Reinforcement Learning for Bipedal Locomotion on Cassie

Phi Luu and Benjamin Geyer

Abstract

Robots are designed to perform complicated tasks, some of which include standing and walking. In order to control a robot to perform such tasks, model-based controllers were introduced, and they have demonstrated great success in this area. However, due to their simplicity, these controllers are not fully aware of many details, such as torque limits and joint limits. Using model-based approaches to design higher degree-of-freedom systems can be very hard because there are too many behaviors to write down with rules. To tackle this problem, reinforcement learning offers a model-free approach for controlling bipedal locomotion which can more fully exploit the dynamics and capabilities of the robot. In our research, we aim to use reinforcement learning to train a bipedal robot, Cassie, to be more adaptive and agile against adversarial environments.

Simulating Cassie

We used a simulator named MuJoCo to simulate interactions between Cassie and the surrounding environment. The purpose of experimenting on a simulator is to have more control over various factors that may affect the outcome of an experiment, such as air resistance, friction, depreciation of motors and other mechanical parts, etc. Simulation offers a large number of advantages, some of which include

- Setting the gravity to zero (or any desired number)
- Shaping the floor into a sinusoid wave
- Generating various types of obstacles
- Setting friction and collision parameters

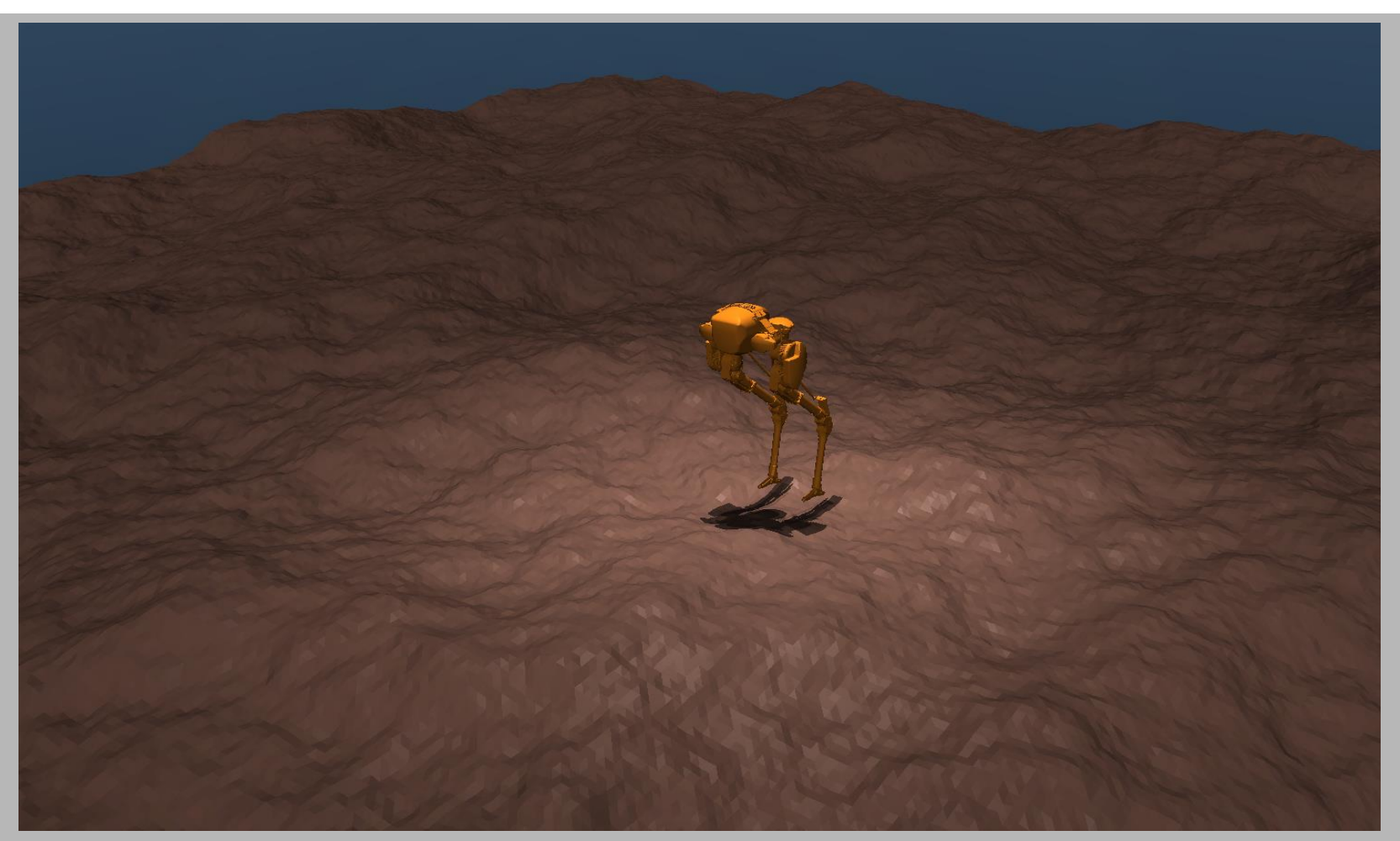


Fig. 1: Cassie with a terrain generated using Perlin Noise in MuJoCo

The list keeps going on and on. By taking the advantages of MuJoCo, we were able to

