Mechanical Design and Control of Flexible-Legged Jumping Robots

A Thesis

Presented to the

Graduate Faculty of the

University of Louisiana at Lafayette

In Partial Fulfillment of the

Requirements for the Degree

Masters of Science

Andrew Albright

Spring 2022

Mechanical Design and Control of Flexible-Legged Jumping Robots

Andrew Albright

APPROVED:

Joshua E. Vaughan, Chair
Assistant Professor of Mechanical Engineering

Anthony S. Maida
Associate Professor of Computer Science

Alan A. Barhorst
Department Head of Mechanical Engineering

Mary Farmer-Kaiser
Dean of the Graduate School

To all the poor souls using Word, one day you will see the light that is LaTeX.

“Before we work on artificial intelligence why don’t we do something about natural stupidity?”

— Steve Polyak

Acknowledgments

I would like to firstly thank my advisor Dr. Joshua Vaughan for his leadership, both academic and personal, during my time here at the university. His support has been invaluable in helping me to understand the underlying ideas behind this material. Additionally, his constant drive to produce high quality material has kept me motivated in my attempts to do the same. I would not be here without him.

Additionally, I would like to thank my committee members, Dr. Alan Barhorst, Dr. Anthony Maida and Dr. Brian Post for their input in regards to this work. Along with my lab members during my time in the C.R.A.W.L.A.B., Gerald Eaglin, Adam Smith, Y (Eve) Dang, Brennan Moeller and Darcy LaFont. Their conversations and advice have greatly assisted me in my efforts to complete this work.

Lastly, I would like to thank the Louisiana Crawfish Board for their financial support in the form of a grant from the University of Louisiana at Lafayette. This grant has allowed me to work knowing my fiscal security was in tack.

Table of Contents

List of Tables

List of Figures

I Introduction and System Description

A legged locomotive robot can have many advantages over a wheeled or tracked one, particularly in regards to their ability to navigate uneven and unpredictable terrain [2–4]. They can achieve this advantage because of the numerous movement types they can deploy. Abilities such as independently placing their feet within highly rigid terrain and jumping or bounding over obstacles have been shown to be effective ways of locomoting []. These advantages do not come at no cost, however. Legged systems are traditionally power inefficient compared to wheeled vehicles making them a less attractive option for applications where power conservation is required. Research has been conducted showing the usefulness of adding flexible components, like the legs seen on the robot in Figure 1, for combating efficiency and other issues [4–7]. The addition of these components in legged robots has been shown to increase system performance measures such as running speed, jumping capability and power efficiency []. However, the addition of flexible components creates a system that is highly non-linear, and thus requires a more complex control system.

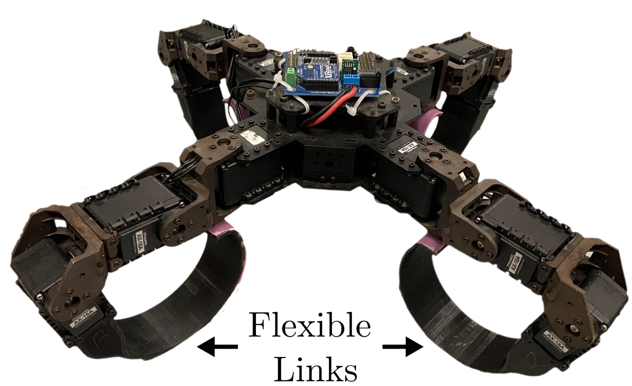


Figure 1. Flexible Robotics System

1.1 Improving Performance with Flexible Components

The addition of flexible components within robotic systems has been shown to be an effective way of improving performance metrics such as movement velocity and power efficiency [4, 7]. Of the different techniques that have be deployed the use of series elastic actuators (SEAs) has been shown to be very effective for increasing energy efficiency [8, 9]. Storing energy in the non-rigid parts of motor joints such as the spring seen in Figure 2 have proven to be an effective way in increasing efficiency. The addition of flexible joints is not the only technique that has been used to improve performance however; utilizing tendon like elastic members to connect actuators to links has also been shown to be an effective way of improving efficiency [1]. The use of tendons, being an example of replicating what is found in nature, is a common method

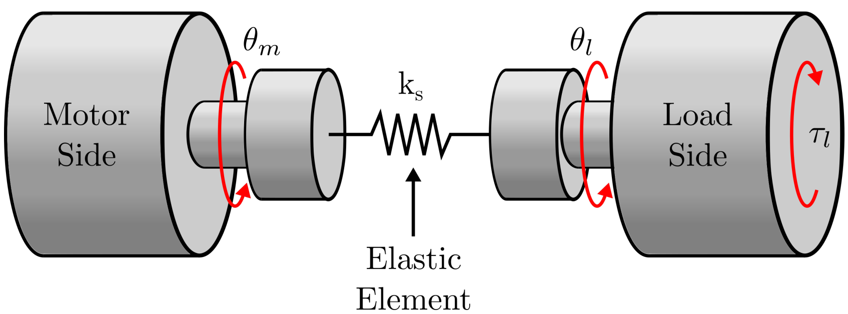


Figure 2. Rotary Style Series Elastic Actuator

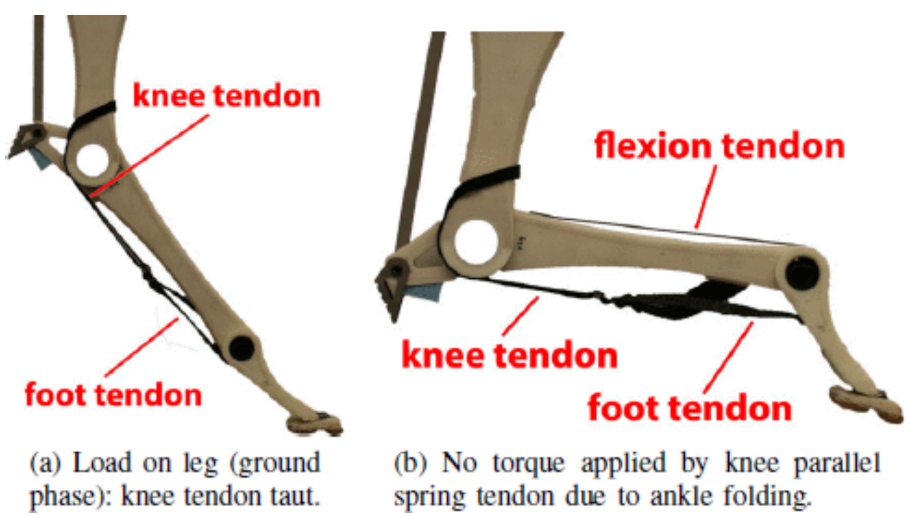


Figure 3. Tendon Like Flexibility from [1] This is blurry af.

of finding unique mechanical designs that perform well in the real world. An example of this type of design can be seen in Figure 3. Following a similar idea, research has also been conducted finding the usefulness of including flexibility in the the spine of of 2D running robots where the velocity of the robot was drastically increased [10].

1.2 Controlling Flexible Systems

Finding an optimal control strategy for flexible-legged jumping systems can be a challenge. . . .

Examples of traditional methods for controlling flexible systems.

1.3 Reinforcement Learning

Reinforcement Learning (RL) is the process of training an agent to define a series of commands using an environment where those commands can be applied. An agent in this work is synonymous with a controller and the environment the controller is deployed in can be thought of as a robotic system. Training the controller requires iteratively deploying the controller's commands, or actions, to the the environment and observing the results. The results are often in the form of the state of the environment and a reward resulting from the action that was applied. The reward is defined by the designer so that the controller is trained to accomplish a desired task. Other than the reward, the controller has no way to decern what commands are good when the

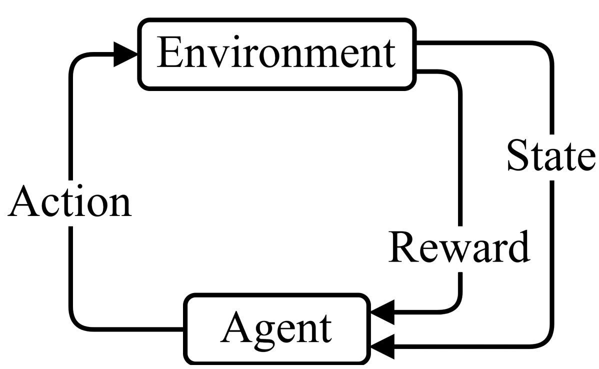


Figure 4. Reinforcement Learning Process

environment is in some state. The iterative process described is often shown in block diagram form and is shown in Figure 4.

Learning an optimal control strategy to complete a task is done so by deploying a learning algorithm utilizing information such as the state of the environment and the reward. For general robotics applications, at each discrete time step t, the environment will be in a state $s \in \mathcal{S}$, the controller will select an action $a \in \mathcal{A}$ according to the current policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ and apply said action within the environment. The environment will transition to a new state s' and will generate a reward r based on the users definition. The return is defined as a discounted sum or rewards, $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$, where γ is a discount factor for discerning between giving importance to near-term or long-term rewards.

The challenge of an RL algorithm is to optimize the policy, π_ϕ, with parameters, ϕ, such that actions generated at each time step will maximize the return. Ultimately an optimized policy will maximize the expected return, $J(\phi) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_0]$. Because many robotics controls processes are continuous control problems, the policy, π_ϕ, can be updated using gradient decent in terms of the parameters ϕ, which looks like $\nabla_\phi J(\phi)$. For actor-critic type architectures, the expected return can be represented using the Q-function:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s \sim p_\pi} \left[\nabla_a Q^\pi(s, a)|_{a=\pi(s)} \nabla_\phi \pi_\phi(s) \right] \tag{1}$$

where $Q^\pi(s, a) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_t | s, a]$ is the value function and the critic in the case of the TD3 architecture. Updating the Q-function can be completed by using temporal difference between the Q-function and a target Q-function [11, 12], which is based in the Bellman Equation:

$$Q^\pi(s, a) = r + \gamma \, \mathbb{E}_{s', a'} \left[Q^\pi(s', a') \right], \quad a' \sim \pi(s') \tag{2}$$

where $Q^\pi(s, a)$ can be represented and estimated using a differentiable function approximator, $Q_\theta(s, a)$, with parameters θ [13]. To maintain a fixed objective over multiple policy updates, the target Q-function approximator, $Q_{\theta'}(s, a)$, is used to update the policy and itself is updated to follow the main Q-function approximator be either matching the parameters or by polyak averaging, $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, where τ is a tunable hyperparameter.

