section 3, the monopode environment details will be defined. Next, in Section 4, the input used during training will be explained. Then, in Section 5, the algorithm used along with the method of the experiments will be presented. The performance of the learned designs are shown in Section 6.

2. Related Work

2.1. Flexible Locomotive Systems

The use of flexible components within locomotive robotics systems has shown improvements in performance measures such as movement speed and jumping height [**?**,**?**]. Previous work has shown that the use of flexible components in the legs of legged locomotion systems increase performance while decreasing power consumption [**?**]. Research also has been done showing the uses of series-elastic-actuators for locomotive systems [**?**]. In much of this work, human interaction with the robotic systems is considered such that rigidity is not ideal [**?**]. The studies of flexible systems are challenging however, as the models which represent them are often nonlinear and therefore difficult to develop control systems for. As such, finding optimal mechanical designs is challenging and often times overlooked.

2.2. Controlling Flexile Systems Using RL

Control methods developed for flexible systems have been shown to be effective for position control and vibration reduction [**?**, **?**]. Because of the challenges seen in scaling the controllers, methods utilizing reinforcement learning are of interest. This method has been used in simple planar cases, where it was compared to a PD control strategy for vibration suppression and proved to be a higher performing method [**?**]. Additionally, it has also been shown to be effective at defining control strategies for flexible-legged locomotion. The use of actor-critic algorithms such as Deep Deterministic Policy Gradient [**?**] have been used to train running strategies for a flexible legged quadruped [**?**]. Much of the research is based in simulation, however, and often the controllers are not deployed on physical systems, which leads to the question of whether or not these are useful techniques in practice.

2.3. Concurrent Design

Defining an optimal controller for a system can be difficult due to challenges such as mechanical and electrical design limits. This is especially true when the system is flexible and the model is nonlinear. A solution to this challenge is to concurrently design a system with the controllers so that the two are jointly optimized. This strategy has been used to develop better performing mechatronics systems [**?**]. More recent work has used advanced methods such as evolutionary strategies to define robot design parameters [**?**]. In addition to evolutionary strategies, reinforcement learning has been shown to be a viable solution for concurrent design of 2D simulated locomotive systems [**?**]. This is further shown to be a viable method by demonstrating more complex morphology modifications in 3D reaching and locomotive tasks [**?**]. However, these techniques have not yet been applied to flexible systems for locomotive tasks.

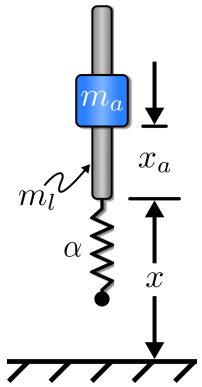

Figure 1. Monopode System

3. Monopode Model

The monopode model in Figure 1 has been shown to be useful for creating a simplified representation of many animal based running and jumping gaits [**?**]. As such, it is used in this work to demonstrate the ability of reinforcement learning for the mechanical design aspect of concurrent design. The model parameters used in the simulations in this paper are summarized in Table 1. The variable m_a represents the mass of the actuator, which moves along the rod of mass m_l. A nonlinear spring shown in the figure by constant α is used to represent flexibility within the jumping system. A damper (not shown in Figure 1), is parallel to the spring. Variables x and x_a represent the vertical position of the system with respect to the ground and the position of the actuator along the rod, respectively.

The equations of motion describing the system are:

$$\ddot{x} = \frac{\gamma}{m_t}\left(\alpha\, x + \beta\, x^3 + c\,\dot{x}\right) - \frac{m_a}{m_t}\,\ddot{x}_a - g \qquad (1)$$

where x and $\dot{x}$ are position and velocity of the rod, respectively, the acceleration of the actuator, $\ddot{x}_a$, is the control input, and m_t is the mass of the complete system. Constants α and c represent the nonlinear spring and damping coefficient, respectively, and constant β is set to $1e8$. Ground contact determines the value of γ, so that the spring and damper do not supply force while the leg is airborne:

$$\gamma = \begin{cases} -1, & x \leq 0 \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

Additionally, the system is constrained such that it only moves vertically and the spring can only compress a certain amount. In this work the spring is allowed to compress the same distance that the actuator can move, which is 0.008 m.