- Train Cassie continuously
- Retry quickly after failing without breaking the real prototype of Cassie
- Generate various terrains using Perlin Noise
- Explore different approaches and test which one helps Cassie learn the fastest
- Reduce the costs of real prototype training while maintaining realistic physical interactions.

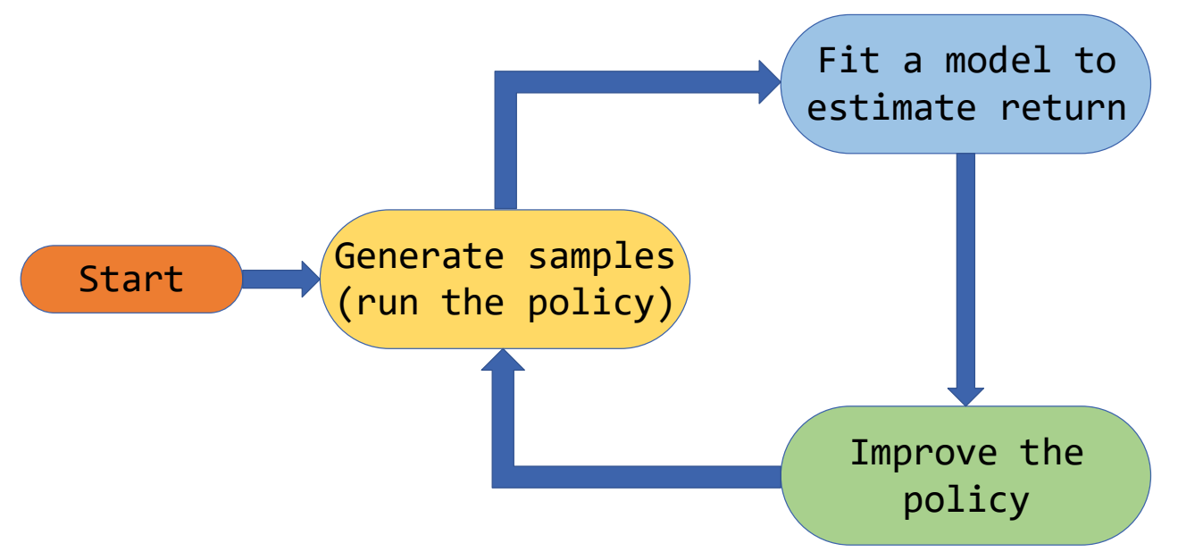
Phi Luu and Benjamin Geyer are with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA.
Emails: `{luuph,geyerb}@oregonstate.edu`

Reinforcement Learning and Vanilla Policy Gradient — A Brief Overview

Reinforcement learning (RL) is a subset of machine learning. In RL, an `agent` (in our case, Cassie) approaches a task by considering the current `state` of an `environment` it is in and then choosing an `action`. The `agent` then takes that `action` within the `environment`, and its resulting `state` is recorded and passed back to the `agent` to repeat the process. Each `(action, state)` pair is evaluated on how well it completes the given task by a `reward function`.

The goal of RL is to take the action at each state that maximizes the reward. The action taken at a given state is determined by a neural network π with policy parameters θ (weights between each neuron). Thus, the ideal solution is achieved by finding the optimal parameters for the network.

A popular method to achieve this goal is **Vanilla Policy Gradient (VPG)**. It works by sampling the cumulative reward given from various series of actions and resultant states. We then use calculus to estimate the maximum rewards. The policy is then shifted in the direction of the highest ascent until it converges on a global maximum (see flowchart).