1.4 Twin Delayed Deep Deterministic Policy Gradient

There are many algorithms used to tain a network based controller in a RL application, some of which have shown their ability to learn high performing control strategies for robotics systems [14–16]. Of the different algorithms used in research today, the one selected and tested in this work is Twin Delayed Deep Deterministic Policy Gradient (TD3) [17]. Figure 5 displays the general flow in information for this algorithm. A few things to note here are that this is an actor-critic type algorithm, it is an off-policy algorithm, and the state and action spaces are continuous within their respective ranges. . . .

Do I need to describe it in detail here? If so how much?

There are many implementations of the TD3 algorithm that are available, however the StableBaselines3 implementation is used to complete the work in this thesis [18].

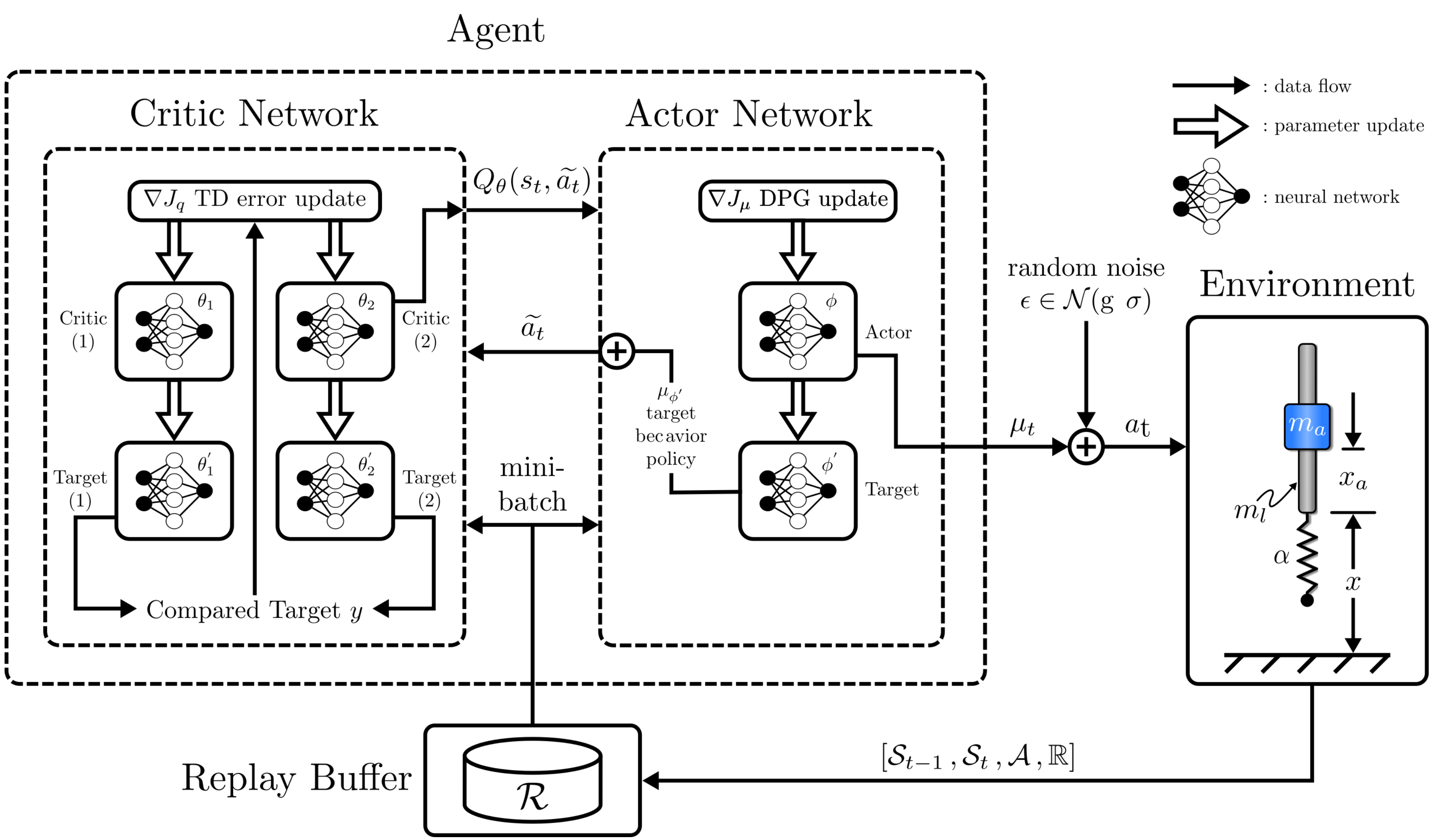


Figure 5. Twin Delayed Deep Deterministic Policy Gradient Block Diagram with Monopode as Environment

II Learning Efficient Jumping Strategies for the Monopode System

Utilizing reinforcement learning to (RL) train a neural network based controller has been shown to be useful for controlling many robotic systems [15, 16]. Successful control of rigid legged robots both in simulation and on physical hardware has been shown to be highly effective [14, 19]. However, the use of RL to train controllers for flexible systems is limited, particularly in regards to legged locomotive systems. RL has been shown to be capable of defining more effective and efficient jumping techniques for a single-legged robot with SEAs [20]. . . .

Some information regarding controlling flexible robots

2.1 Monopode Jumping System

The intent of the work completed in this thesis is to validate the use of RL to concurrently design a system/controller architecture for flexible-legged locomotive systems. To evaluate the methods used, a monopode system like the one shown in Figure 6 was used to represent a flexible jumping system. This system has been studied

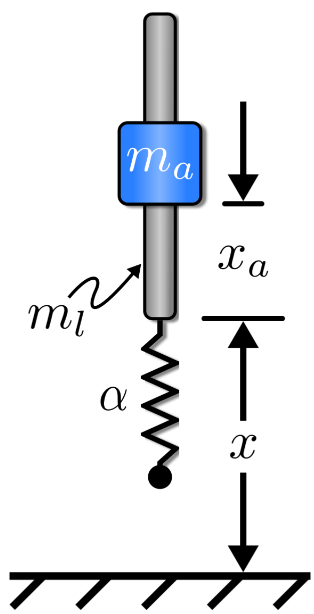


Figure 6. Monopode Jumping System

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 $\frac{\mathrm{N}}{\mathrm{m/s}}$
Gravity, g	9.81 m/s^2
Actuator Stroke, $(x_a)_{\max}$	0.008 m
Max. Actuator Velocity, $(\dot{x}_a)_{\max}$	1.0 m/s
Max. Actuator Acceleration, $(\ddot{x}_a)_{\max}$	10.0 m/s^2

and has been proven to be an effective base for modeling the jumping gaits for many different animals [21].

The monopode is controlled by accelerating the actuator mass, m_a, along the rod mass, m_l, causing a hopping like motion. A nonlinear spring is modeled and represented by the variable α in the figure. Also included in the model is a damper parallel with the spring, having a damping coefficient of c, though it is not shown in the figure. Variables x and x_a represent the rod's global position and the actuator's local position with respect to the rod, respectively. The equations of motion for the system are:

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

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 linear 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:

$$\gamma = \begin{cases} -1, & x \leq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Additionally, the spring compression limit, or the systems position in the negative x

direction, is limited to 0.008m. Additionally, the system is confined to move only vertically in regards to Figure 6 so that controlling ballance is not required.

2.2 Training Environment

In this Chapter, RL is used to find efficient control strategies for a flexible-legged robotic system. The monopode described in Ch. 1 was used as a test system to validate the effectiveness of the RL approach. An traditional RL environment aligning with the standards set by OpenAI for a Gym environment was created. []. The observation and action spaces were defined, respectively, as follows:

$$\mathcal{S} = [x_{a_t}, \dot{x}_{a_t}, x_t, \dot{x}_t] \tag{5}$$

$$\mathcal{A} = [\ddot{x}_{a_t}] \tag{6}$$

where x_t, $\dot{x}_t$ were the monopode's position and velocity at time t, and x_{a_t}, $\dot{x}_{a_t}$ and $\ddot{x}_{a_t}$ were the actuator's position, velocity and acceleration, respectively.

Two separate stopping conditions were defined for the environment to evaluate two different jump types and therefore two different input commands. The first was defined as the monopode's position being greater than zero than returning to zero once. The second was defined like the first but with the monopode's position being greater than zero and then less than zero twice. Two different jumps are create from these stopping conditions and they are highlighted in Section 2.4 below.

2.3 Efficient Control Strategies

Efficient control of a robotic system is often one of the most important aspects of a controller's design. Applications where a robotic system is deployed and relies on a limited power source, such as a mobile walking robot, will often require an efficient control strategy. Modern, traditional methods, such as model predictive control, have been shown to produce energy efficient locomotion strategies for wheeled and legged systems [22, 23]. It is of interest in this work to utilize RL, a modern neural network based control method, to find strategies which are designed with power efficiency as the primary objective.

Two different reward functions were designed to accomplish the task of determining how well RL learns efficient jumping strategies. The purpose of defining two different reward functions was to compare the input commands and resulting jumping shapes of the two controller types to determine if the efficient controller was learning to conserve power.

