Quadrupedal Locomotion via Periodic Reward Composition and Deep Reinforcement Learning

by

Johnathon S. Li
BS (Massachusetts Institute of Technology) 2019

A report submitted in partial satisfaction of the
Requirements for the degree of

Masters of Science, Plan II

in

Mechanical Engineering

at the

University of California, Berkeley

Committee in Charge:

Professor Koushil Sreenath, Chairman

Professor Mark W. Mueller

Spring 2021

Contents

Abstract

The development of walking controllers for legged robotics is a complex process that involves careful modeling of the system and robust controller design. If the model of the system is not perfect, non-negligible uncertainty could be introduced into the system, which can result in instability. To counteract this effect of uncertainty, stochastic methods, namely reinforcement learning (RL), have been used to train legged robots to achieve stable gaits. Many of these RL methods lie in the domain of imitation learning, but this may be constricting to the reinforcement learning policy's exploration process. In addition, a high quality reference trajectory must be first defined before the RL training process, and this process can be tedious. On the other hand, model-free RL methods give learning agents the freedom to explore their state spaces, but the resulting gaits may not be optimal nor natural for sim-to-real transfer. Recent work from [10] created an intuitive way to design reward functions to guide a model-free RL agent to learn a spectrum of common bipedal gaits. This work balanced the constraining, but well-specified, method of imitation learning and freeing, but under-specified, method of model-free RL. Since quadrupeds are close relatives of bipeds, there is a natural curiosity to see if this framework would work for quadrupeds. This Masters project aims to answer this question.

1 Introduction

Currently, using reinforcement learning (RL) to learn common quadrupedal gaits, such as bounding, trotting, or galloping, is not yet a solved problem. According to the current state of the art, a key challenge in RL is to utilize the reward function in order to manifest a particular walking gait. When designing a reward function for a locomotion gait, the reward function must be specific enough to produce desired gait characteristics while also not being overly constraining. [10]

It is common for the use of reference trajectories as a means to guide RL algorithms to learn policies that minimize deviation from such reference trajectories all while being robust to environmental uncertainty and disturbances. [12] [7] [13] However, though the policies can learn to mimic such reference trajectories very well, flexibility of the policy is sacrificed.

On the other hand, reference-free RL methods, such as those used in OpenAI Gym benchmarks [1], provide the chance for a policy to explore more of its state-space, but the resulting policy might not be well suited for real-life robots. In addition, tuning a reward function is a tedious process that involves trial-and-error for complex systems, such as real robots with high dimensionality.

Recent work from [10] achieved a balance between reference-trajectory based RL methods and model-free RL methods by introducing a parametric reward function that dynamically changes based on *swing* (foot lifted off of the ground) and *stance* (foot in contact with ground) phases of a bipedal robot. By utilizing the complementary physical traits of swing and stance phases, one can intuitively design a reward function that measures if a robot's leg is in its respective correct leg phase during a time step. This is done through incorporating information about the foot velocities (meaning no contact with the ground) and foot forces (meaning existing contact with the ground) into the reward function.

The hypothesis of this Masters project is to determine if the extrapolation from [10] to quadrupeds will be able to train a quadruped to walk with a common quadrupedal gait. By re-writing the algorithm from scratch as a Python RL library with support for the MIT Mini-Cheetah and Mujoco Half-Cheetah, this project evaluated if the original method in [10] works for quadrupeds. By refining the original algorithm, a planar approximation of a quadruped was successfully trained to perform various gaits such as galloping, pronking and standing in simulation.

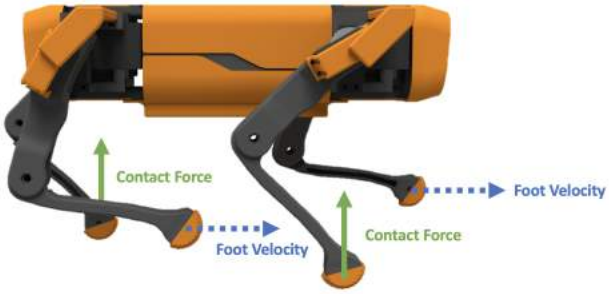

Figure 1: Illustration of swing phase (in blue) for front left and rear right legs and stance phases (in green) of front right and rear left legs.

2 Framework Preliminaries

2.1 Reinforcement Learning Framework

This project involves training in the discrete time space. The environment is characterized by a discrete-time Markov Decision Process (MDP) with a continuous state space S, continuous action space A, a transition function $T(s, a, s')$, and a reward function $R(s, t)$. The control policy is a stochastic mapping $\pi(a|s)$ from observations to actions, and actions will be sampled from our policy π. The general goal of RL, to maximize

$$J(\pi) = \mathbb{E}\left[\Sigma_{t=0}^{T} \gamma^t R(s_t, t)\right] \tag{1}$$

will be used by this project. The discount factor $\gamma \in [0, 1]$, and s_t is a random variable drawn from a Gaussian distribution that represents the state at time t while following actions from the policy π.

2.2 Describing Legged Locomotion

All common quadrupedal gaits can be defined by periodic *swing phases* and *stance phases*. For swing phases, a foot should be in the air with some finite velocity and have zero reaction force with the ground. The opposite is true for stance phases. Reward function design will be guided with these principles. Figure 1 illustrates swing and stance phases for a quadrupedal robot.