Fit a model to estimate return
Start
Generate samples (run the policy)
Improve the policy

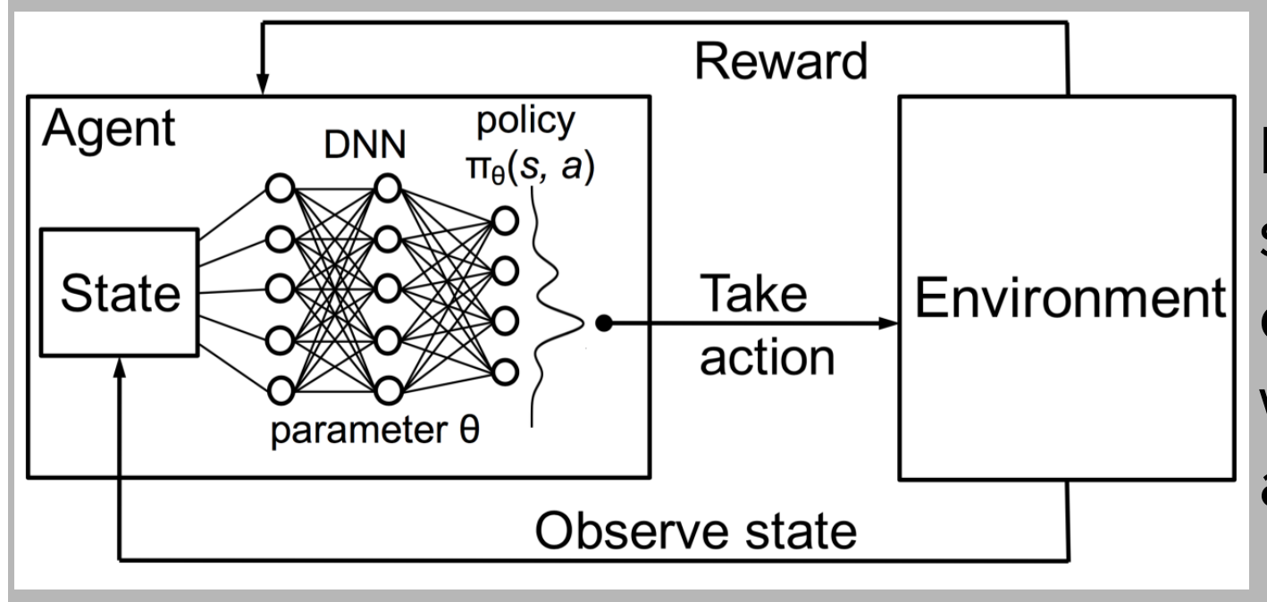


Fig. 2 [1]: This is the basic structure of RL, with an agent controlled by a neural network with an input of the current state and the output of an action