Table 1. Monopode Model Parameters

Model Parameter	Value
Mass of Leg, m_l	0.175 kg
Mass of Actuator, m_a	1.003 kg
Spring Constant, $\alpha_{nominal}$	5760 N/m
Natural Frequency, ω_n	$\sqrt{\frac{\alpha}{m_l+m_a}}$
Damping Ratio, $\zeta_{nominal}$	1e-2 & 7.5e-2 $\frac{\text{N}}{\text{m/s}}$
Actuator Stroke, $(x_a)_{\max}$	0.008 m
Max. Actuator Velocity, $(\dot{x}_a)_{\max}$	1.0 m/s
Max. Actuator Accel., $(\ddot{x}_a)_{\max}$	10.0 m/s^2

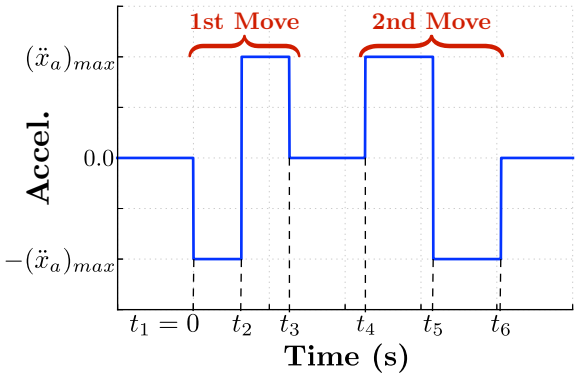

Figure 2. Jumping Command

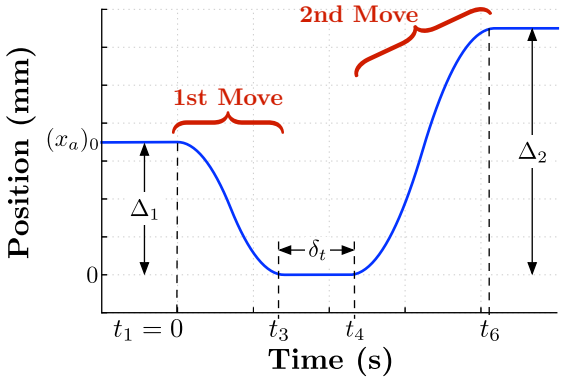

Figure 3. Resulting Actuator Motion

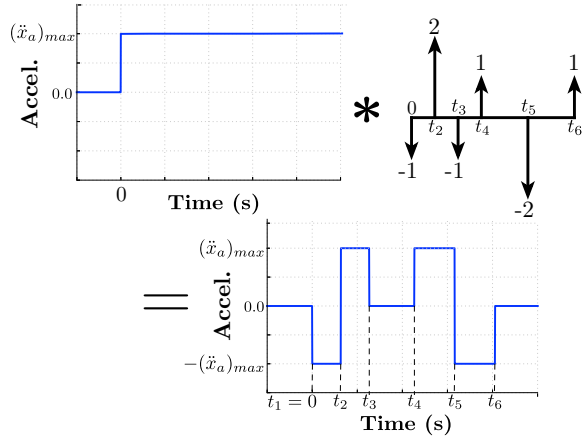

Figure 4. Decomposition of the Jump Command into a Step Convolved with an Impulse Sequence

4. Jumping Command Design

Bang-bang based jumping commands like the one shown in Figure 2 are likely to result in a maximized jump height [**?**]. For this command, the actuator mass travels at maximum acceleration within its allowable range, pauses, then accelerates in the opposite direction. Commands designed to complete this motion are bang-bang in each direction, with a selectable delay between them. The resulting motion of the actuator along the rod is shown in Figure 3. Starting from an initial position, $(x_a)_0$, it moves through a motion of stroke length Δ_1, pauses there for δ_t, then moves a distance Δ_2 during the second portion of the acceleration input.