3 Periodic Reward Composition Design

3.1 Normalization of Time

Since legged locomotion is periodic in behavior, it is natural to incorporate a design variable that is also periodic in nature. Time *t* in this project is indexed via a cycle time variable, ϕ, which cycles through normalized time. ϕ is defined as

$$\phi = \frac{k}{L} \bmod 1.0 \tag{2}$$

where k is a counter variable that increments by 1 every single time the low-level controller to the quadruped is called. L is a user-defined parameter that determines how many iterations of calling the low-level controller equates to a full period. $\phi \in [0.0, 1.0]$ which normalizes time. For example, if the gym environment runs at 500 Hz and L = 1000, one full period will equate to 2 seconds.

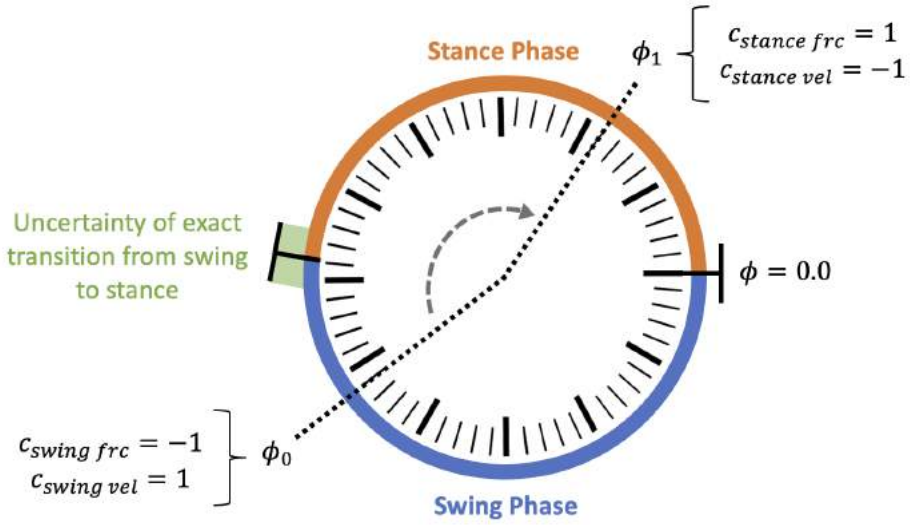

Figure 2: Relationship between ratio r and normalized time ϕ for a leg that starts in a swing phase. Here, $r \approx 0.5$. In green shows the stochasticity described in Section 3.2. The phase coefficients c_i shown are described in Section 3.4. Here, during a swing phase, the force phase coefficient $c_{\text{swing, frc}}$ = -1 to penalize a contact force while it is the opposite for $c_{\text{swing, vel}}$ = 1. As the environment's high level clock switches from a swing to a stance phase, the coefficients switch positvity to reward foot force instead of foot velocity.

3.2 Characterization of Phases in Normalized Time

Since ϕ is normalized between 0.0 and 1.0, it is very easy to determine the ratio of the interval L that should be characterized by a swing phase. The phase ratio r defines the fraction of L discrete normalized time steps that will be in swing phase. $1 - r$ (plus minus some uncertainty) determines the fraction of L discrete normalized time steps that will be in a stance phase. In my code implementation, the stance phase is constrained to start immediately after the swing phase. In addition, since RL is a stochastic framework, r is not deterministic. To facilitate the stochasticity of the MDP, the ratio r is drawn from a Von-Mises distribution:

$$r \sim \Phi(a, \kappa) \tag{3}$$

where κ is a variance parameter that determines how close are samples to the mean a, which is picked by the user. For legged locomotion, a good range for the value of $a \in [0.45, 0.7]$. For $\kappa \to \infty$, the standard deviation $\sigma \to 0$ and the distribution quickly becomes deterministic. For this project, κ was set to 5000. An illustration is provided in Figure 2.

3.3 Phase Parameterization of Reward Function

The reward function is designed as a function of ϕ and the state s. The reward function is defined as:

$$R(s, \phi) = \sum_i R_i(s, \phi), \qquad R_i(s, \phi) = \alpha_i \cdot q_i(s, \phi), \quad \text{with } \sum_i \alpha_i = 1.0 \tag{4}$$

R_i is an individual reward component to specify a particular quadrupedal gait characteristic during a particular phase; for example, the height of the base or the orientation of the base. α_i is a tuning coefficient for the reward component. $q_i(s)$ is a normalized reward in the form of

$$q_i(s, \phi) = e^{-|E|} \quad , \quad \text{where } E \text{ is an error calculation.} \tag{5}$$

So by the definition of the sum of α_i and the decaying exponentiation of $q_i(s)$, the total reward $R(s, \phi)$ will be bounded between 0.0 and 1.0.

3.4 Using Phase Coefficients for Swing and Stance Phases

NOTE: The following applies to a unipedal example to maintain compactness and clarity of explanation.

In (4), normalized reward components $q_i(s, \phi)$ are dependent on ϕ for reward components that correspond to the calculation of reward for foot forces and foot velocities. For these calculations, a new type of variable called *phase coefficients* $c(\phi)$ are introduced.

