# 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



## 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 [5, 12, 25]. 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 [5, 18, 29]. 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 [13]. These, however, are still not representative of fully flexible links, which have been shown to drastically improve locomotive performance measures such as running speed [19].

Control methods have been developed that work well for flexible systems like the ones mentioned [16, 17]. 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 [2, 26]. 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 [8]. 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 [10]. 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 [6]. Using this technique for legged-locomotion has also been studied, but thus far has been limited to the case of rigid systems [20].

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 [12, 25]. Previous work has shown that the use of flexible components in the legs of legged locomotion systems increase performance while decreasing power consumption [19]. Research also has been done showing the uses of series-elastic-actuators for locomotive systems [18]. In much of this work, human interaction with the robotic systems is considered such that rigidity is not ideal [29]. 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 [1, 16]. 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 [11]. 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 [15] have been used to train running strategies for a flexible legged quadruped [8]. 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 [14]. More recent work has used advanced methods such as evolutionary strategies to define robot design parameters [28]. In addition to evolutionary strategies, reinforcement learning has been shown to be a viable solution for concurrent design of 2D simulated locomotive systems [10]. This is further shown to be a viable method by demonstrating more complex morphology modifications in 3D reaching and locomotive tasks [20]. However, these techniques have not yet been applied to flexible systems for locomotive tasks.

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, $\omega_n$ | $\sqrt{\frac{\alpha}{m_l+m_a}}$ |
| 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$^2$ |

## 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 [3]. 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 $\alpha$ 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 $\alpha$ and $c$ represent the nonlinear spring and damping coefficient, respectively, and constant $\beta$ is set to $1e8$. Ground contact determines the value of $\gamma$, 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.

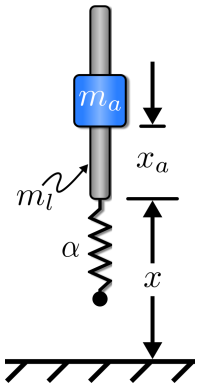


Figure 1. Monopode System

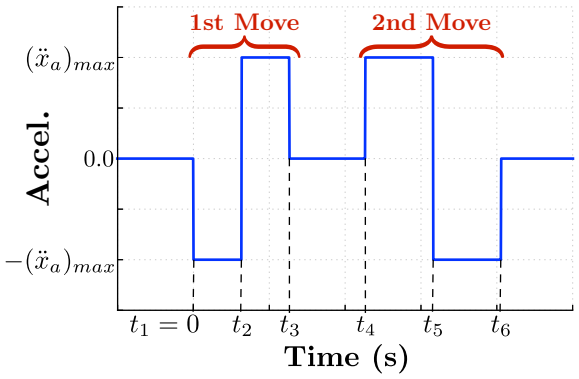


Figure 2. Jumping Command

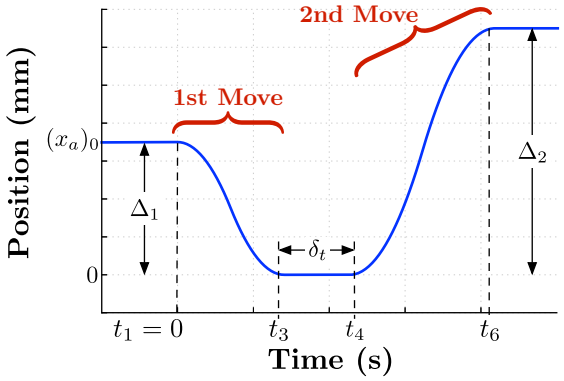


Figure 3. Resulting Actuator Motion

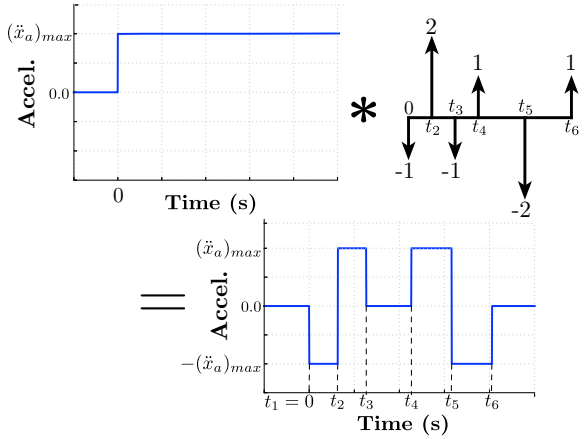


Figure 4. Decomposition of the Jump Command into a Step Convolved with an Impulse Sequence