The first reward function was one that ignored power usage all together and focused solely on the height of the jump:

$$R = x_t \tag{7}$$

where x_t was the height of the monopode system at any given time step. The second reward function was one that was defined to accomplish the same task, but also consider power consumption and was defined as:

$$R = \frac{x_t}{\sum_{t=0}^{t} P_t} \tag{8}$$

where P_t was the power consumption of the monopode system at any given time step defined mechanically as the product of the actuator's acceleration, velocity and mass:

$$P_t = m_a \, \dot{x_a} \, \ddot{x_a} \tag{9}$$

where m_a was the mass of the actuator, and $\dot{x_a}$ and $\ddot{x_a}$ where the actuators velocity and acceleration, respectively.

2.4 Input Complexity

Two different jumping types were also analyzed. The first was referred to as a single jump command, and the second a stutter jump command. The intent of utilizing two different jumping commands was to determine if a RL algorithm was more or less effective in learning differing strategies depending on the complexity of the desired command.

An example single jump can be seen in Figure 7. The intended command from the learned controller would be one that would jump the monopode once. This type of command would ideally compress the spring/damper by accelerating the actuator in the positive direction. This would allow the system to store some energy in the spring that could be used to jump the system. The actuator mass should then accelerate downward

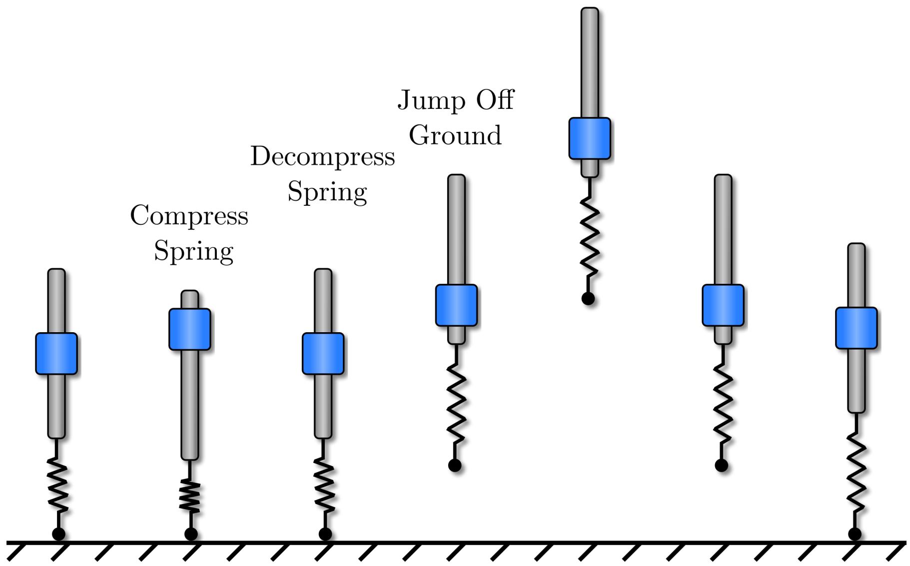


Figure 7. Example Single Jump

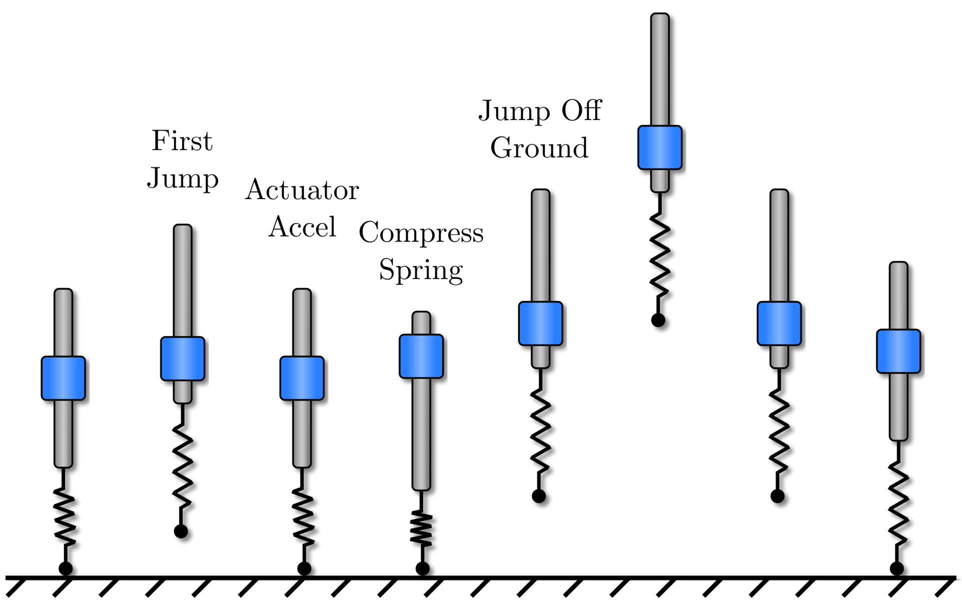


Figure 8. Example Stutter Jump

allowing the spring to decompress until it reaches its nominal length. At this point the system should be accelerating upwards and the actuator downwards such that the monopode leaves the ground completing a single jump.

An example stutter jump can be seen in Figure 8. The intended command from the learned controller would be one that would jump the monopode twice. This type of command would firstly complete an optimal single jump. Following that motion, the actuator should assume an acceleration direction to recompress the spring storing more energy with a farther compression. When the spring is compressed to its maximum value or the system's total acceleration reaches zero, the actuator mass should accelerate downwards allowing the spring to decompress until it again reaches its nominal length. At this point the system should be accelerating upwards and the actuator downwards, similar to the single jump, such that the monopode leaves the ground completing a stutter jump.

2.5 Deploying TD3

Because training an RL controller does not guarantee that the controller will learn an optimal strategy without finding local optima, training more than one

Table 2. TD3 Training Hyperparameters

Hyperameter	Value
Learning Rate, α	0.001
Learning Starts	1000 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	50 Random Seeds

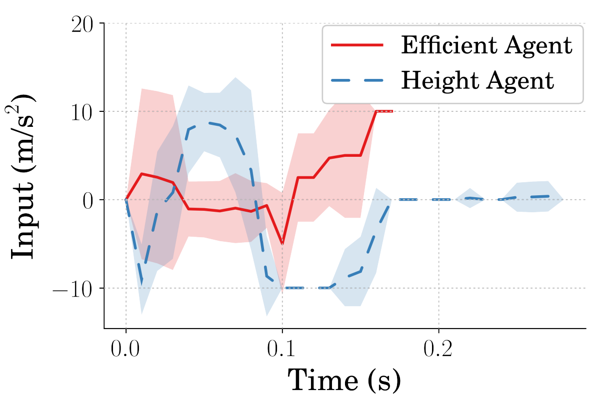
controller is often practiced to evaluate performance. In this work, fifty different controllers where trained, each with a different random network initialization. Each controller was trained for a total 500k time steps. The remaining hyperameters set using the TD3 algorithm are defined in Table 3.

...

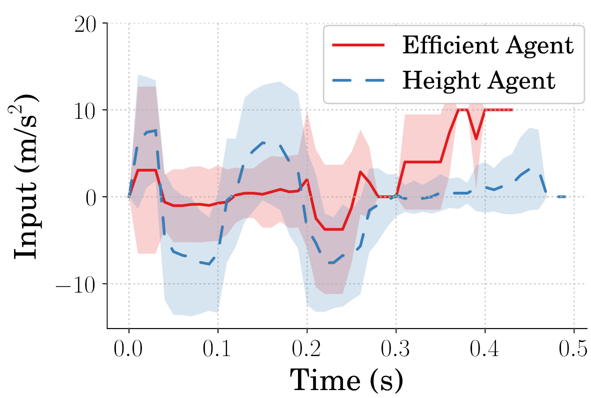
Get some training data here. Reward Primarily.

2.6 Average Performance of Network Controller

2.6.1 Input Commands. Average learned input commands for both the single and stutter jumping cases are shown in Figure 9 to compare controllers that were trained to jump efficiently to those trained to jump high. At first glance, there are obvious differences regarding timing, magnitude and direction. There are also slight differences in variance seen between the two controller types. Starting with Figure 9a, which displays the input commands for the single jumping case, it is most obvious that the direction for the initial acceleration of the actuator mass for the efficient controllers and height controllers differ. In the case where the controller is learning to jump high, an initial acceleration in the negative direction is learned, which contrasts the case where the controller is learning an efficient command. Further, the magnitude of the



(a) Single Jumping Input

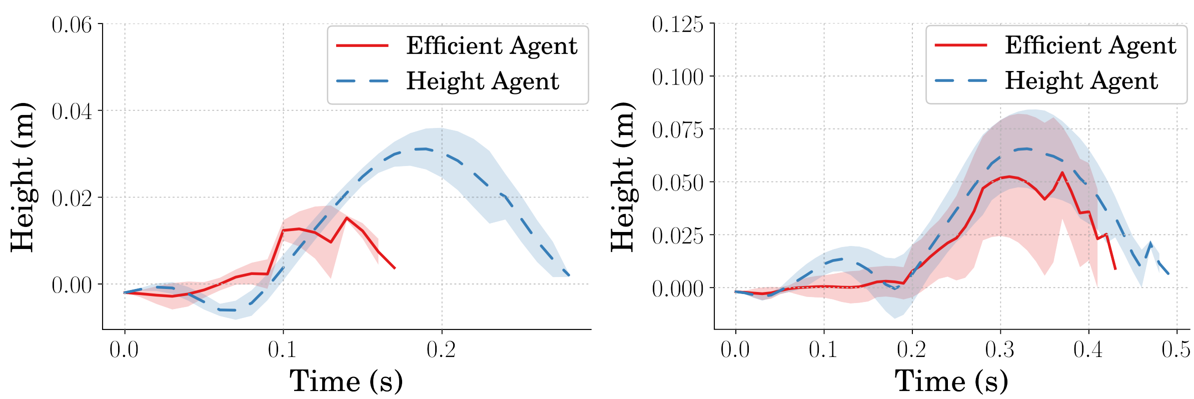


(b) Stutter Jumping Input

Figure 9. Average and Standard Deviation Inputs to monopode

commands is drastically different which may be an indicator for conserving power. Looking now at Figure 9b, it is immediately apparent that the magnitudes of the commands differ greatly. They are however, more similar in regards to their timings and directions when comparing the stutter jumping command to the single jumping command. In both the single and stutter jumping cases, it can be seen that there is upward acceleration command towards the end of the jump, which again might be an indicator of a more efficient jumping strategy. Furthermore, it can be observed that the single jumping case, there exists more variance across different instances of the trained efficient controllers in comparison to the height controllers. This does not seem to be the case for the stutter jumping command type, though both cases do seem to generate controllers with high variance inputs across instances.