This bang-bang-based profile can be represented as a step command convolved with a series of impulses, as shown in Figure 4 [**?**]. Using this decomposition, input-shaping principles and tools can be used to design the impulse sequence [**?**, **?**]. For the bang-bang-based jumping command, the amplitudes of the resulting impulse sequence are fixed, $A_i = [-1, 2, -1, 1, -2, 1]$. The impulse times, t_i, can be varied and optimal selection of them can lead to a maximized jump height of the monopode system [**?**]. Commands of this form will often result in a stutter jump like what is shown in Figure 5, where the small initial jump allows the system to compress the spring to store energy to be used in the final jump. This jumping command type was used as the input for the monopode during the simulation phase of training.

5. Learning Mechanical Design

5.1. Reinforcement Learning Algorithm

Two different algorithms are used and compared in this work for finding mechanical design parameters. They are the Twin Delayed Deep Deterministic Policy Gradient (TD3) and the Proximal Policy Optimization (PPO) algorithm [**?**, **?**]. A comparison of actor-critic and policy gradient based algorithms will be shown such that one might be selected in future work for implementing a fully concurrent design process. The StableBaselines3 implementation of the algorithms was used as they are well written, popular in research, and thoroughly documented [**?**].

The training hyperparameters were selected based on the

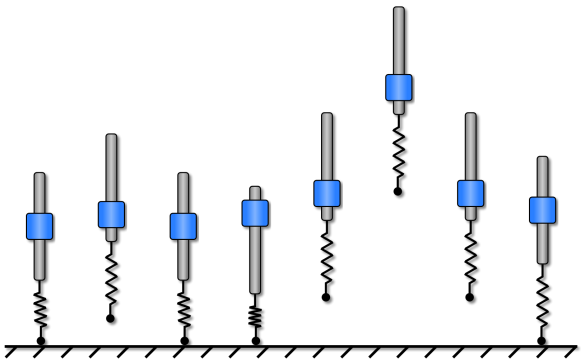

Figure 5. Example Stutter Jump

Table 2. TD3 Training Hyperparameters

Hyperparameter	Value
Learning Rate, α	0.001
Learning Starts	100 Steps
Batch Size	100 Transitions
Tau, τ	0.005
Gamma, γ	0.99
Training Frequency	1:Episode
Gradient Steps	$\propto$ Training Frequency
Action Noise, ϵ	None
Policy Delay	1 : 2 Q-Function Updates
Target Policy Noise, ϵ	0.2
Target Policy Clip, c	0.5
Seed	100 Random Seeds

algorithm author recommendations and StableBaselines3 experimental findings. The vales are displayed in Table 2. All of the hyperparameters, with the exception of the rollout (Learning Starts) and the replay buffer, were set according to StableBaselines3 standards. The rollout setting was defined such that the agent could search the design space at random, filling the replay buffer with enough experience to prevent the agent from converging to a design space that was not optimal. The replay buffer was sized proportional to the number of training steps due to system memory constraints.

The training hyperparameters for PPO found in Table 3 were set similarly to TD3's. All the PPO hyperameters, other than learning rate and N-steps, were set according to the algorithm's author and StableBaselines3's recommendations. The learning rate and N-steps were adjusted as the algorithm was not able to converge within 1000 episodes given the initial hyperparameter values of 0.0003 and 2048.

Make a hyperameters table for PPO.

Table 3. PPO Training Hyperparameters

Hyperparameter	Value
Learning Rate, α	0.001
Batch Size	64
Clip Range, ϵ	0.2
Gamma, γ	0.99
GAE-Lambda,	0.95
Training Frequency	1:Episode
Gradient Steps	$\propto$ Training Frequency
Policy Delay	1 : 2 Q-Function Updates
Target Policy Noise, ϵ	0.2
Target Policy Clip, c	0.5
Seed	100 Random Seeds

5.2. Training Environment Design

To allow the agent to find a mechanical design, a reinforcement learning environment conforming to the OpenAI Gym standard [**?**] was created for the monopode model described in Section 3, including a fixed controller input based on the algorithm described in the previous section. Unlike the common use case for RL, which is tasking the agent with finding a control input to match a design, the agent in this work was tasked with finding mechanical parameters to match a control input.