## 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 [27]. 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 $\Delta_1$, pauses there for $\delta_t$, then moves a distance $\Delta_2$ 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 [24]. Using this decomposition, input-shaping principles and tools can be used to design the impulse sequence [22, 23]. 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 [27]. 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 [9, 21]. 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

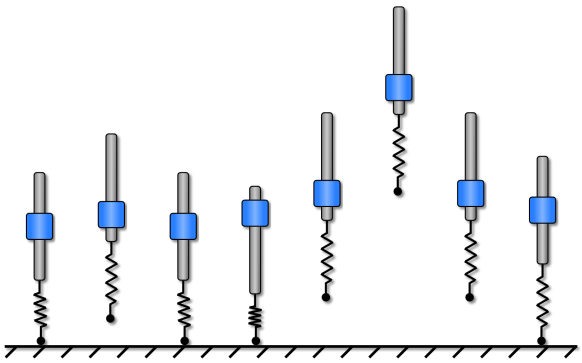

Figure 5. Example Stutter Jump

Table 2. TD3 Training Hyperparameters

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

of the algorithms was used as they are well written, popular in research, and thoroughly documented [7].

The training hyperparameters were selected based on the 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. The learning rate and N-steps parameters were selected

Table 3. PPO Training Hyperparameters

| Hyperparameter | Value |
|---|---|
| Learning Rate, $\alpha$ | 0.001 |
| Batch Size | 64 |
| Clip Range, $\epsilon$ | 0.2 |
| Gamma, $\gamma$ | 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, $\epsilon$ | 0.2 |
| Target Policy Clip, $c$ | 0.5 |
| Seed | 100 Random Seeds |

such that results converged in a similar amount of episodes as TD3.

### 5.2. Training Environment Design

To allow the agent to find a mechanical design, a reinforcement learning environment conforming to the OpenAI Gym standard [4] 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:

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

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

where $\alpha$ and $\zeta$ 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:

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

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 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. Algorithm Performance

### 6.1. Design Space Performance

Figures 6 represents the height the monopode could reach for the design/action space. The design space provided 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. Additionally, it can be used to visually determine the gradient of the jumping height in regards to the design parameters, and shows that the optimal results should occur around a 4000 N/m spring constant and the minimum damping ratio.

### 6.2. Design Learned

Figure 7 shows the rewards the agents received during training and support that both algorithm types learned designs which converged. At 1000 episodes the mean rewards of both algorithms are similar, both within one standard deviation of either algorithm's performance. However, PPO shows to be the preferable algorithm as it has a significantly