The purpose of phase coefficients is to give information to the reward component if a contact force or a foot velocity should contribute a reward or penalty for the error calculation in $q_i(s, \phi)$. For example, if a leg is in a swing phase, a foot contact should be penalized but on the other hand, if a leg is in stance phase, a foot contact should be rewarded. From this information, a phase coefficient for a contact force, c_{frc}, and a phase coefficient for a foot velocity, c_{vel} can be defined. In a swing phase, c_{frc} should be negative while c_{vel} should be positive, and vice-versa for a stance phase. In order to calculate c_{frc} and c_{vel} to satisfy these traits and to prevent discontinuities in the reward function, they are calculated as

$$c_{\text{frc}} = \tanh(\beta \cdot (\phi - r)) \ , \tag{6}$$

$$c_{\text{vel}} = -c_{\text{frc}} \ , \tag{7}$$

where β is a steepness coefficient to determine how steep the slope around $\phi = r$ should be. However, (6) only takes into consideration a phase coefficient calculation for a leg that starts in a swing phase but not a leg that starts in the stance phase since tanh starts at -1 when $\phi = 0.0$ and ends at 1 as $\phi \to 1.0$. In order to alleviate this issue, the equation was updated with a boolean variable f that denotes if a leg starts in either a swing or stance phase upon initialization. If f = `True`, then the leg starts in stance. The updated version of (6) is

$$\text{if } f = \texttt{True}: \quad \begin{cases} c_{\text{frc}} &= -1.0 \cdot \tanh\left(\beta \cdot (\phi - r)\right) \\ c_{\text{vel}} &= -c_{\text{frc}} \end{cases}$$

$$\text{if } f = \texttt{False}: \quad \begin{cases} c_{\text{frc}} &= \tanh\left(\beta \cdot (\phi - r)\right) \\ c_{\text{vel}} &= -c_{\text{frc}} \end{cases}$$

In (5), one can see that an error calculation $E = 0.0$ will maximize $q_i(s, \phi)$, so it is important to design a reward that can achieve $E = 0.0$ when the foot is doing the correct action. This is why phase coefficients are designed as such. The reward component design which satisfies the condition that $E = 0.0$ when the leg is performing the correct action is defined as

$$q(s, \phi, f)_{\text{foot frc / vel}} = e^{-|E|} \quad , \quad E = 1.0 - c(\phi, f) \cdot m(s) \tag{8}$$

where $c(\phi, f)$ can either be positive to decrease the error or be negative to increase the error. $m(s) \in \{-1, 1\}$ is a variable that denotes the presence or absence of a foot force or a foot velocity, which gives important information to the reward component. Since $m(s) \in \{-1, 1\}$ and $c(\phi, f) \in [-1, 1]$, using domain calculation for composition of functions gives that $[c(\phi, f) \cdot m(s)] \in [-1, 1]$ as well. Hence, $E \in [0.0, 2.0]$. If the foot is doing the correct action during its respective phase, E should be 0.0, which maximizes $q(s, \phi, f)$.

Using the updated versions of (6) and (8), the reward component for a uniped's foot force and foot velocity can be defined as

$$R_{\text{uni}}(s, \phi, f) = \alpha_{\text{frc}} \cdot q_{\text{frc}}(s, \phi, f) + \alpha_{\text{vel}} \cdot q_{\text{vel}}(s, \phi, f) \ . \tag{9}$$

Gait	Boolean Array **F**
Trot	`[True, False, False, True]`
Gallop	`[False, False, True, True]`
Pace	`[True, False, True, False]`
Pronk	`[False, False, False, False]`
Stand	`[True, True, True, True]`

Table 1: Boolean Array **F** for achieve various gaits. The indices in the array correspond to the front right foot, front left foot, rear right foot, and rear left foot.

In addition, since the calculation of c_{frc} and c_{vel} depend on r, a random variable, the reward function then becomes

$$\mathbb{E}[R_{\text{uni}}(s,\phi,f)] = \alpha_{\text{frc}} \cdot \mathbb{E}[q_{\text{frc}}(s,\phi,f)] + \alpha_{\text{vel}} \cdot \mathbb{E}[q_{\text{vel}}(s,\phi,f)] \tag{10}$$

3.5 Extrapolation to a Quadruped

To extend the reward of a uniped to a quadruped is intuitive and simple. (8) simply becomes

$$q(s,\phi,f)_{\text{foot frc / vel}} = e^{-|E|} \quad , \quad E = \texttt{float}(j) - \sum_j c_j(\phi, \mathbf{F}_j) \cdot m_j(s) \quad , \quad j = \text{\# of legs} \tag{11}$$

To extend the phase coefficient framework for a quadrupedal reward function, one just needs to set $j = 4$. In addition, the uniped boolean variable f will become **F**, an array of boolean values (of length j) rather than just a scalar.

Through domain calculation of composite functions, the new domain of $\left[\sum_j c_j(\phi, f) \cdot m_j(s)\right] \in [-j, j]$. Hence, $E \in [0.0, 2j]$. As long as all legs are doing the correct action during a particular gait phase, E should be zero. If all legs are not, then $q(s,\phi,f)$ will be minimized.

The new reward component for foot forces and foot velocities then becomes

$$\mathbb{E}[R_{\text{quad}}(s,\phi,f)] = \sum_j \left(\alpha_{\text{frc},j} \cdot \mathbb{E}[q_{\text{frc},j}(s,\phi,\mathbf{F}_j)] + \alpha_{\text{vel},j} \cdot \mathbb{E}[q_{\text{vel},j}(s,\phi,\mathbf{F}_j)]\right) \tag{12}$$

By altering the values in **F**, different gaits can be achieved. The various gaits that can be achieved for a quadruped can be seen in Table 1. A visualization of one of these gaits in terms of how (6) looks is illustrated in Figure 3.