The mechanical parameters the agent was tasked with optimizing were the spring constant and the damping ratio of the monopode system. At each episode during training, the agent selected a set of design parameters from a distribution of available designs and a simulation was run with those parameters and a fixed control input. The actions applied, $\mathcal{A}$, and transitions saved, $\mathcal{S}$, from the environment were defined as follows:

$$\begin{aligned}\mathcal{A} = \{\{a_\alpha \in \mathbb{R} : [0.1\alpha, 1.9\alpha]\},\\ \{a_\zeta \in \mathbb{R} : [0.1\zeta, 1.9\zeta]\}\}\end{aligned} \tag{3}$$

$$\mathcal{S} = \left\{\mathbf{X},\, \dot{\mathbf{X}},\, \mathbf{X}_a,\, \dot{\mathbf{X}}_a\right\} \tag{4}$$

where α and ζ are the nominal spring constant and damping ratio of the monopode, respectively; x_t and $\dot{x}_t$ are the rod height and velocity steps of the monopode, and x_{at} and $\dot{x}_{at}$ are the actuator position and velocity steps of the monopode, all captured during simulation.

5.3. Reward Function Design

The RL algorithm was utilized to find designs for two different reward cases. Time series data was captured during the simulation phase of training and was used to evaluate the designs performance through these rewards. The first reward case used was:

$$\mathbb{R}_1 = (\mathbf{X})_{\text{max}} \tag{5}$$

where $\mathbf{X}$ was the timeseries rod height of the monopode captured during simulation. The goal of the first reward was to find a design that would cause the monopode to jump as high as possible.

The reward for the second case was:

$$\mathbb{R}_2 = -\frac{(\mathbf{X})_{\max} - x_s}{x_s} \tag{6}$$

where x_s was the desired jump height, which was set to 0.1 m. The second case was utilized to test RL's ability to find a design that minimized the error between the maximum height reached and the desired maximum height to reach.

5.4. Training Schedule

To evaluate the ability of an RL algorithm to robustly find design parameters meeting performance needs regardless of the neural network initializations, 10 different agents were trained with different network initialization seeds. The RL agents were trained in two different environments, one with a narrow range of allowable damping ratios and one with a wider range of possible damping ratios.

The number of episodes that were performed was 1000, with the first 100 being rollout steps. This provided the agents, in both cases previously mentioned, enough learning time to converge to designs that satisfied the performance requirements. During the training process, the height reached during the simulation phase (per environment step) and the design parameters selected by the algorithm where collected to evaluate the learning process.

6. Jumping Height Reached

6.1. Design Space Performance

Figures 6 and 7 represent the heights the monopode could reach for the two different design spaces. The design space provided for the first case, shown in Figure 6, represents a space where the allowable damping ratio was limited to a fairly narrow range. This limits the solution space, making it less likely that the agent will settle to a locally optimal value. The design space provided for the second case, shown in Figure 7, represents a space where a wider range of damping ratios are allowed. This wider range of possible values makes it more likely that the agent will settle to a local maxima.

6.2. Design Learned Given Narrow Design Space

Figure 8 shows the height achieved by the learned designs for the agents given the narrow range of possible damping ratio values. For the agents learning designs to maximize jump height, Figure 8 can be compared with Figure 6 showing that the agent learned a design nearing one