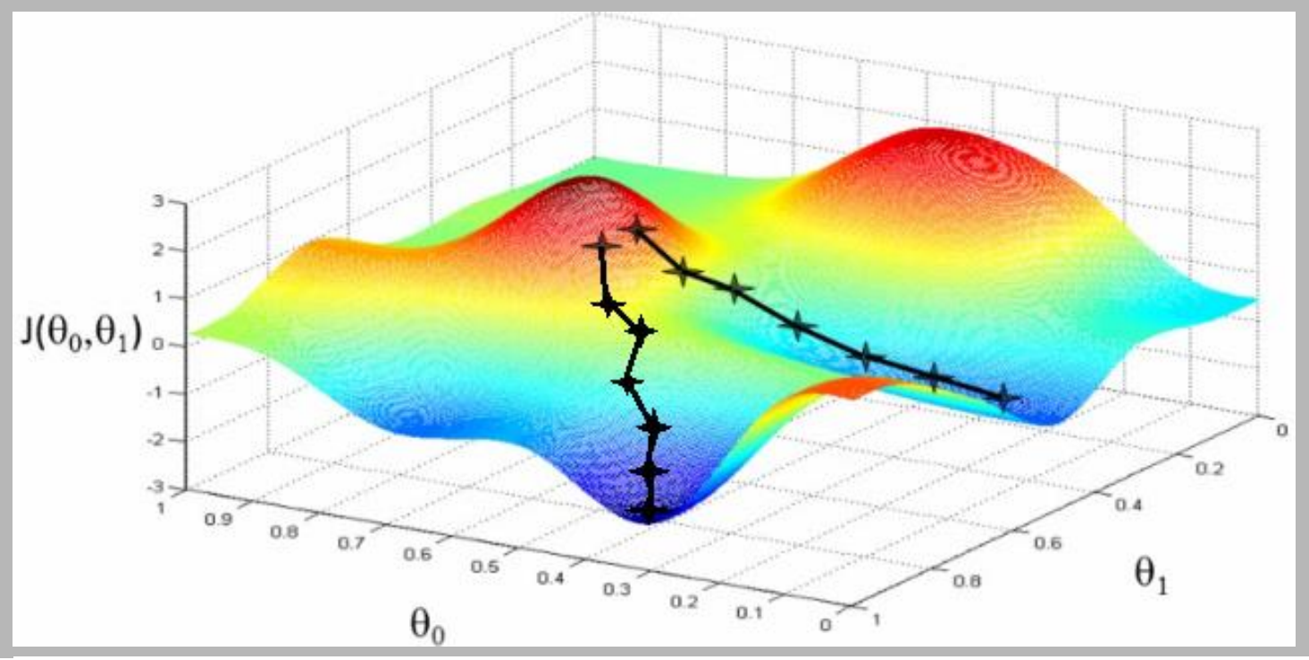


Fig. 3 [2]: A visual representation of gradient ascent converging on a maximum

Implementing Vanilla Policy Gradient in Cassie

The basic structure of our implementation includes the following:

- **State space**: Determined by the positions and velocities of three main components: body, left foot, and right foot [3].
- **Action space**: We used two different action spaces: Torque and Target Joint Angle. Choosing an action space determines Cassie's ability to learn, and so it is very important.
- **Reward function**: We initialized r = 0 and changed it according to the following
  - Standing:

$$r \mathrel{-}= coefficient \cdot (target\ height - body\ height)^2$$
$$r \mathrel{-}= coefficient \cdot (average\ position\ of\ two\ feet)^2$$
$$r \mathrel{+}= 1$$

- Maintain the desired height
- Place feet underneath the center of mass
- Reward for staying alive

  - Walking:

$$r \mathrel{+}= coefficient \cdot (position\ after - position\ before)$$
$$r \mathrel{+}= 0.5$$

- Reward for moving forward
- Reward for staying alive

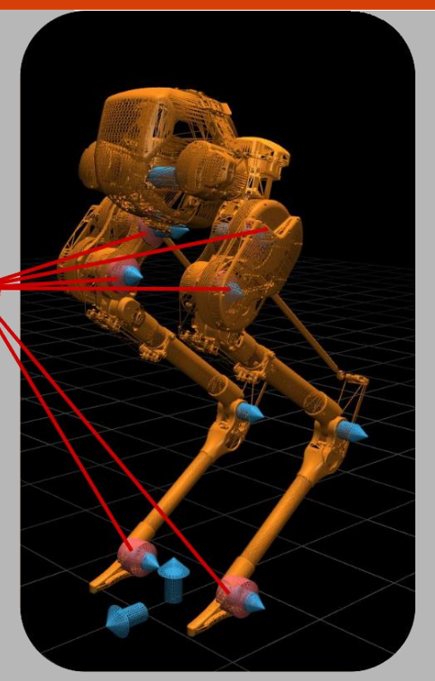


Fig. 4 [4]:
Torque action space
Torques for all actuators

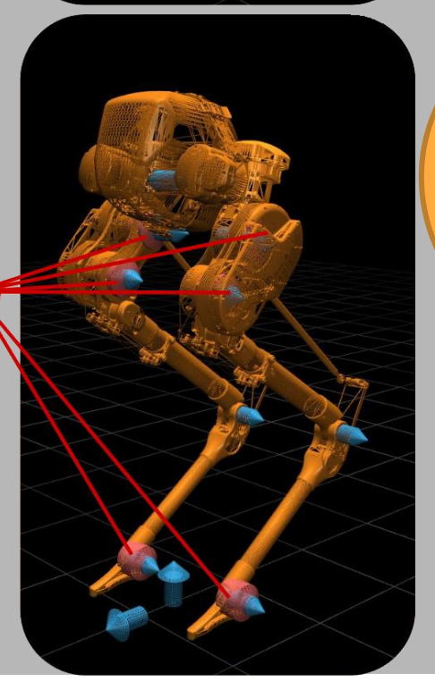


Fig. 5 [4]:
Target Joint Angle (PD) action space
Target angles for all joints

We use **REINFORCE** algorithm to optimize the policy parameters, per below:

```
Repeat until convergence:
    1) Sample trajectories
    2) Calculate the gradient
       $\nabla_\Theta J(\Theta) \approx \sum_i \left[\sum_t \nabla_\Theta \log \pi_\Theta(a_t^i | s_t^i)\right]\left[\sum_t r(a_t^i, s_t^i)\right]$
    3) Update the parameter
       $\Theta \leftarrow \Theta + \alpha \cdot \nabla_\Theta J(\Theta)$
```