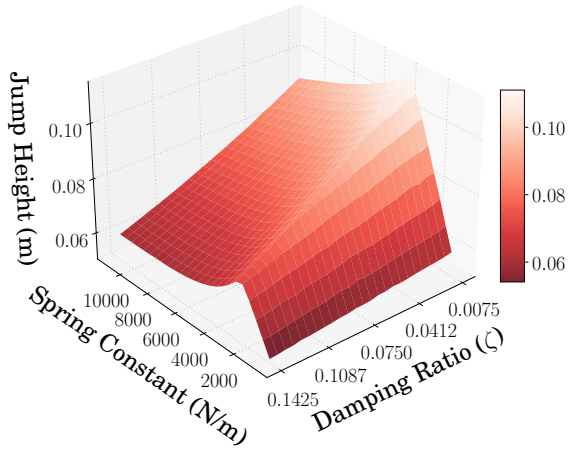


Figure 6. Jumping Performance of Broad Design Space

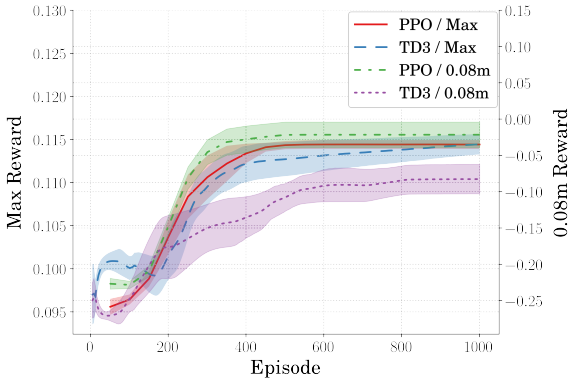


Figure 7. Reward Received During Training

smaller standard deviation. This means that fewer agents would need to be trained using PPO, as training an agent using the PPO algorithm will more consistently lead to the algorithms optimal result.

what are the differences though?

what are the differences though?

The average and standard deviation of the spring constant and damping ratio design parameters the agents selected during training are shown in Figures 8 and 9. These figures show that the PPO and TD3 Algorithm on average converge to a similar result. However, the variation in the standard deviation of the parameter choices between the PPO agents, show that each agent's parameter choice varied greatly between episodes. This implies a higher preference for exploration in the PPO agents than the TD3. The larger standard deviation of the TD3 algorithms does not demonstrate the algorithm's preference for exploration, instead it

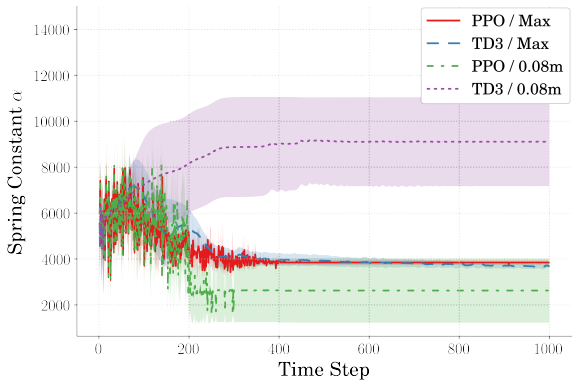


Figure 8. Spring Constant Selected During Training

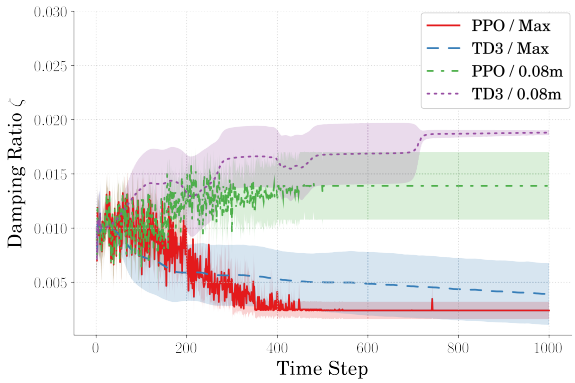


Figure 9. Damping Ratio Selected During Training

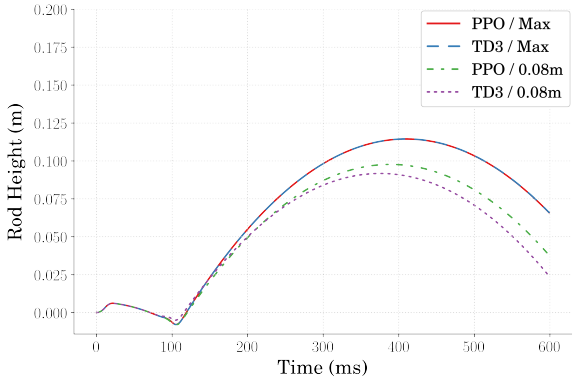


Figure 10. Response of Average Optimal Designs

shows that agents trained with TD3 algorithm converged on solutions with more significant differences.

### 6.3. Average Design Performance

Figure 10 shows the jumping performance of the mean designs learned for both algorithms tested. The peak heights achieved can be compared again to Figure 6 to show that the agents learned designs nearing those achieving maximum performance.

## 7. Conclusion

This needs some rewriting.

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.