In addition to R_{quad}, the reward function is supplemented by other penalties to ensure more natural walking. The new reward quantities are defined as:

$$R_{\text{cmd}}(s) = \alpha_1 \cdot q_{\dot{x}}(s) + \alpha_2 \cdot q_{\text{height}}(s) + \alpha_3 \cdot q_{\text{orientation}}(s) \tag{13}$$

$$R_{\text{smooth}}(s) = \alpha_4 \cdot q_{\text{torque}}(s) + \alpha_5 \cdot q_{\text{base accel}}(s) + \alpha_6 \cdot q_{\text{action diff}}(s, s(t-1)) \tag{14}$$

Here, α_i are tuning coefficients. $q_{\dot{x}}(s)$ is a reward for the normed error between a desired base x-velocity and the quadruped's actual base x-velocity. $q_{\text{height}}(s)$ is the the reward component for how close the quadruped base is to a desired center of mass height. $q_{\text{orientation}}(s)$ is a reward for the normed quaternion similarity between the base orientation versus a desired orientation. This facilitates a correct heading during a gait. $q_{\text{torque}}(s)$ is a penalty on large torque outputs and $q_{\text{base accel}}(s)$ penalizes linear and angular acceleration of the quadruped base to prevent jerkiness in motion. $q_{\text{action diff}}(s, s(t-1))$ rewards small deviations between the previous action drawn from $\pi(a|s(t-1))$ and the current action $\pi(a|s(t))$.

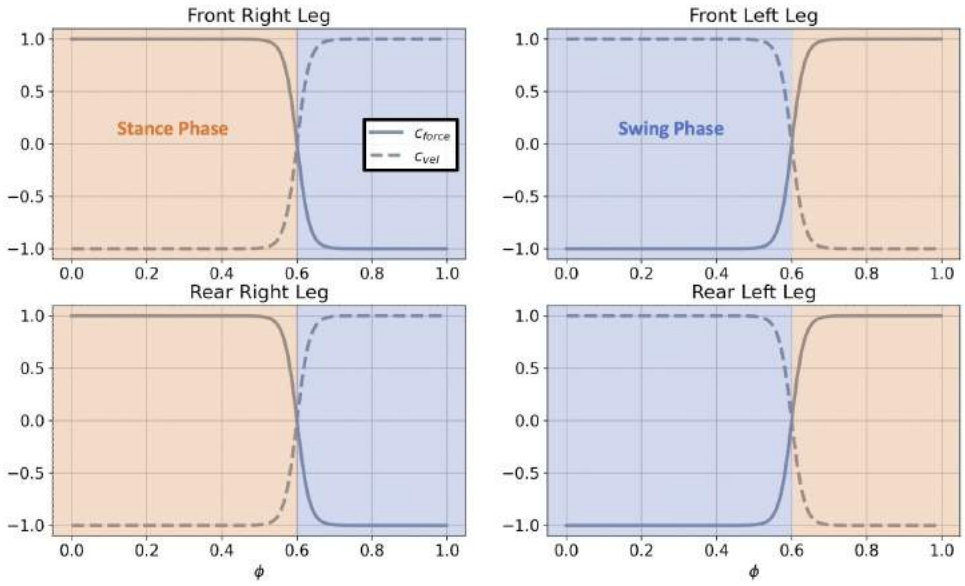

Figure 3: Phase coefficients over normalized time ϕ to result in a pace gait. Here, the steepness coefficient $\beta = 50.0$ and the swing ratio $r = 0.6$. The solid line indicates the phase coefficient for foot contact force while the dashed line indicates the phase coefficient for foot velocities. Here the front right leg and rear right leg are initialized to start in a stance phase, hence $c_{\text{force}} > 0.0$ while on the other hand, the front left and rear left legs start in a swing phase, so $c_{\text{vel}} > 0.0$. After the ratio r is reached, the coefficients switch, and the legs alternate the gait phase they were in.

With the new reward components, the final reward function to be used during training is:

$$\mathbb{E}[R(s, \phi)] = \mathbb{E}[R_{\text{quad}}(s, \phi, f)] + R_{\text{cmd}}(s) + R_{\text{smooth}}(s) \tag{15}$$

4 Reinforcement Learning Algorithm

4.1 Building a Quadruped Simulator

With the eventual goal of sim-to-real, it was important to build a RL gym environment for a robot which was available in the Berkeley Hybrid Robotics Lab. The only quadruped available at the time of building the simulator was the MIT Mini-Cheetah [5], so the low-level simulation and gym environment were built around the Mini-Cheetah. I used PyBullet [2], an open-source physics simulator, to create a simulated version of the MIT Mini-cheetah. The low-level simulator runs at 500 Hz, and the controller written is a joint position PD controller with proportional gains, $K_p = 1.0$ and $K_d = 0.02$. In figure 4, one can see the MIT Mini-Cheetah with a standing controller.

This low-level Python simulator is then wrapped by an OpenAI gym API to communicate with `stable-baselines3` [3], as discussed in Section 4.2.

4.2 Choosing a RL Algorithm

In [10], the authors used Proximal Policy Optimization [9], an on-policy RL algorithm developed at OpenAI. However, they used a recurrent neural network (RNN) [8] for their actor and critic policies. Their reasoning was that the MDP is technically a Partially Observable MDP (POMDP), so the use of state history would benefit the actor and critic to get closer to the true state. While RNNs are good for time dependent inputs, they also take much longer to train and are more prone to vanishing gradient issues. To prevent such, a Long Short-Term Memory (LSTM) policy [4] can be used, but even still, LSTMs still take much longer to train than vanilla multi-layer perceptrons (MLP).

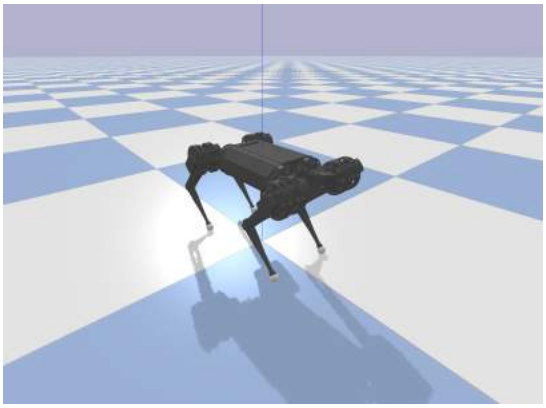

Figure 4: MIT Mini-Cheetah in PyBullet with a standing controller.