2.6.2 Jumping Height Performance. Average jumping performance resulting from the learned single and stutter jumping commands is displayed in Figure 10 so that the efficient controllers might be compared with their height counterparts. In both the single and stutter jumping cases, there are differences seen in jumping ability when comparing the efficient and height controller types. Starting with Figure 10a, it is most apparent that the height controllers learned a command input

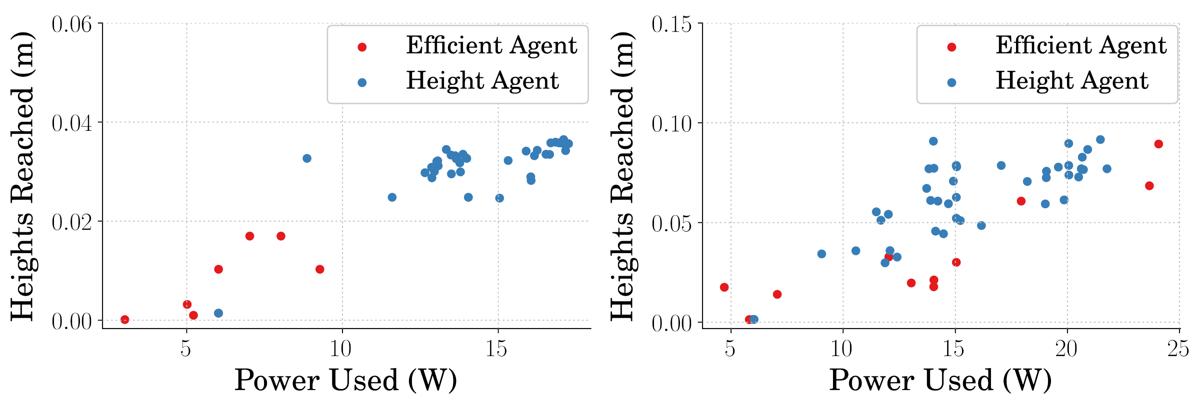


(a) Single Jumping Height

(b) Stutter Jumping Height

Figure 10. Average and Standard Deviation Heights of monopode

that was able to outperform the efficient controllers in terms of jump height. The resulting motion from the input discussed in the previous section can also be seen in that the efficient controller learns to simply compress the spring, then jump the monopode. The height controllers in contrast, disregarding power consumption, learned to decompress the spring from its nominal position, keeping it below the point of leaving the ground, then recompressing for a much higher jump. Figure 10b shows the stutter jumping controller's jumping performance, which are differing, but less drastic than the single jumping case. The likeness of the input command shapes propagate and the resulting jumping shapes share similar in form. The large differences seen in the stutter jumping shapes are similar those of the inputs that create them, in that they differ mostly regarding the magnitudes. In both the single and stutter jumping cases, the upward acceleration command seen in the efficient controllers toward the end of the command can be seen in that the position of the monopode creates a psuedo plato. Additionally, regarding variance, the single jumping controllers seems to produce jumping shapes with similar levels of variance. As for the stutter jumping case, though similar in high levels of input variance, the jumping height variance for the efficient controllers is noticeably higher that that of the height controller.



(a) Single Jumping Efficiency

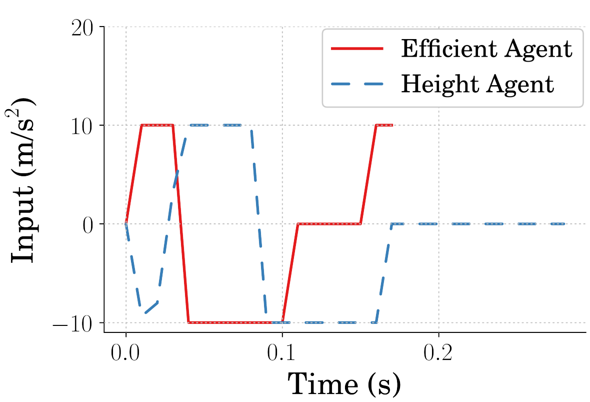
(b) Stutter Jumping Efficiency

Figure 11. Height Reached vs Power Consumed of monopode

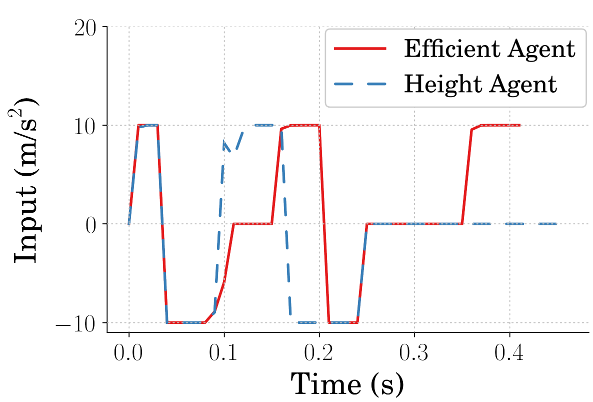
2.6.3 Height Reached vs. Power Used. Height reached versus power consumed data, for both the single jumping and stutter jumping cases is shown in Figure 11. It is firstly apparent that in both the single and stutter jumping cases, the efficient controllers utilize less power and therefore suffer regarding jump height. This matches what was seen in the previous sections discussing input command and jumping shapes. In the single jumping case, shown in Figure 11a, there is an apparent separation between the two controller types where the height controllers are mostly clustered in the upper-right part of the dataset. As for the stutter jumping case, show in Figure 11b, the difference is less obvious though still present. The variance of the two controller types, being quite high, matches what is seen in the previous sections and results in more mixing of the data.

2.7 Optimal Performance of Network Controller

2.7.1 Input Commands. Taking the best of the fifty different controllers trained for both the single and stutter jumping cases and comparing the efficient and height controller's performance can show what is possible with a properly defined RL problem. Figure 12, shows the differences in the input commands generated when selecting the highest performing controller in terms of reward received. At first glance,



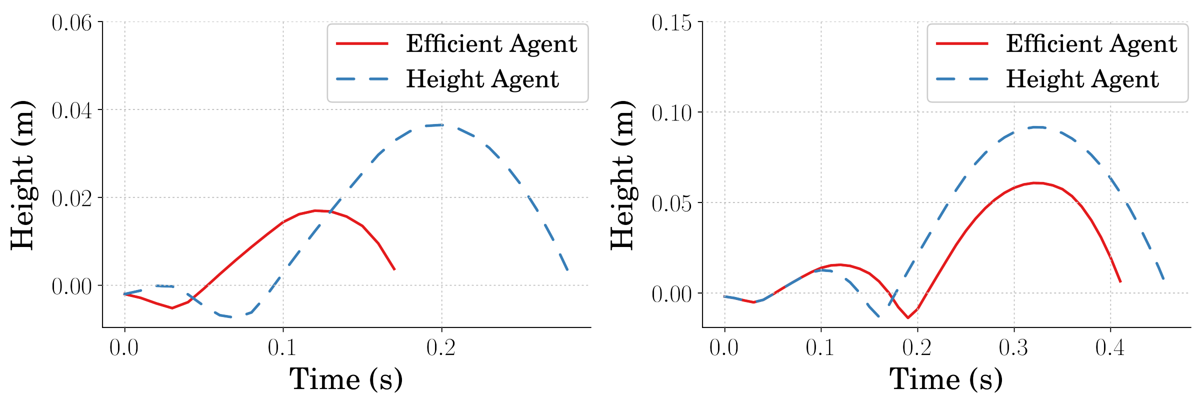
(a) Single Jumping Input



(b) Stutter Jumping Input

Figure 12. Optimal Inputs to monopode

it can be seen that there are less differences between the the efficient and height controllers in comparison to the average results from Section 2.6. A major similarity being that magnitudes are similar across all cases such that the controllers utilize the actuator's maximum acceleration. Looking at Figure 12a, which compares the efficient and height controllers for the single jumping input, the major difference are of course the timing and direction of the command. This is similar to the average performance evaluation, from Section 2.6.1, in that the efficient controller does not take advantage of the slight decompression of the spring before the monopode leaves the ground. Because of this, the efficient controller learns a different timing for a single jump. As for the stutter jumping case, shown in Figure 12b, the differences between the efficient and height controller is less drastic. The initial timing is largely the same as both controller types learn to utilize the decompression of the spring. The differences begin when decompressing the spring a second time and completing the first jump. The efficient controller learns a command similar in form to a bang-coast-bang command, where in contrast the height controller learns a command similar to that of a bang-bang shaped input.



(a) Single Jumping Height

(b) Stutter Jumping Height

Figure 13. Optimal Heights of monopode

2.7.2 Jumping Height Performance.

In line with the previous section, it is of interest to evaluate the jumping performance curves of the best controllers trained. Figure 13 displays the jumping performance for both jump types as well as both controller types when utilizing the inputs shown in Section 2.7.1. These curves validate what was shown in the previous section and display that the efficient controllers do not generate commands that jump the monopode as high. Figure 13a, which compares the efficient and height controllers for the single jump, verifies that the efficient controller does not learn to utilize the slack in the spring. In Figure 13b, it can be seen that when utilizing a command more similar in form to a bang-coast-bang command, like the efficient controller learned, the timing of the jump sequence is shifted and the resulting final height is less than the height controller who's command is more similar to a bang-bang shaped command. . . .

Get the percentage data on height and power used.