## References

[1] Mojtaba Ahmadi and Martin Buehler. Stable control of a simulated one-legged running robot with hip and leg compliance. *IEEE Transactions on Robotics and Automation*, 13(1):96–104, 1997. 2

[2] Sarthak Bhagat, Hritwick Banerjee, Zion Tsz Ho Tse, and Hongliang Ren. Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges. *Robotics*, 8(1):1–36, 2019. 1

[3] R. Blickhan and R. J. Full. Similarity in multilegged locomotion: Bouncing like a monopode. *Journal of Comparative Physiology A*, 173(5):509–517, 1993. 2

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. pages 1–4, 2016. 4

[5] Gabriele Buondonno, Justin Carpentier, Guilhem Saurel, Nicolas Mansard, Alessandro De Luca, and Jean Paul Laumond. Actuator design of compliant walkers via optimal control. *IEEE International Conference on Intelligent Robots and Systems*, 2017-Septe:705–711, 2017. 1

[6] Tianjian Chen, Zhanpeng He, and Matei Ciocarlie. Hardware as Policy: Mechanical and computational co-optimization using deep reinforcement learning. *arXiv*, (CoRL), 2020. 1

[7] Antonin Raffin Dormann, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(22):1–8, 2021. 4

[8] Zach Dwiel, Madhavun Candadai, and Mariano Phielipp. On Training Flexible Robots using Deep Reinforcement Learning. *IEEE International Conference on Intelligent Robots and Systems*, pages 4666–4671, 2019. 1, 2

[9] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *35th International Conference on Machine Learning, ICML 2018*, 4:2587–2601, 2018. 3

[10] David Ha. Reinforcement learning for improving agent design. *Artificial Life*, 25(4):352–365, 2019. 1, 2

[11] Wei He, Hejia Gao, Chen Zhou, Chenguang Yang, and Zhijun Li. Reinforcement Learning Control of a Flexible Two-Link Manipulator: An Experimental Investigation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–11, 2020. 2

[12] Jonathan Hurst. The Role and Implementation of Compliance in Legged Locomotion. *The International Journal of Robotics Research*, 25(4):110, 2008. 1, 2

[13] Fumiya Iida, Gabriel Gómez, and Rolf Pfeifer. Exploiting body dynamics for controlling a running quadruped robot. *2005 International Conference on Advanced Robotics, ICAR '05, Proceedings*, 2005:229–235, 2005. 1

[14] Q. Li, W. J. Zhang, and L. Chen. Design for control - A concurrent engineering approach for mechatronic systems design. *IEEE/ASME Transactions on Mechatronics*, 6(2):161–169, 2001. 2

[15] 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. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016. 2

[16] Zheng-hua Luo. Direct Strain Feedback Control of Flexible Robot Arms: New Theoretical and Experimental Results. 38(11), 1993. 1, 2

[17] A Mathematical Modeling. Gain Adaptive Nonlinear Feedback Control of Flexible SCARA / Cartesian Robots. (AIM):1423–1428, 2003. 1

[18] Gill A. Pratt and Matthew M. Williamson. Series elastic actuators. *IEEE International Conference on Intelligent Robots and Systems*, 1:399–406, 1995. 1, 2

[19] U Saranli, M Buehler, and D E Koditschek. RHex: A Simple and Highly Mobile Robot. *International Journal of Robotics Research*, 20(7):616–631, 2001. 1, 2

[20] Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:9798–9805, 2019. 1, 2

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. pages 1–12, 2017. 3

[22] N C Singer and W P Seering. Preshaping Command Inputs to Reduce System Vibration. *Journal of Dynamic Systems, Measurement, and Control*, 112(1):76–82, 1990. 3

[23] W Singhose, W Seering, and N Singer. Residual Vibration Reduction Using Vector Diagrams to Generate Shaped Inputs. *Journal of Mechanical Design*, 116(2):654–659, 1994. 3

[24] Khalid L Sorensen and William E Singhose. Command-induced vibration analysis using input shaping principles. *Autom.*, 44:2392–2397, 2008. 3