The development repository of this research project can be found at https://github.com/CassieRL/cassierl

Our Results

Standing

With our initial reward function, Cassie was unable to learn how to stand, instead simply falling as quickly as possible to minimize the negative reward. By multiplying each negative reward by a constant that is less than 1, we were able to lessen their effect on the reward as a whole and allow Cassie to learn to hold itself upright. The average reward per iteration, as well as an image of Cassie attempting to stand, is shown below in figures 6 and 7, respectively.

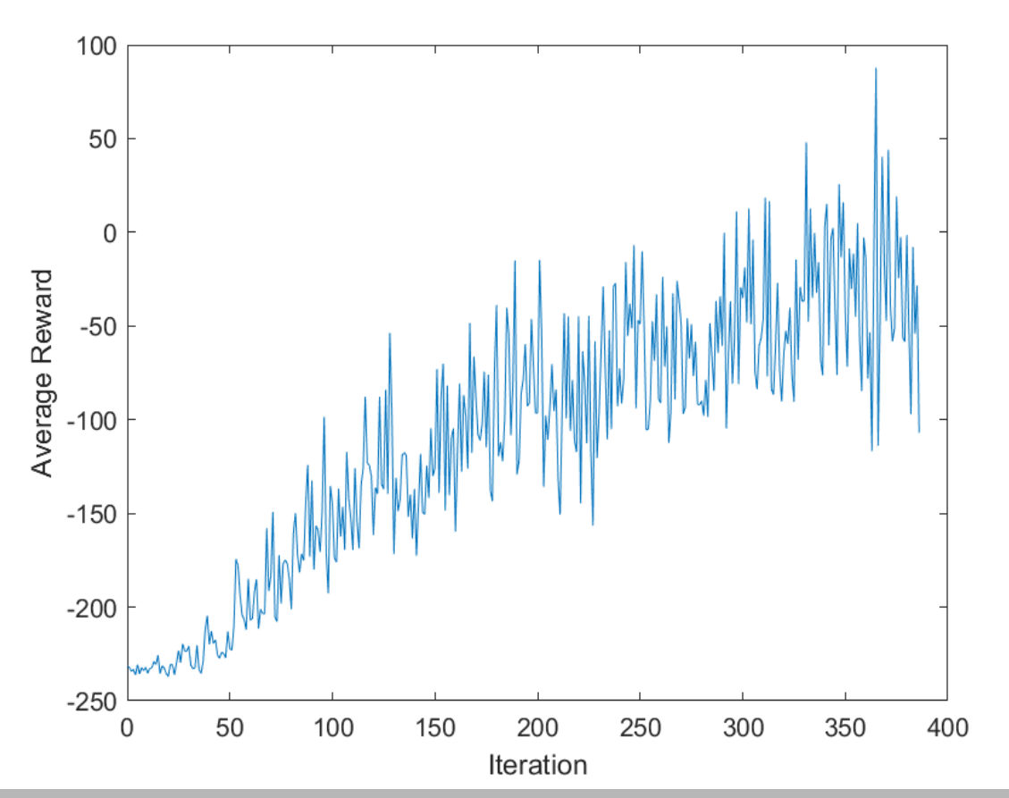


Fig. 6: The average reward for this standing test increased fairly steadily indicating that it is learning effectively. However, the high variance in the graph indicates that it is not taking factors regarding its stance, such as the distance between its feet, into account as much as we would like.