2.8 Conclusion

Two different controller types where trained to generate two different jumping commands for the simplified monopode jumping system described in Section ??. The first type of controller was one that would command the monopode system to jump high where the reward was based on nothing other than system height. The second type of controller was one which controlled the monopode to jump high but at the cost of power consumed, such that high jumps that consumed high amount of power where less desirable than high jumps that consumed less power. It has been shown that the rewards past to RL algorithms that are training controllers can be manipulated so that the the learned input commands take advantage of the spring/damper that exists within the monopode jumping system. Furthermore, the timing of the commands, the input magnitude and direction are all affected when defining a reward strategy that seeks to increase power efficiency. It should be the concluded then, that RL might serve as a useful method for defining control strategies for flexible-legged jumping systems, particularly when energy efficiency is of interest.

III Using Input Shaping to Validate RL Controller

Discussion on the use of input shaping to find optimal jumping strategies for jumping systems. How can one use these techniques to validate RL controllers? Do RL algorithms generate robust controllers when trained robustly. . . .

Include some discussion of: 1. Most RL having black boxes 2. Need for interpratable RL 3. Input shaping as a way to interpret RL

3.1 Input Shaping Controller Input

Information from the paper DV wrote and some other resources found from the sources in that paper. Will need some resources from DV to fill in some input shaping information.

3.2 RL Controller Input

Discussion and figures from the inputs defined by the RL algorithms for the monopode system.

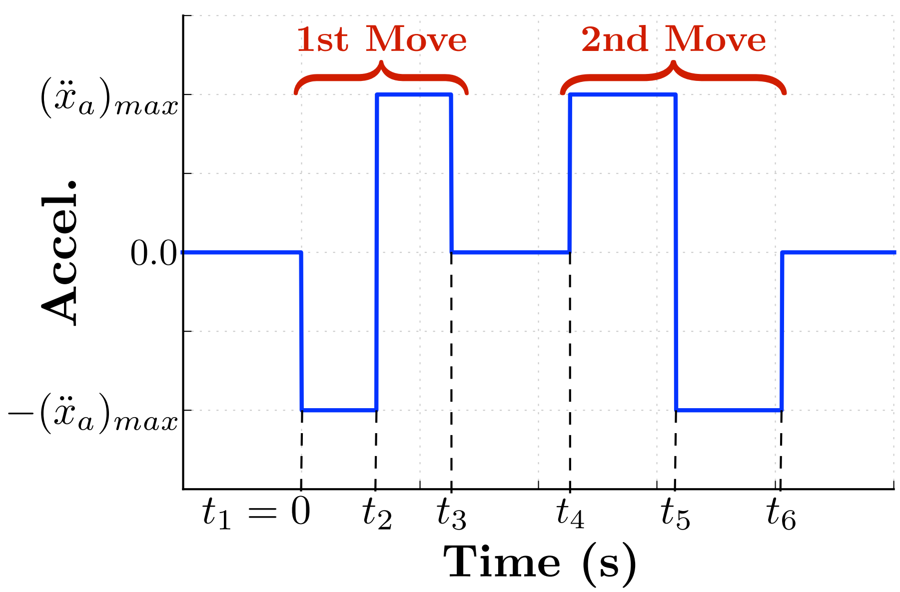


Figure 14. Jumping Command

3.3 Training a Robust Controller

How are robust controllers trained using RL algorithms? How did we do ours? Show the results.

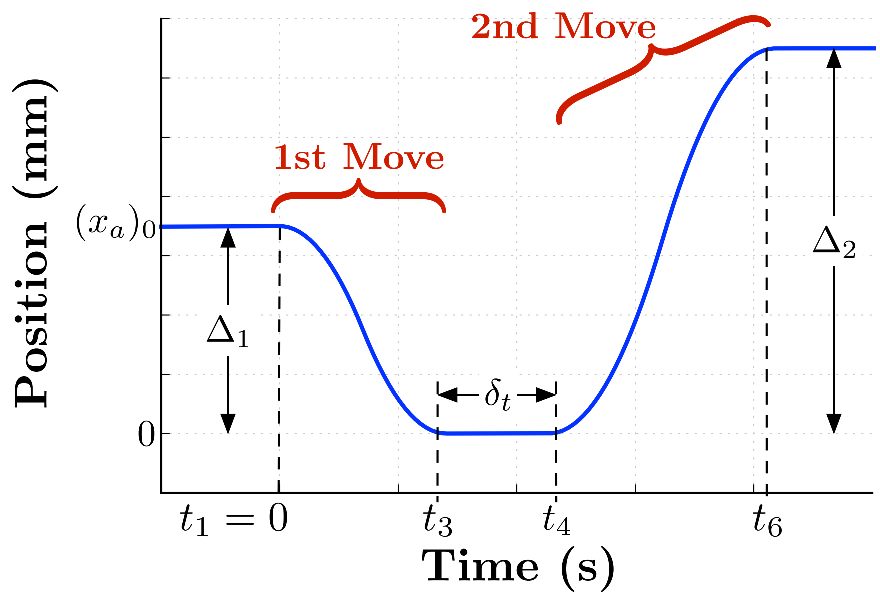


Figure 15. Resulting Actuator Motion

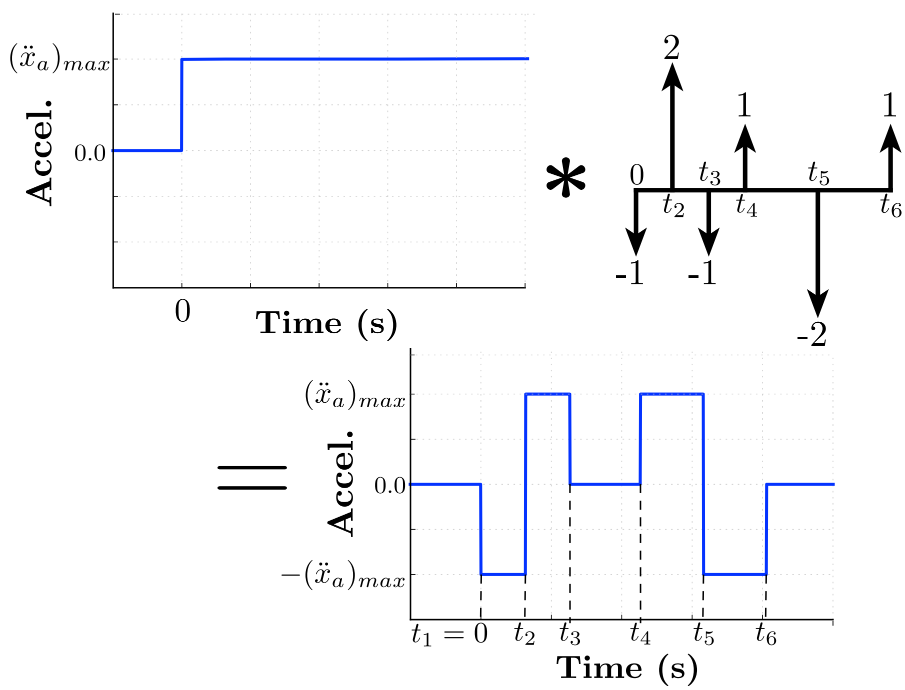


Figure 16. Decomposition of the Jump Command into a Step Convolved with an Impulse Sequence

3.4 Conclusion

Discuss the results.

IV Mechanical Design of a the Monopode Jumping System

Often it is the goal of a controls engineer to design a controller to accommodate and manipulate systems according to the system description provided. However, research has been conducted showing the value of studying the manipulation of mechanical design parameters in order to achieve a desired system behavior []. In this chapter, reinforcement learning is shown to be useful as a tool to learn mechanical designs given a predefined system controller for the monopode jumping system. . . .

Some discussion of work here. Include some evolutionary algorithm stuff.

[].

4.1 Controller Input

Bang-bang based jumping commands like the one discussed in Figure 14 of the Chapter 4 are likely to result in a maximized jump height [24]. For these command types, regarding the monopode jumping system, 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 was rod is shown in Figure 15. Starting from an initial position, x_{a_0}, the actuator 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 was shown in Figure 16 [25]. Using this decomposition, input-shaping principles and tools can be used to design the impulse sequence [26, 27]. 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 [24]. Commands of this form will often result in a stutter jump like what was shown in Figure 8 of Chapter 2, 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.

...

I think a bit of this belongs in the previous chapter?

4.2 Environment Definition

To allow the agent to find a mechanical design, a reinforcement learning environment conforming to the OpenAI Gym standard [28] was created for the monopode model described in Chapter 2, including a fixed controller input based on the algorithm described in Section 4.1. 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. The actions applied, $\mathcal{A}$, and transitions saved, $\mathcal{S}$, from the environment were defined as follows:

$$\mathcal{A} = \{\{a_\alpha \in \mathbb{R} : [-0.9\alpha, 0.9\alpha]\}, \{a_\zeta \in \mathbb{R} : [-0.9\zeta, 0.9\zeta]\}\} \tag{10}$$

$$\mathcal{S} = \left\{ \sum_{t=0}^{t_f} x_t, \ \sum_{t=0}^{t_f} \dot{x}_t, \ \sum_{t=0}^{t_f} x_{at}, \ \sum_{t=0}^{t_f} \dot{x}_{at} \right\} \tag{11}$$

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

4.3 Rewards for Learning Designs

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:

$$R_1 = \left(\sum_{t=0}^{t_f} x_t \right)_{max} \tag{12}$$

where x_t was the monopode's rod height at each step 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:

$$R_2 = \frac{1}{\frac{|\mathbb{R}_1 - x_s|}{x_s} + 1} \tag{13}$$

where x_s was the desired jump height, which was set to 0.01 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.

4.4 Ability to Learn a Design

Figures 17 and 18 represent the heights the monopode could reach for two different design spaces. The design space provided for the first case, shown in Figure 17, 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 18, 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.