Due to computational limitations, discussed in Section 5.1, I opted to use a MLP for the PPO actor and critic networks. To alleviate partial observability, the input to the policy actor is appended with state observations from prior time steps. Hence, the policy becomes $\pi\left(a|s, s(t-1), s(t-2)...s(t-n)\right)$ for $n =$ the amount of state history.

4.3 Network Architecture

Both the actor and critic are both MLPs with two hidden layers, each with 128 nodes and **tanh** for the activation function. The output of the neural network corresponds to 12 desired motor joint positions bounded between [-1, 1]. The actions are then un-normalized by the low level Mini-cheetah simulator and the angles for a standing configuration are then added to the policy output.

4.4 State Space

To prevent the discrete-time MDP from becoming stationary [6], some sort of information about the normalized time must be incorporated into the state space. In order to achieve this, the policy is conditioned on an encoding of information related to *cycle offset parameters*.

Cycle offset parameters θ_{FL}, θ_{FR}, θ_{RL}, and θ_{RR}, which correspond to the front left leg, front right leg, rear left leg, and the rear right leg respectively, are used to provide information to the policy where exactly a foot is in the ϕ normalized temporal space. The offset parameters are paired up with the boolean array $\mathbf{F}$, so if $F_j = \texttt{True}$, the cycle offset would equal 0.5 as a means to separate stance legs from swing legs for the policy in addition to providing information about ϕ. If $F_j = \texttt{False}$, the cycle offset would be set to 0.0.

The following, called *clock inputs* p_i, were inputted to the actor and critic:

$$p_j = \left\{ \sin\left(\frac{2\pi(\phi + \theta_j)}{L} \right) \right\}, \quad \text{for } j = \{ \text{ FR, FL, RR, RL } \} \tag{16}$$

In addition to the clock inputs, information about the phase ratio r was also inputted to the actor and critic networks. The state space $\mathbb{S}$ is defined by:

$$\mathbb{S} = \begin{cases} \hat{q} & \text{Base position, orientation, joint angles} \\ \dot{\hat{q}} & \text{Base vel, base angular vel, joint vel} \\ r, p_j \ (j = \{ \text{ FR, FL, RR, RL } \}) & \text{Ratio and Clock inputs} \end{cases}$$

Figure 5 shows the entire training pipeline between PPO + State History and the PyBullet Mini-Cheetah Simulator.

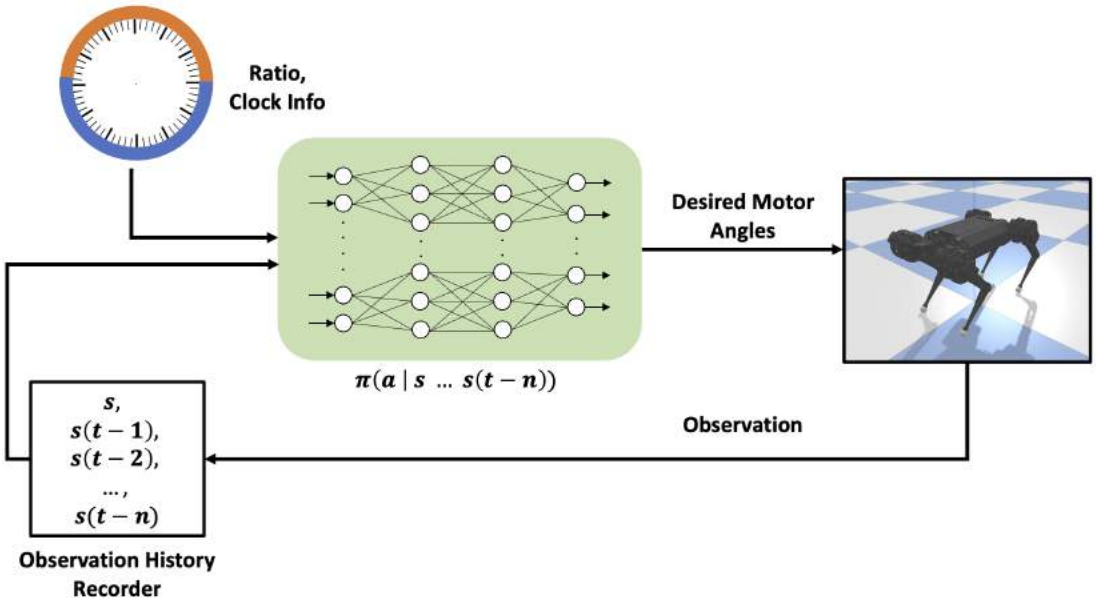

Figure 5: Control diagram of PPO + State History and Mini-cheetah simulator.

5 Training Details

5.1 Hardware

Training was performed on a Intel i5 8 GB RAM Microsoft Surface Book and an Intel i7 Dell XPS 15 9500. The former can perform approximately 5 million training steps in one day while the latter can perform approximately 10 million in one day. This was an enormous bottleneck in the training process of this Masters project since over 150 million steps [10] were needed to complete a training session; therefore, to facilitate the process of hyper-parameter tuning and reward function tuning, I decided to use an approximation of a quadruped in a plane to validate my algorithm changes.