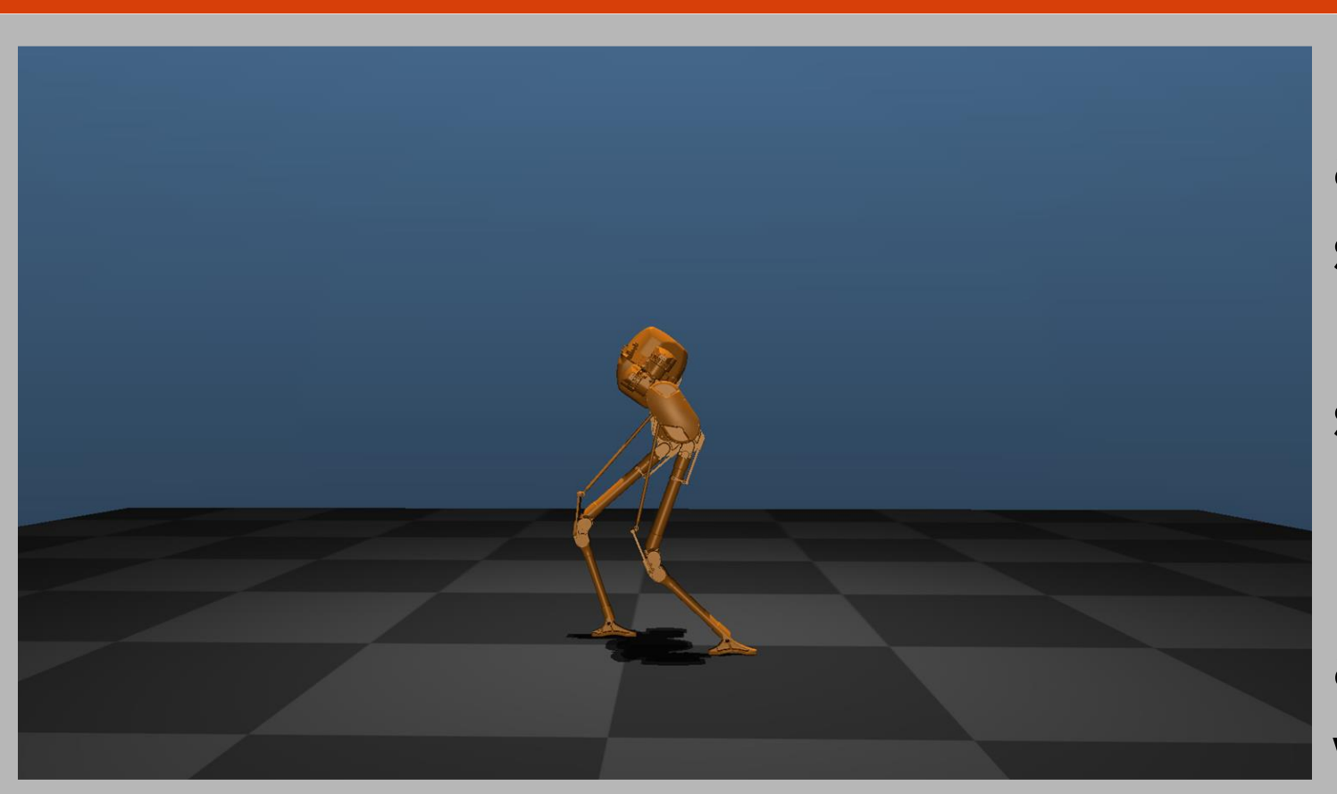


Fig. 7: Cassie attempting to stand. Though it remains upright, its stance is not ideal, leaving more room between its front and back foot than we would like.

Walking

We trained Cassie for walking using both torques and PD as action spaces. They both had similar results — learning to move successfully, but not in a typical walking motion. Cassie learned much faster on PD action space than on torque, indicating that PD might be better to use for quicker learning in the future. The reward graphs for torques and PD action spaces are shown in Figures 8 and 9, respectively, and the walking motion is displayed in Figure 10.