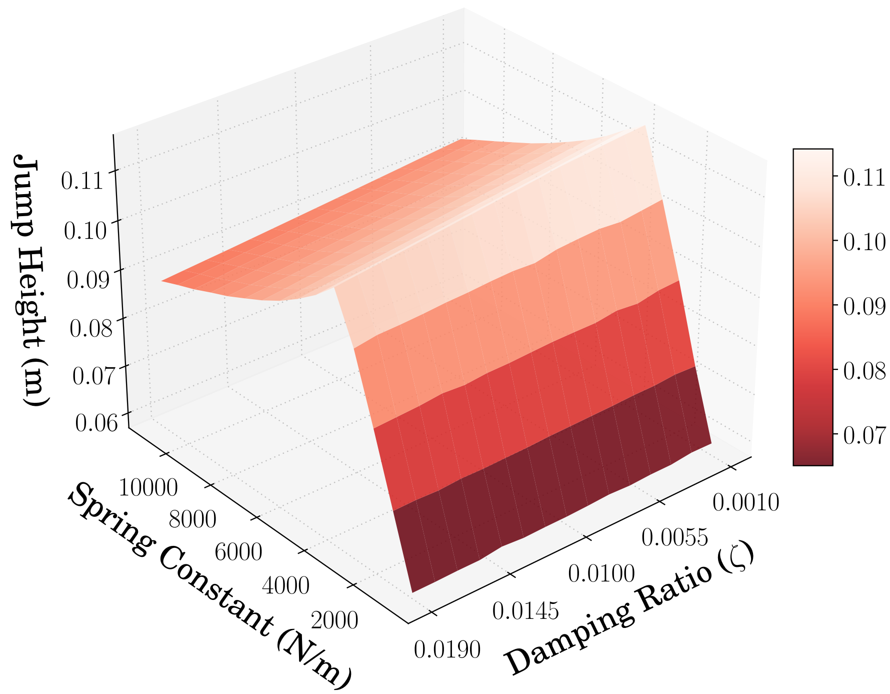


Figure 17. Jumping Performance of Narrow Design Space

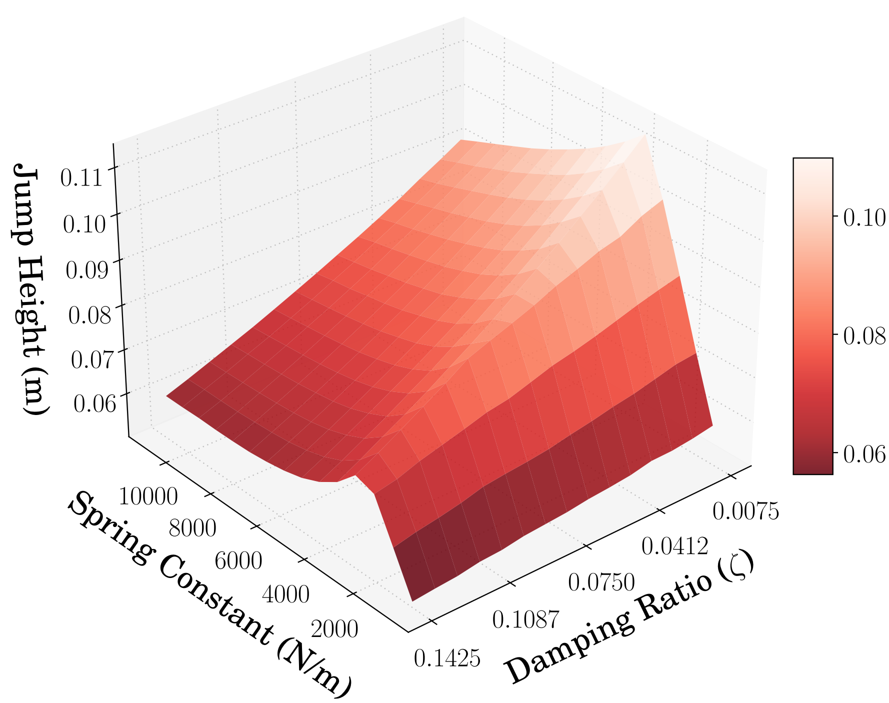


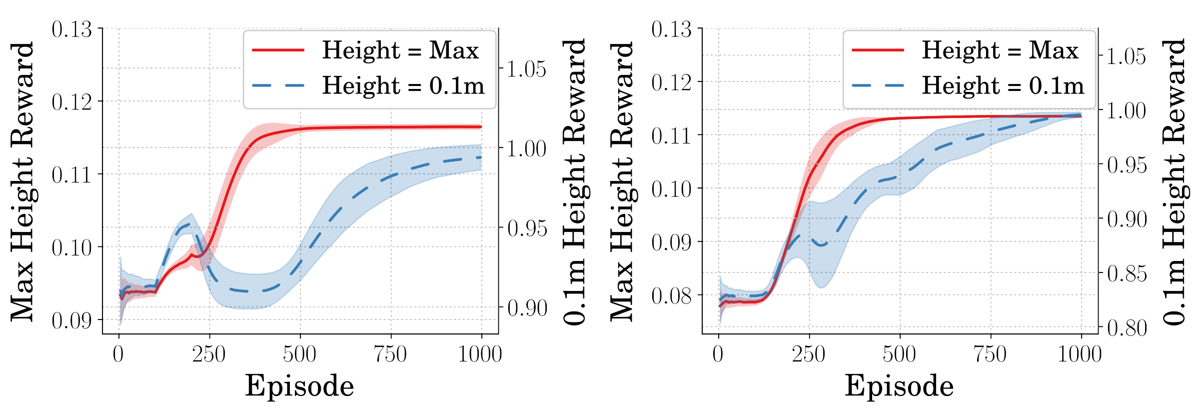
Figure 18. Jumping Performance of Broad Design Space

4.5 Deploying TD3

The training hyperparameters were selected based on TD3's author recommendations and Stable Baselines3 [18] experimental findings and are displayed in Table 3. All of the hyperparameters, with the exception of the rollout (Learning Starts) and the replay buffer, were set according to Stable Baselines3 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

Table 3. TD3 Training Hyperparameters

Hyperameter	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



(a) Reward vs. Episode: Narrow Design Space **(b)** Reward vs. Episode: Wide Design Space

Figure 19. Reward vs. Episode for Learning Mechanical Design

number of training steps due to system memory constraints.

The average rewards for both the narrow and the wide design space agents are shown in Figure 19. They represent the agents learning a converging solution to the problem of finding optimal design parameters. Looking at Figure 19a, it is apparent that given a more narrow design space, both the high and the specified jumping agents were still able to learn a converging solution. It can also be observed that there exists more variance for the specified height agent type compared to the height agents. Looking at Figure 19b, it is also apparent that the agents given a broader design space where both able to learn a converging design solution. Though given more designs to choose from, it appears the specified height agents are taking longer to search the design space, and towards the end, finding a designs with less variance.

4.6 Jumping Performance

4.6.1 Narrow Design Space. Figure 20 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 20 can be compared

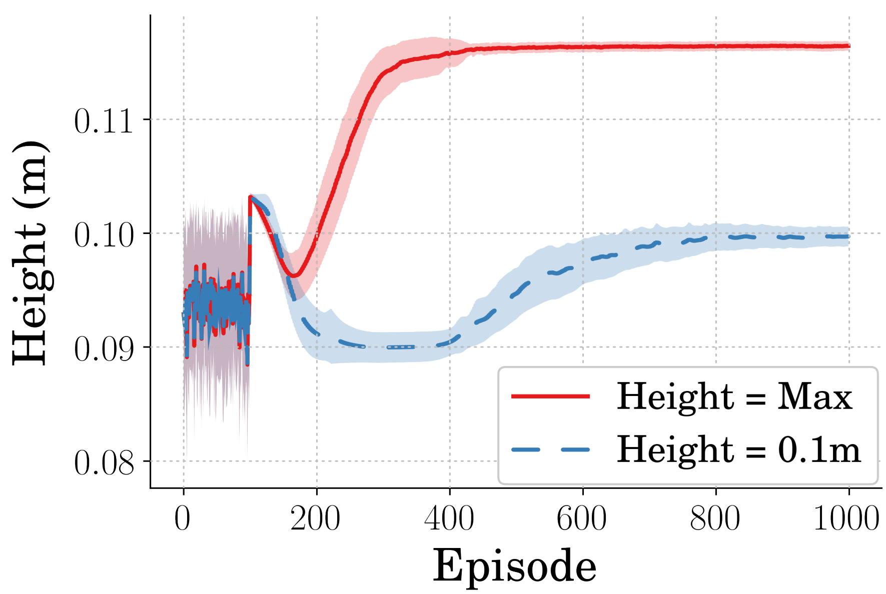


Figure 20. Height Reached During Training

with Figure 17 showing that the agent learned a design nearing one which would achieve maximum performance. Additionally, looking at the agents learning designs to jump to the specified 0.1 m, the designs learned accomplish this with slightly more variance than that of the maximum height case.