5.2 Approximation of 3D Quadrupedal Motion

Since RL is a delicate process that is sensitive to hyper parameter tuning, it is imperative that an optimal set of hyper parameters for PPO are used. This meant that a hyper parameter sweep was necessary, and this process took many tries to complete. In addition, the MIT Mini-Cheetah has two additional motors than the Cassie robot, which makes the action space much more complex and thus requires a much longer training horizon. By extrapolation via the curse of dimensionality, training could take around 20-30% more training iterations than those in [10]. Therefore, I decided to use a planar approximation of a quadrupedal robot instead.

A great candidate for a planar approximation is the well-known Half Cheetah environment from the Mujoco suite [11]. The dimensionality of the Half Cheetah is exactly half of the Mini-Cheetah, which brings the action space down to 6 dimensions instead of 12. In addition, the Half Cheetah's actuators are placed in the exact configuration of the MIT Mini-Cheetah. The Half Cheetah has one motor each for hip abduction, shoulder rotation and knee rotation for each leg which is the same for the Mini-cheetah. For these reasons, the planar approximation of forward quadrupedal dynamics is valid.

5.2.1 Trivial Pre-Experimental Testing

Since I changed multiple aspects of the algorithm presented in [10], namely the lack of a LSTM and how contact rewards are calculated, I wanted to make sure that the algorithmic changes were engineered properly. In RL, it is good practice to test one's algorithm on the simplest task that can be achieved, so for my case, this means getting the Half Cheetah to stand as still as possible. In order to

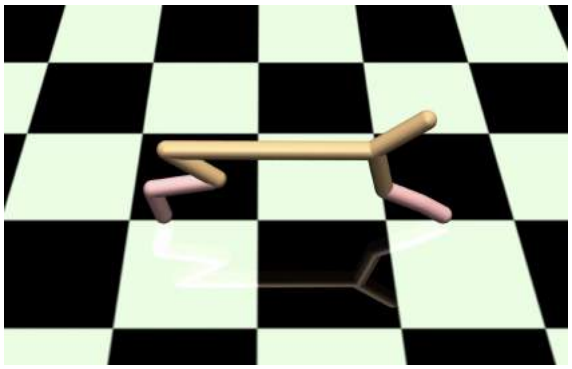

Figure 6: Half Cheetah standing still in mujoco using periodic reward composition.

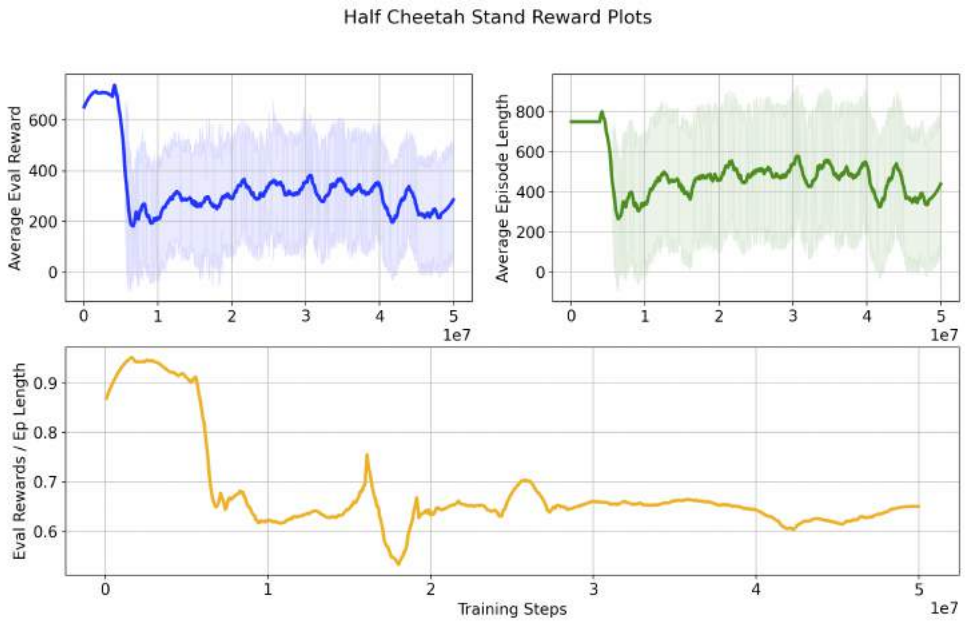

Figure 7: Reward plots for Half Cheetah standing experiment using periodic reward composition. By half a million training iterations, the Half Cheetah had already learned how to stand, but due to over-fitting, the reward slowly decreases. The total reward per timestep (Eqn. (15)) is normed between 0.0 and 1.0, so the lower plot indicates how well on average the Half Cheetah does at a particular training time step t. However, based on how the reward was tuned, a reward threshold of 0.5 was defined as satisfactory for the task. It can be seem that the policy reward per time step never dipped below 0.5 on average.

achieve this, all the weights in the policy must approach zero which would result in an output of all zeros. This is the default standing configuration of the Half Cheetah.

In order to use periodic reward composition as a framework for standing, the swing ratio r was set to 1.0, and the boolean array $\mathbf{F}$ was set to `[True, True]`, which means the legs would be stance legs for the duration of $\phi \leq r$, which in this case is the entire period.

In figure 6, it can be seen that the policy successfully learns how to stand using periodic reward composition. Figure 7 shows the rewards during the training and evaluation.