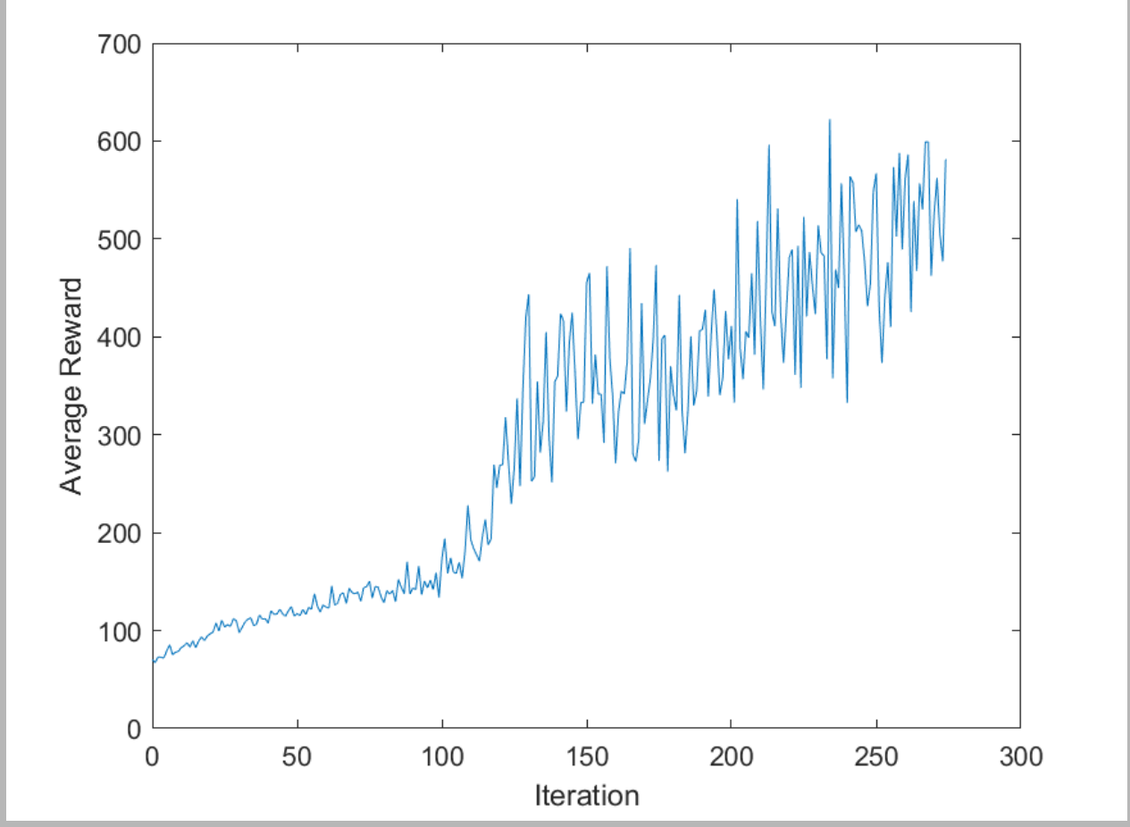


Fig. 8: The average reward for torques increased steadily, however at a certain point it develops high variance, similar to the standing test.

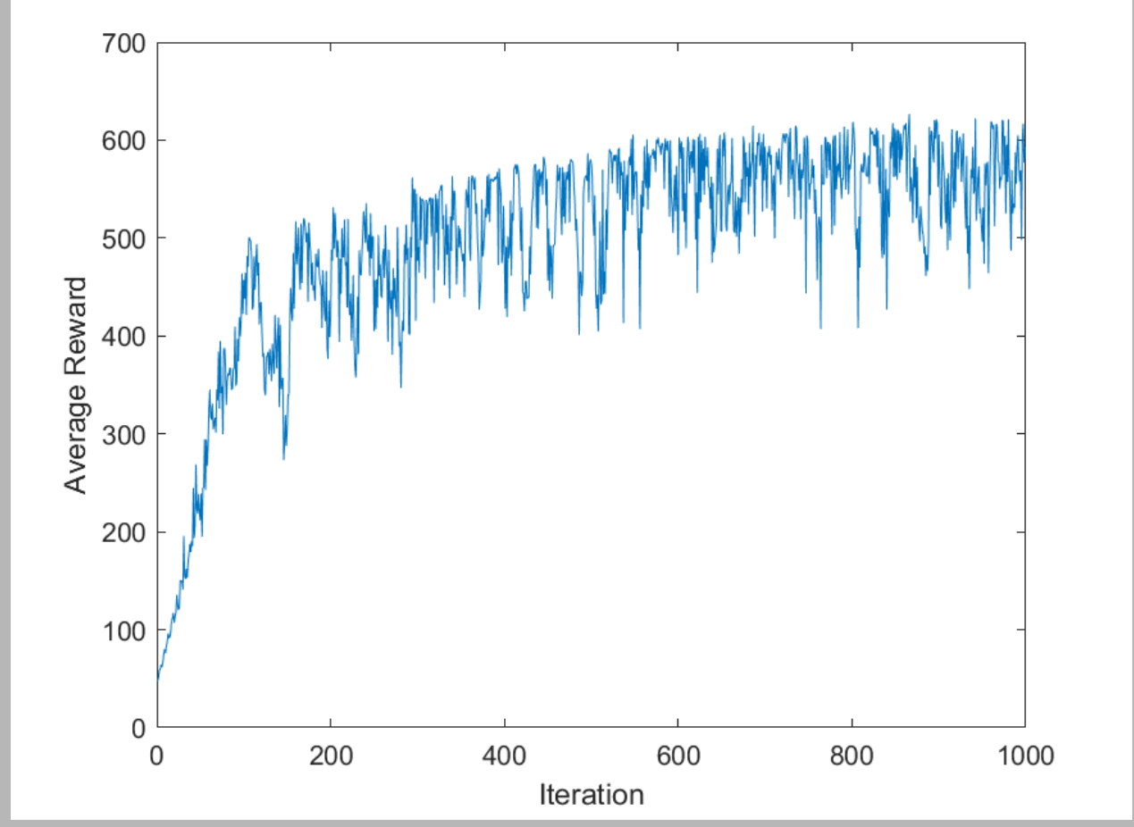


Fig. 9: The average reward for PD increased much more rapidly than torques, indicating faster learning, however it too had significant variance.