The average and standard deviation of the spring constant and damping ratio

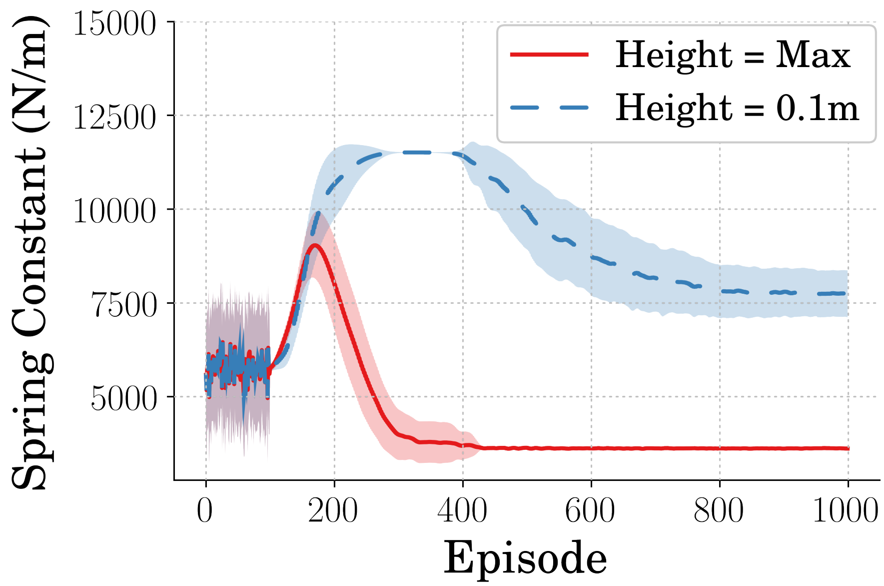


Figure 21. Spring Constant Selected During Training

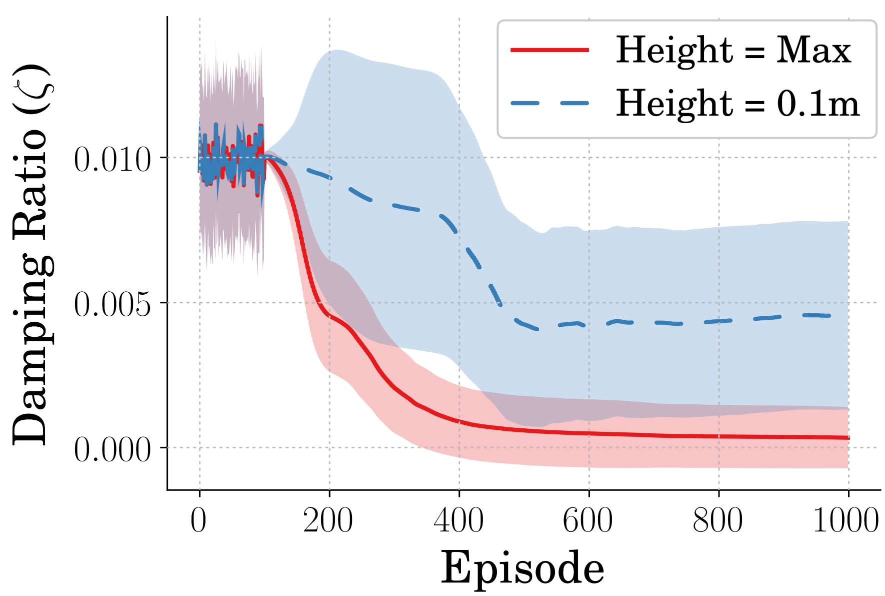


Figure 22. Damping Ratio Selected During Training

design parameters the agents selected during training are shown in Figs. 21 and 22. 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 designs which maximized height found designs with very little variance in terms of spring constant and significantly less variances in terms of damping ratio.

4.6.2 Wide Design Space. Figure 23 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 23 can be compared with Figure 18 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 0.1 m, the designs learned accomplish this, only with slightly more variance than what is seen in the maximum height agents.

The average and standard deviation of the spring constant and damping ratio

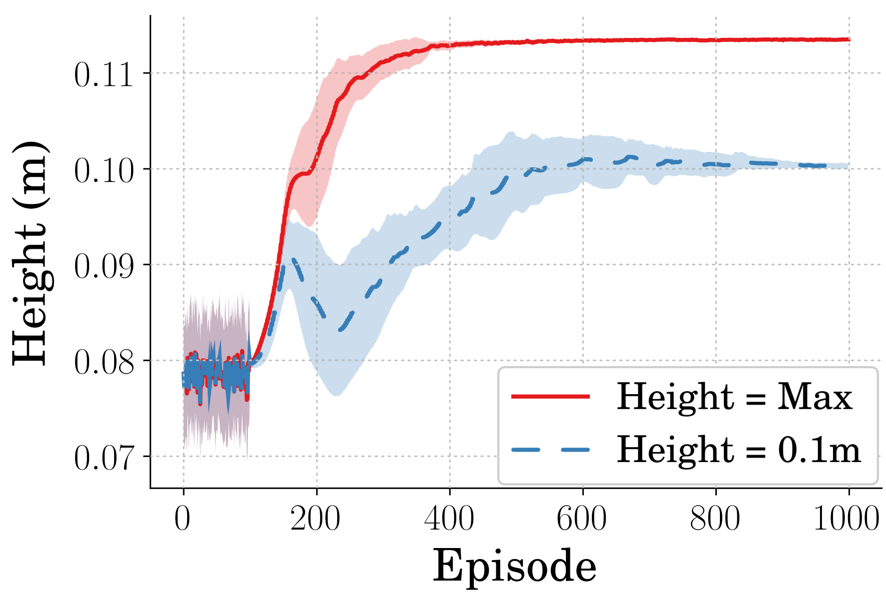


Figure 23. Height Reached During Training

design parameters the agents selected during training are shown in Figures 24 and 25. 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 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.

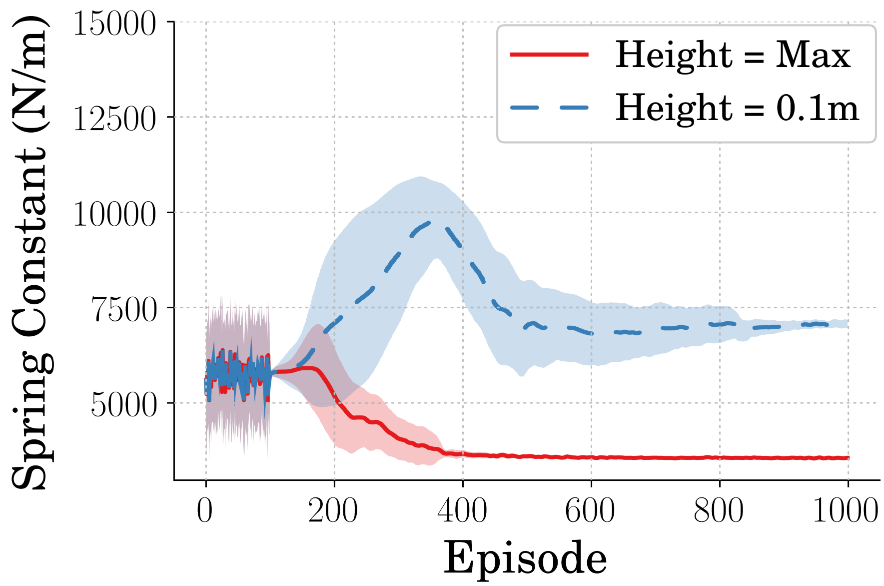


Figure 24. Spring Constant Selected During Training

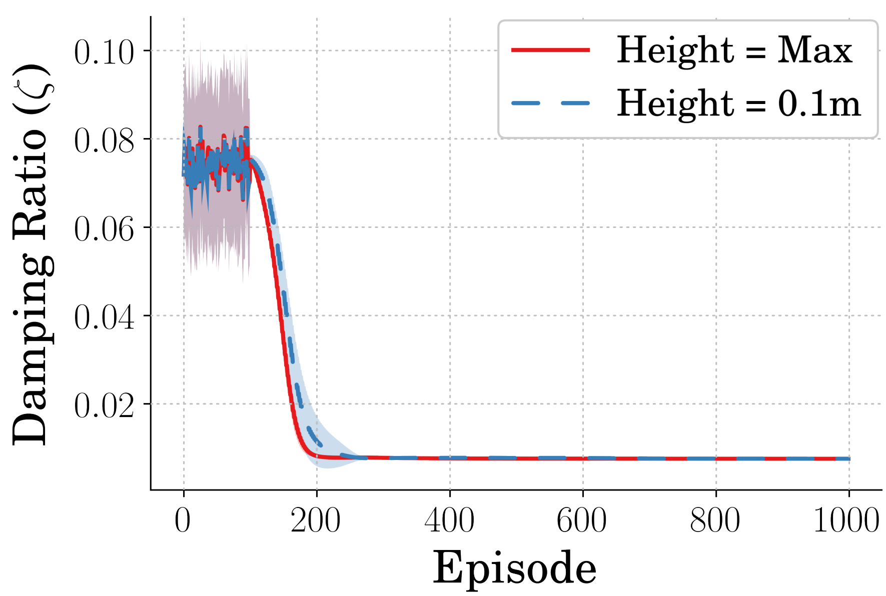


Figure 25. Damping Ratio Selected During Training

The agents which were learning designs that maximized height found them with very little variance in terms of spring constant and damping ratio.

4.6.3 Average Design Performance. The final mean and standard deviation of the design parameters for the two different cases are presented in Table 4. Figure 26 shows the jumping performance of the mean designs learned for both cases tested. The agents tasked with finding designs to jump to the specified 0.1 m, did so with minimal error. The difference seen in maximum height reached between the two

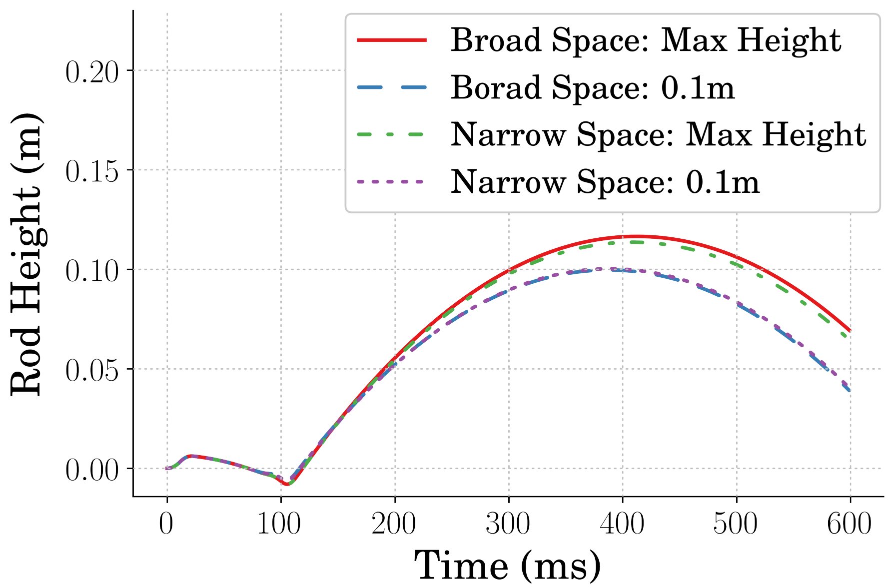


Figure 26. Height vs Time of Average Optimal Designs

Table 4. Learned Design Parameters

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

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 17 and 18 to show that the agents learned designs nearing those achieving maximum performance.