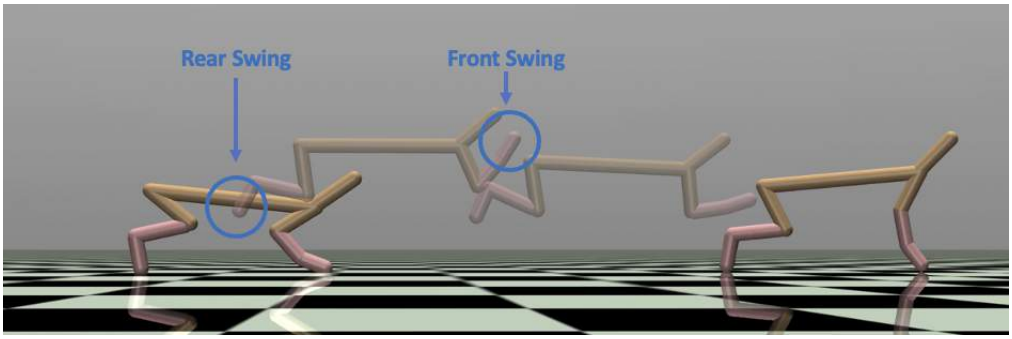

Figure 8: Half Cheetah performing a pronking gait. Here, one can see that both rear and front legs immediately lift up and then hit the ground once the swing phase is over.

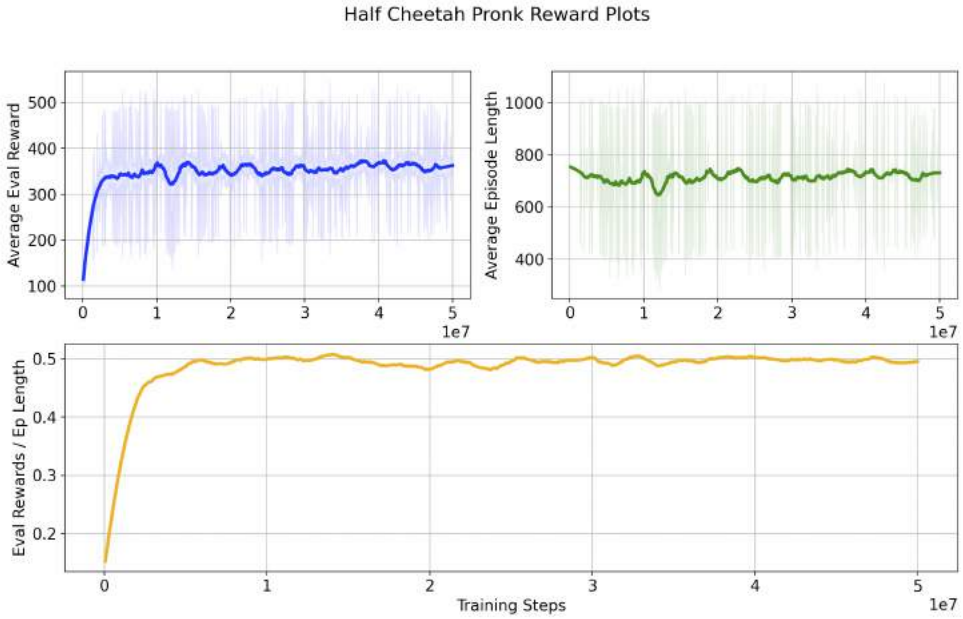

Figure 9: Half Cheetah pronking gait reward plot.

5.3 Training and Evaluation

The training process for the Half Cheetah for non-stationary gaits was set up in a fashion very similar to the process described in Section 5.2.1.

The swing ratio r was selected uniformly from a range in [0.45, 0.7] for each episode start and the starting stance leg boolean array $\mathbf{F}$ was communicated to the low level Half Cheetah simulator from the PPO algorithm. Since the Half Cheetah is a 2D approximation, some 3D gaits will appear to be identical since there is no temporal leg shift in the third dimension. Namely, the gaits pace, trot, and gallop all look the same in 2D, so the name of the gait could be chosen to be whichever; hence, I just went with gallop for naming convention's sake. $\mathbf{F}_{\text{gallop}}$ was set to `[True, False]`. In addition to the alternating leg gaits exists the non-alternating gait for a quadruped, which is called a pronk; therefore, $\mathbf{F}_{\text{pronk}} =$ `[False, False]`.

For the PPO policy, a batch of six trajectories were used, and each worker had a trajectory length of 100 timesteps. For the exact form of (15) used during the training process, please see Table 2.

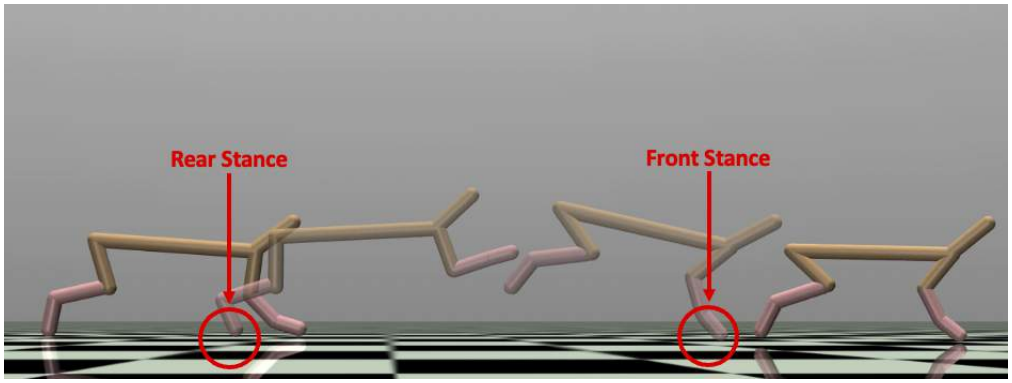

Figure 10: Half Cheetah performing a galloping gait. Here, one can see that the Half Cheetah starts with the front leg in swing phase while the rear leg is in stance phase. As $\phi \to r$, the front leg then is in a stance phase while the rear leg changes to a swing phase.