[25] Yuuta Sugiyama and Shinichi Hirai. Crawling and jumping of deformable soft robot. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4(c):3276–3281, 2004. 1, 2

[26] Thomas George Thuruthel, Egidio Falotico, Federico Renda, and Cecilia Laschi. Model Based Reinforcement Learning for Closed Loop Dynamic Control of Soft Robotic Manipulators. *IEEE Transactions on Robotics*, 35:124–134, 2019. 1

[27] Joshua Vaughan. Jumping Commands For Flexible-Legged Robots. 2013. 3

[28] Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Towards efficient automatic robot design. *arXiv*, pages 1–17, 2019. 2

[29] Ting Zhang and He Huang. Design and Control of a Series Elastic Actuator with Clutch for Hip Exoskeleton for Precise Assistive Magnitude and Timing Control and Improved Mechanical Safety. *IEEE/ASME Transactions on Mechatronics*, 24(5):2215–2226, 2019. 1, 2

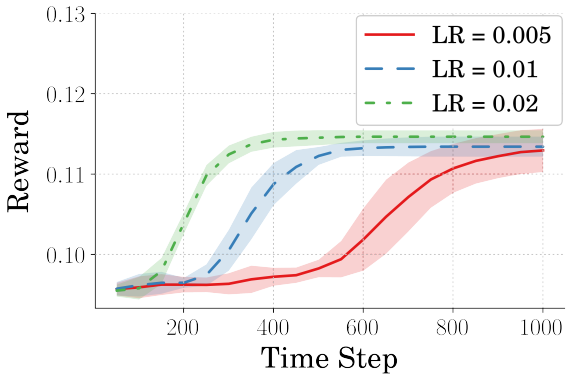


Figure 11. Reward Due to Changes in N Steps

## A. Learning Rate

In the process of this project the effects of learning rate on the PPO and TD3 Algorithms were explored. Figures 11 and 12 show that the Time Steps needed for the PPO algorithm to converge is very sensitive to the choice in learning rate, it also shows that the overall mean and standard deviation of the rewards is sensitive. On the other hand, Td3 does not appear to be at all sensitive to changes in the learning rate, at least not in the range of learning rates explored in this study.

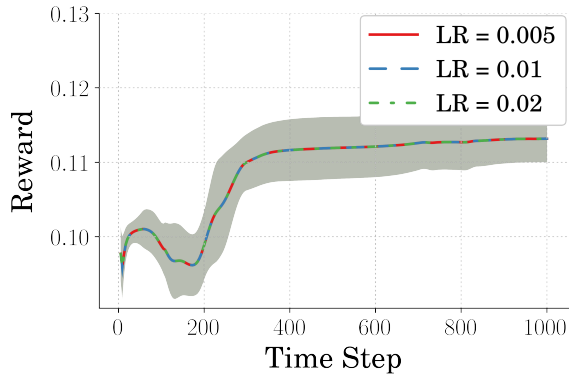


Figure 12. Reward Due to Changes in N Steps

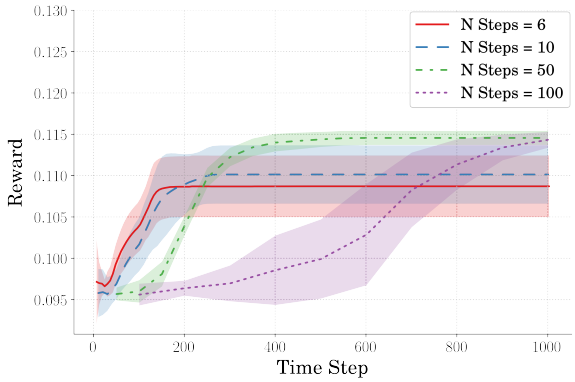


Figure 13. Reward Due to Changes in N Steps

## B. Update Rate

In addition to learning rate the effects of the update rate on PPO was explored. This study found an update rate of one update per 50 time steps to be optimum. The results shown in Figure 13 show that an update rate of 100 is to slow as the algorithm is unable to converge by the 1000th Time Step. The figure also shows that updating every 10 steps is also a poor choice. This is because the algorithm is forced to make updates on small batch sizes.