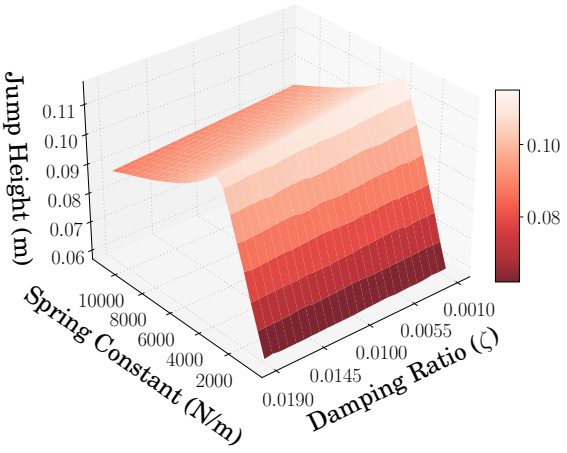

Figure 6. Jumping Performance of Narrow Design Space

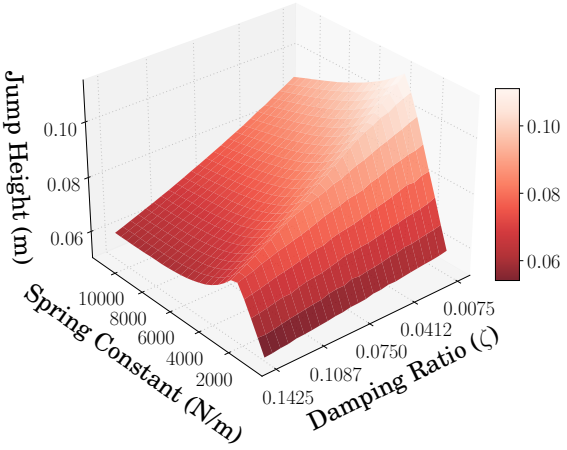

Figure 7. Jumping Performance of Broad Design Space

which would achieve maximum performance. Additionally, looking at the agents learning designs to jump to the specified 0.01 m, the designs learned accomplish this with slightly more variance than that of the maximum height case. Figure 9 shows the rewards the agents received during training and support that both agent types learned designs which converged.

The average and standard deviation of the spring constant and damping ratio design parameters the agents selected during training are shown in Figures 10 and 11. These plots represent the learning curves for the agents learning design parameters to maximize jump height and the agents learning design parameters to jump to 0.01 m. There is a high variance in both the spring constant and the damping ratio found for the agents that learned designs to jump to a specified height. The agents which were learning

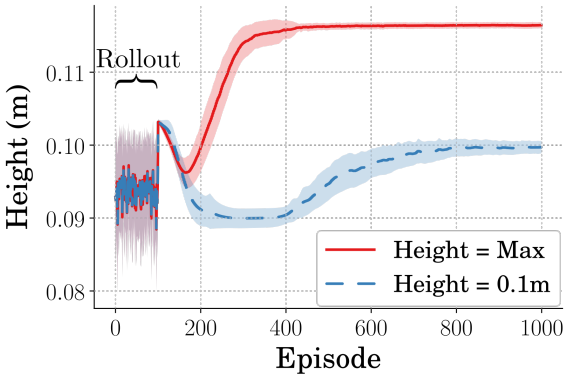

Figure 8. Height Reached During Training

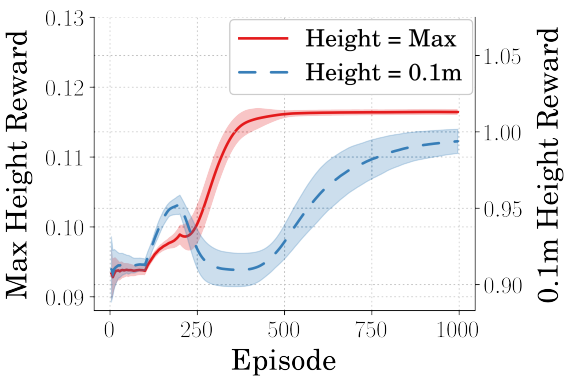

Figure 9. Reward Received During Training

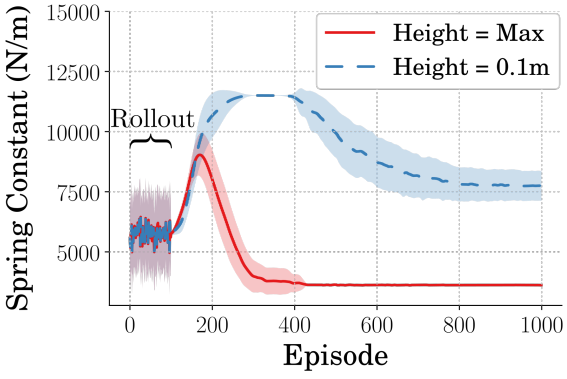

Figure 10. Spring Constant Selected During Training

designs which maximized height found designs with very little variance in terms of spring constant and significantly less variances in terms of damping ratio.