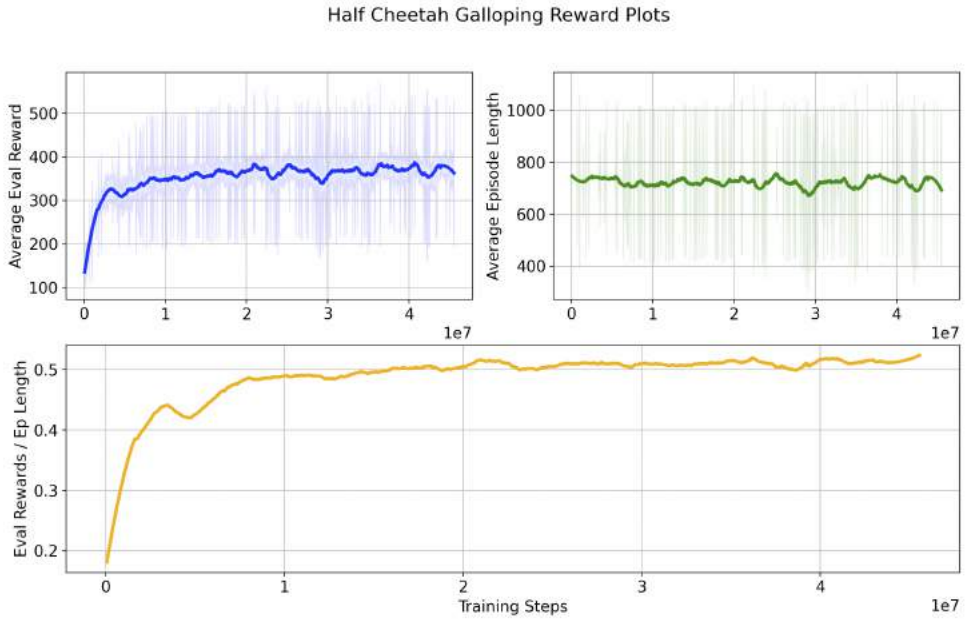

Figure 11: Half Cheetah galloping gait reward plot.

6 Results

For each gait, training took approximately 2-3 days. To facilitate an efficient reward tuning process, the galloping gait experiment was ran in parallel to the pronking gait experiment.

For the pronking training experiment, the PPO learned how to pronk in approximately 10 million training steps. See figures 8-9. For the galloping training experiment, the PPO learned how to gallop in approximately 20 million training steps. See figures 10-11.

In the reward plots, it can be seen that the rewards converge at about 0.5, which is lower than the reward of the trivial standing experiment, but the result was satisfactory in a qualitative sense. The timing of the legs in respect to ϕ may have not been perfect and in addition, there were other terms, as seen in (15), that may have decreased in order to facilitate the desired gait.

Reward Component $q(\cdot) = e^{-\lvert E \rvert}$	E
$q(s, \phi, f)_{\text{foot frc}}$	See Eqn (11)
$q(s, \phi, f)_{\text{foot vel}}$	See Eqn (11)
$q_{\dot{x}}(s)$	$\dot{x}_{\text{des}} - \dot{x}_{\text{base}}$
$q_{\text{height}}(s)$	$z_{\text{des}} - z_{\text{base}}$
$q_{\text{orient}}(s)$	$1 - (w_{\text{base}} \cdot w_{\text{des}})^2$
$q_{\text{torque}}(s)$	$u^T \cdot u$
$q_{\text{base accel}}(s)$	$\frac{\dot{x}_{\text{base}}(t) - \dot{x}_{\text{base}}(t-1)}{\Delta t}$
$q_{\text{action diff}}(s)$	$u(t) - u(t+1)$

Table 2: Reward component formulas used during Half Cheetah Training Process. Here, w is the quaternion of the Half Cheetah base. Note that the yaw component is always zero since the Half Cheetah is fixed in the XZ plane. t is the current training time step, not the counter time k which is used to increment ϕ. u is the action array which is outputted by $\pi(a|s)$.

7 Conclusion

From the results, it can be seen that periodic reward composition can indeed be used on quadrupedal systems. This conclusion is valid since the planar approximation of the Half Cheetah approximate the forward dynamics of the MIT Mini-Cheetah (or other common quadruped robots with similar motor configurations). Without the use of LSTMs, but rather state history, a vanilla MLP policy for PPO was able to learn how to stand, pronk and gallop.

8 Future Work

The next steps would be to train the full 3D system when the lab receives a satisfactory compute cluster as the 3D system will need 32 trajectories, compared to only 6 for the Half Cheetah. Once the 3D system is successfully trained, the next steps would be to use dynamics randomization to facilitate the transfer of sim-to-real.

9 Thanks and Acknowledgements

I would like to thank my adviser Professor Koushil Sreenath and my mentor Bike Zhang for their unwavering support during these unprecedented times of COVID. I could not have done this without them. In addition, I'd like to thank my lab colleague, Ayush Agrawal, for his insightful reinforcement learning advice and discussions.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[3] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert,

Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6295–6301, 2019.

[6] Erwan Lecarpentier and Emmanuel Rachelson. Non-Stationary Markov Decision Processes, a Worst-Case Approach using Model-Based Reinforcement Learning, Extended version. *arXiv e-prints*, page arXiv:1904.10090, April 2019.

[7] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals, 2020.

[8] Bastiaan Quast. rnn: a recurrent neural network in r. *Working Papers*, 2016.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[10] Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition, 2020.

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

[12] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning, 2018.

[13] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonanthan Hurst, and Michiel van de Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 317–329. PMLR, 30 Oct–01 Nov 2020.