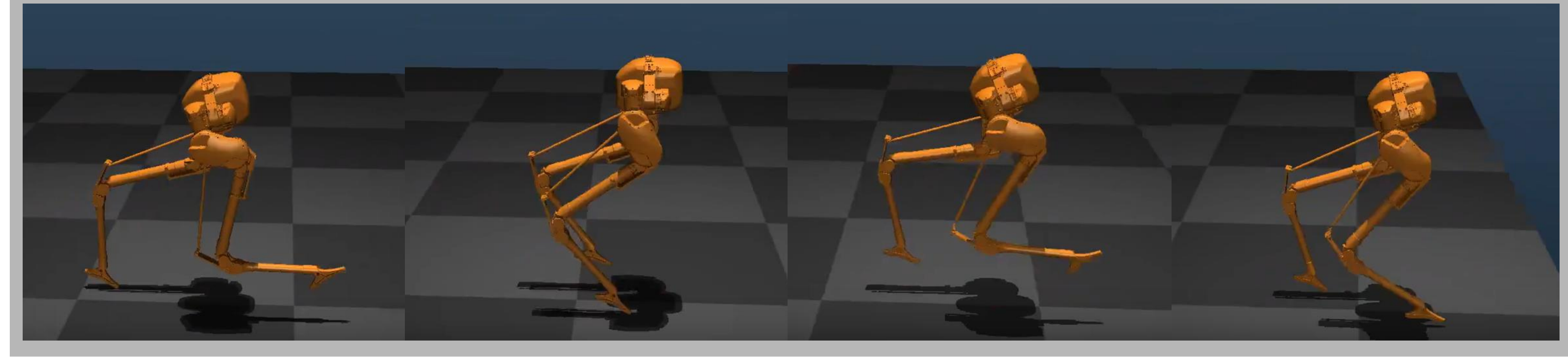


Fig. 10: Cassie galloping forward after training on PD. Although it moves forward successfully, it does not do so with a motion akin to walking.

Conclusion

Vanilla Policy Gradient is simple and easy to implement. However, it is usually challenging to choose a right learning rate, as a step size that is too large will cause the training to diverge and a tiny step size will take the training a vast amount of time to converge. Additionally, as seen in the results, Vanilla Policy Gradient create moderately high variance in the average return.

During this research, we found that our choice of **action space** and the way we shape the **reward function** had a huge impact on Cassie's ability to learn. Target Joint Angle demonstrated to be a better action space than Torque with respect to learning speed.

We found this research on reinforcement learning very interesting. With the current trend in machine learning and computational power, we think reinforcement learning could be a reliable training method in the future.

Direction for Future Research

- Improve walking performance by using reference trajectories [3]
- We were able to generate various terrains using Perlin Noise, but we haven't integrated them into our training experiments. The algorithm can become more robust if terrains are involved in the training.
- Include more joints in the MuJoCo model and allow Cassie to learn more complex movements
- Add external forces and obstacles to test and increase the robustness of the algorithm.

Acknowledgments

We would like to thank Taylor Apgar and Dr. Jonathan Hurst for all their guidance in this research. Thanks to Yathartha Tuladhar for constructing rllab interface for training experiments.

References

[1] "drl.png," Hongzi Mao, http://people.csail.mit.edu/hongzi/content/publication_img/drl.png.
[2] "gradient-descent.png." *Tuning the learning rate in Gradient Descent*, Vasilis Vryniotis, 27 Oct. 2013, http://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent.
[3] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. Panne. "Feedback Control for Cassie with Deep Reinforcement Learning." *Cornell University Library*. Accessed May 15th, 2018.
[4] Y. Tuladhar, and T. Apgar. "Reinforcement Learning for Legged Locomotion." [Online]. Available: https://github.com/CassieRL/cassierl/blob/master/Docs/Writeup.pdf.