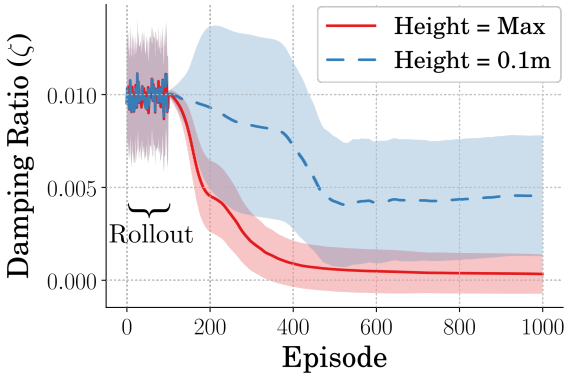

Figure 11. Damping Ratio Selected During Training

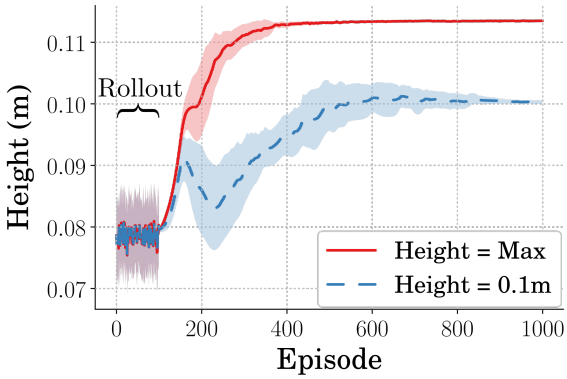

Figure 12. Height Reached During Training

6.3. Design Learned Given Broad Design Space

Figure 12 shows the height achieved by the learned designs for the agents given a wider range of damping ratios. For the agents learning designs to maximize jump height, Figure 12 can be compared with Figure 7 showing that the agents learned a design nearing one which would achieve maximum performance. Additionally, looking at the agents learning designs to jump to the specified 0.01 m, the designs learned accomplish this, only with slightly more variance than what is seen in the maximum height agents. Figure 13 shows the rewards the agents received during training and support that both agent types learned designs which converged. In this case though, the agent learning a design to jump to a specific height requires more learning steps to converge.

The average and standard deviation of the spring constant and damping ratio design parameters the agents selected during training are shown in Figures 14 and 15. For the agents that learned designs to jump to a specified height, it can be seen that there is a high variance in spring constant throughout training. However, the majority of agents