4.7 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. It should be concluded ultimately that utilizing an RL algorithm, such as TD3, for the mechanical design aspect of a concurrent design method, is a viable solution.

V Concurrent Design of the Monopode System

Finding a control architecture for a mechanically defined system is often the the workflow for generating a controlled robotic system. However, the mechanical system is not always a simple one and generating a controller for it may require a more complex workflow. It is of interest as well to allow the mechanical parameters of the system, and therefore the system description, to be fluid, allowing for a more optimal mesh between controller and system. Designing the system and control input in unison has been researched and is ofter referred to as concurrent design. ...

Some examples of non-RL methods of concurrent design

[].

However, the utilization of more complex deep learning methods has also been shown to be an effective strategy for finding optimal concurrent designs []. It has been used to find a concurrent designs for legged robotic systems leading to improved performance in regards to movement velocity []. Some research has done where the controllers were deployed in a simulation to real process validating that this technique is an effective strategy for developing a system/controller architecture [].

5.1 Environment Definition

Explain how the environment was implemented.

5.2 Learning Concurrent Designs

Describe the rewards defined for the learning of concurrent designs.

5.3 Jumping Cases

The cases which the agents were learning designs and controllers for.

5.4 Mechanical Designs

- Figure: the designs that the agent learned

- Figure: The learning curve during training of the agents

5.5 Controller Performance

- Figure: the performance of the controllers with nominal designs

- Figure: the performance of the controllers with learned designs

5.6 Conclusion

The conclusion.

VI Concurrent Design of a Two-Link Flexible-Legged Jumping System

Discussion on concurrent design for robotic systems. How does this relate to our system and work? Overview of the system.

- Figure: system
- Table: System parameters

6.1 Jumping Cases

The cases which the agents were learning designs and controllers for.

6.2 Mechanical Designs: Simulation

This is the data basically.

6.3 Controller Performance: Simulation

- Figure: the performance of the controllers with nominal designs
- Figure: the performance of the controllers with learned designs

6.4 Mechanical Designs: Sim-to-Real

This is the data basically.

6.5 Controller Performance: Sim-to-Real

- Figure: the performance of the controllers with nominal designs
- Figure: the performance of the controllers with learned designs

6.6 Conclusion

The conclusion.

VII Appendix: StableBaselines3

Maybe a needed description of the code.

VIII Appendix: Equations of Motion

EOMs.

Bibliography

[1] Folkertsma, G. A., Kim, S., and Stramigioli, S., "Parallel stiffness in a bounding quadruped with flexible spine," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2210–2215, 2012.

[2] Park, H. W., Wensing, P. M., and Kim, S., "High-speed bounding with the MIT Cheetah 2: Control design and experiments," *International Journal of Robotics Research*, vol. 36, no. 2, pp. 167–192, 2017.

[3] Blackman, D. J., Nicholson, J. V., Pusey, J. L., Austin, M. P., Young, C., Brown, J. M., and Clark, J. E., "Leg design for running and jumping dynamics," *2017 IEEE International Conference on Robotics and Biomimetics, ROBIO 2017*, vol. 2018-Janua, pp. 2617–2623, 2018.

[4] Seok, S., Wang, A., Chuah, M. Y., Hyun, D. J., Lee, J., Otten, D. M., Lang, J. H., and Kim, S., "Design principles for energy-efficient legged locomotion and implementation on the MIT Cheetah robot," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 3, pp. 1117–1129, 2015.

[5] Sugiyama, Y. and Hirai, S., "Crawling and jumping of deformable soft robot," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 4, no. c, pp. 3276–3281, 2004.

[6] Galloway, K. C., Clark, J. E., Yim, M., and Koditschek, D. E., "Experimental investigations into the role of passive variable compliant legs for dynamic robotic locomotion," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1243–1249, 2011.

[7] Hurst, J., "The Role and Implementation of Compliance in Legged Locomotion," *The International Journal of Robotics Research*, vol. 25, no. 4, p. 110, 2008.

[8] Pratt, G. A. and Williamson, M. M., "Series elastic actuators," *IEEE International Conference on Intelligent Robots and Systems*, vol. 1, pp. 399–406, 1995.

[9] Ahmadi, M. and Buehler, M., "Stable control of a simulated one-legged running robot with hip and leg compliance," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 1, pp. 96–104, 1997.

[10] Kani, M. H. H. and Ahmadabadi, M. N., "Comparing effects of rigid, flexible, and actuated series-elastic spines on bounding gait of quadruped robots," pp. 282–287, 2013.

[11] Sutton, R. S. and Matheus, C. J., "Learning Polynomial Functions by Feature Construction," *Machine Learning Proceedings 1991*, pp. 208–212, 1991.

[12] Watkins, C., "Learning From Delayed Rewards," 1989.

[13] Mnih, V. and Silver, D., "Playing Atari with Deep Reinforcement Learning," pp. 1–9.

[14] Zhao, W., Queralta, J. P., and Westerlund, T., "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey," *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020*, pp. 737–744, 2020.

[15] Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M., "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards," pp. 1–10, 2017.

[16] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W., "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research," pp. 1–16, 2018.

[17] Fujimoto, S., Van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," *35th International Conference on Machine Learning, ICML 2018*, vol. 4, pp. 2587–2601, 2018.

[18] Dormann, A. R., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Noah, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 22, pp. 1–8, 2021.

[19] Ha, S., Xu, P., Tan, Z., Levine, S., and Tan, J., "Learning to walk in the real world with minimal human effort," *arXiv*, no. CoRL, pp. 1–11, 2020.

[20] Fankhauser, P., Hutter, M., Gehring, C., Bloesch, M., Hoepflinger, M. A., and Siegwart, R., "Reinforcement learning of single legged locomotion," *IEEE International Conference on Intelligent Robots and Systems*, pp. 188–193, 2013.

[21] Blickhan, R. and Full, R. J., "Similarity in multilegged locomotion: Bouncing like a monopode," *Journal of Comparative Physiology A*, vol. 173, no. 5, pp. 509–517, 1993.

[22] Harper, M. Y., Nicholson, J. V., Collins, E. G., Pusey, J., and Clark, J. E., "Energy efficient navigation for running legged robots," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 6770–6776, 2019.

[23] Pace, J., Harper, M., Ordonez, C., Gupta, N., Sharma, A., and Collins, E. G., "Experimental verification of distance and energy optimal motion planning on a skid-steered platform," *Unmanned Systems Technology XIX*, vol. 10195, p. 1019506, 2017.

[24] Vaughan, J., "Jumping Commands For Flexible-Legged Robots," 2013.

[25] Sorensen, K. L. and Singhose, W. E., "Command-induced vibration analysis using input shaping principles," *Autom.*, vol. 44, pp. 2392–2397, 2008.

[26] Singer, N. C. and Seering, W. P., "Preshaping Command Inputs to Reduce System Vibration," *Journal of Dynamic Systems, Measurement, and Control*, vol. 112, no. 1, pp. 76–82, 1990.

[27] Singhose, W., Seering, W., and Singer, N., "Residual Vibration Reduction Using Vector Diagrams to Generate Shaped Inputs," *Journal of Mechanical Design*, vol. 116, no. 2, pp. 654–659, 1994.

[28] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym," pp. 1–4, 2016.

Albright, Andrew. Bachelor of Science, North Carolina State University, Spring 2020;
Masters of Science, University of Louisiana at Lafayette, Spring 2022
Major: Mechanical Engineering
Title of Thesis: Mechanical Design and Control of Flexible-Legged Jumping Robots
Thesis Director: Dr. Joshua E. Vaughan
Pages in Thesis: 125; Words in Abstract: 185

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus nec tellus eget elit aliquet accumsan sit amet in lacus. In hac habitasse platea dictumst. Ut sit amet elit odio. Aenean lobortis mollis metus, sed consequat neque tristique in. Curabitur nec hendrerit metus. Praesent non scelerisque urna, vitae iaculis diam. Aliquam nisl est, imperdiet eu nulla sed, bibendum pulvinar arcu. In ultricies purus purus, vulputate congue justo volutpat ut. Donec nunc magna, rutrum nec turpis et, viverra efficitur lorem. In hac habitasse platea dictumst. Vestibulum maximus lobortis nisl, eget molestie sem sollicitudin nec. Mauris ut enim eu ipsum auctor rhoncus ac vel eros.

Vivamus tincidunt, tortor eu rutrum dapibus, orci turpis porta metus, ac iaculis quam eros sollicitudin nisl. Nam id massa elementum, commodo mi at, lobortis nisl. Fusce vestibulum eu lorem non aliquam. Morbi eleifend tortor id metus elementum, ac tincidunt lorem commodo. Pellentesque vestibulum, erat in tempus vehicula, ex urna auctor leo, ut lobortis eros mauris nec erat. Aliquam erat volutpat. Sed sed pretium risus.

Biographical Sketch

Forrest Montgomery was born in Lafayette, Louisiana for all intents and purposes. He began his academic career at the University of Louisiana with an internal struggle between majoring in Mechanical Engineering or Industrial Design. This thesis is evident of the choice he made. After earning his Bachelor's degree at the University of Louisiana at Lafayette in the Spring of 2015, he joined the CRAWLAB and conducted research in dynamics, controls, and robotics under the tutelage of Dr. Joshua Vaughan. This research culminated with earning a Master's degree in Mechanical Engineering again at the University of Louisiana at Lafayette in the Summer of 2017.