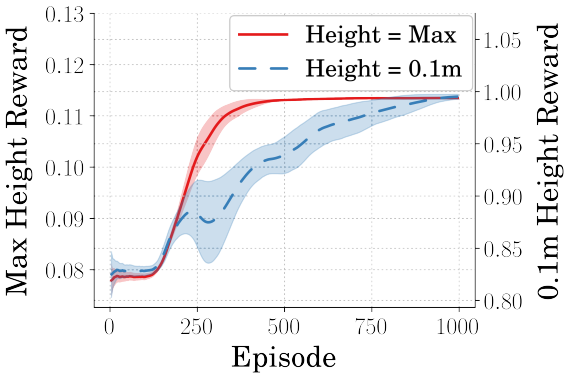

Figure 13. Reward Received During Training

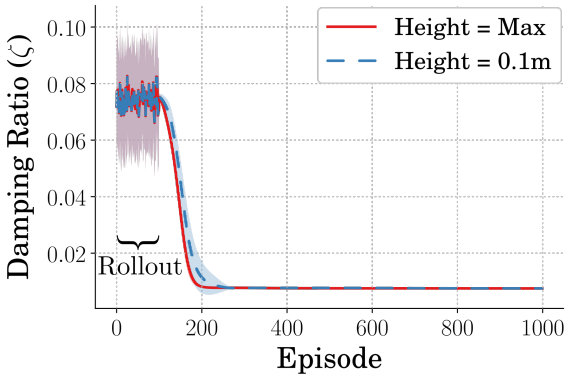

Figure 15. Damping Ratio Selected During Training

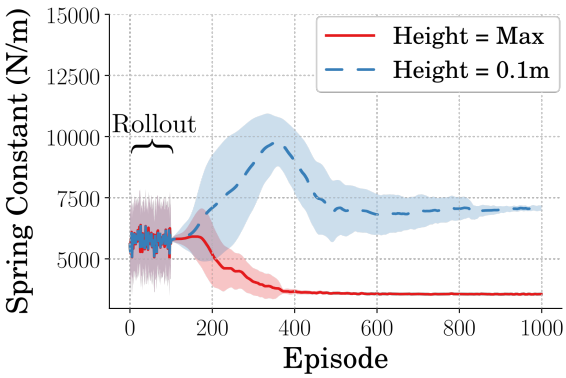

Figure 14. Spring Constant Selected During Training

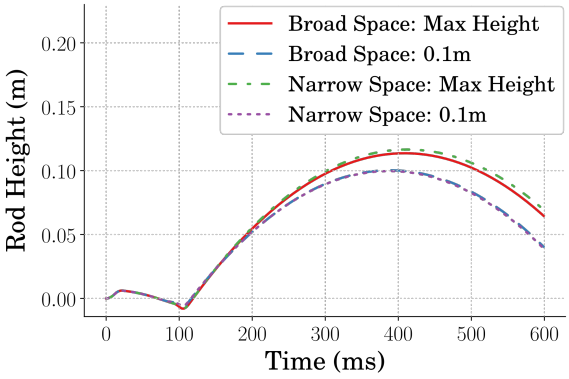

Figure 16. Height vs Time of Average Optimal Designs

converge to a specific design, lowering the variance. The same can be seen in the damping ratio; however, the variance is mitigated significantly earlier in training. The agents which were learning designs that maximized height found them with very little variance in terms of spring constant and damping ratio.

6.4. Average Design Performance

The final mean and standard deviation of the design parameters for the two different cases are presented in Table 4. Figure 16 shows the jumping performance of the mean designs learned for both cases tested. The agents tasked with finding designs to jump to the specified 0.01 m, did so with minimal error. The difference seen in maximum height reached between the two cases represents the difference in the damping ratio design space the agents had access to. The peak heights achieved can be compared again to Figures 6 and 7 to show that the agents learned designs nearing those achieving maximum performance.

7. Conclusion

The monopode model was used in conjunction with a predetermined control input to determine if a reinforcement learning algorithm (TD3) could be used to find optimal performing design parameters regarding jumping performance. This work was done in part to determine if reinforcement learning could be used as the mechanical design learner for an intelligent concurrent design algorithm. It was shown that when providing an agent with a design space that was smaller in size, the agents performed well in finding design parameters which met the performance constraints. The designs found were high in design variance, however. It was additionally shown that when provided with larger design space, the agents excelled at finding design parameters which were lower in design variance but still met the design constraints.

Table 4. Learned Design Parameters

Training Case		Design Parameter	Mean	STD
Narrow Design Space	Learn Max Height	Spring Constant	3.62e03	3.82e01
		Damping Ratio	3.37e-04	2.11e-03
	Learn Specified Height	Spring Constant	7.74e03	1.24e03
		Damping Ratio	4.55e-03	6.49e-03
Broad Design Space	Learn Max Height	Spring Constant	3.55e03	4.86e01
		Damping Ratio	7.53e-03	8.86e-06
	Learn Specified Height	Spring Constant	7.07e03	2.16e02
		Damping Ratio	7.54e-03	3.27e-05