Let A Bipedal Robot Walk In Balance

Yijin Guo

Abstract

This project trains bipedal robots in a simulation environment and try to make it walk upright in balance. It utilizes isaacgym as the simulation environment and apply the robot model of Unitree robotics. During the training process, this work uses the Proximal Policy Optimization algorithm and pay more attention to the design of the reward functions to make the robots walk more balanced. The codes can be found in github.

1 Introduction

The objective of this project is to train a bipedal robot using reinforcement learning methods and enable the robot to walk upright.

This work utilizes Isaac Gym as the environment for constructing simulations of the Unitree H1 model robot. Due to hardware limitations, I record the performance of each robot at the same moment using a camera to assess the training outcomes. The results are shown as videos.

This work models the robot and its environment and employ the Proximal Policy Optimization (PPO) algorithm during the training process. After selecting appropriate reward functions and their weights, it finally achieves a superior simulation result – allowing the robot to walk with its entire body as balanced as possible.

Moreover, there are some interesting findings for the choice of some hyperparameters about training process. Different hyperparameters

This work is in fact a fundamental one for this project. There are abundant further work to do, such as applying to real robots, changing other algorithms, considering more complex terrains, and so on.

2 Method

2.1 The Design for the Environment

Legged Robot The codes are from the github repo unitree-rl-gym. It provides a unified model for the legged robots. It is capable of, given sufficient basic parameters of the robot, updating the environment based on actions, calculating and returning the environment's observations and rewards. Different robots set their respective parameters according to their own joint characteristics, quantity, and requirements for rigidity and friction.

For H1 Robot, it has 19 joints in total and needs to set their stiffness and damping. Regarding the reward function, it takes into account various properties such as the robot's speed and angular velocity, friction and balance, joint movement, and collisions. Researchers can modify the weights of different reward functions according to their needs, or add required functions.

Important Configurations In my work, my goal is to ensure that the robot maintains coordination and balance while all joints of its body are capable of movement. Compared to the baseline, I have decoupled the important joints of the upper body, adjusted the initial angles of each joint, and increased the weight related to the robot's balance in the reward function(especially about project gravity), enabling the robot to maintain balance. The results can be found in the Experiments Part.

2.2 PPO Algorithm

The PPO algorithm[**?**], full name Proximal Policy Optimization, is a very popular reinforcement learning algorithm. It is an on-policy algorithm optimized based on TRPO (Trusted Region Policy Optimization)[**?**].

In TRPO, the objective function is as follows,

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t \right]$$

$$s.t.\ \hat{\mathbb{E}}_t \left[KL[\pi_{old}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right] \leq \delta.$$

Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$, so $r(\theta_{old}) = 1$. And then TRPO maximizes an objective

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right].$$

Comparing with it, PPO adopts some additional modifications for it and successfully achieves better results. Specifically, there are two PPO methods, one is called PPO-Clip and the other is PPO-Penalty.

PPO-Clip This approach adds a constraint on the original objective function to ensure that the new parameters do not differ significantly from the old ones.

The main objective is modified as

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right].$$

where ϵ is a hyperparameter, say, $\epsilon = 0.2$.

PPO-Penalty Another method is to use a penalty on KL divergence and continuously update the penalty coefficient during the iterative process.

The optimization object is

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} \hat{A}_t - \beta KL \left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right].$$

Let $d = \hat{\mathbb{E}}_t \left[KL \left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right]$, then β is updated as follows:

- If $d < \delta/1.5$, then $\beta \leftarrow \beta/2$,
- If $d > \delta \times 1.5$, then $\beta \leftarrow \beta \times 2$;

Algorithm The PPO algorithm is shown as Algorithm 1. In each iteration, N actors collect T timesteps of data. With these NT timesteps, we compute the loss and optimize it with minibatch SGD for K epochs.

Algorithm 1 PPO Algorithm

Initialize policy network π_θ, value network V_ψ, and replay buffer D
for each learning iteration **do**
    for each actor in parallel **do**
        Collect data by running policy π_θ in the environment for T timesteps
        Store the collected data (observations, actions, rewards, etc.) in D
    end for
    Evaluate the current policy and compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    for K epochs **do**
        for each minibatch of size M from D **do**
            Update policy network π_θ and value network V_ψ by optimizing the surrogate objective
            Optionally clip the surrogate objective to maintain a trust region
        end for
    Update target value network $V_{\psi_{\text{target}}}$ to be closer to V_ψ
    $\theta_{\text{old}} \leftarrow \theta$, $\psi_{\text{old}} \leftarrow \psi$
end for

In this work, I adopt both of the approaches for the training. The codes are from the github repo rsl-rl.

3 Experiments

3.1 Set-up

Visualization Method This work uses Isaac Gym as the environment for constructing simulations of the Unitree H1 model robot. However, although issacgym provides a visualization window, I fail to use it considering my hardware configuration. As a result, I turn to cameras in the issacgym simulation environment. For each robot, I set a camera at the right side of a robot with an appropriate posture, and records the moving of the robot in many pictures. The camera posture and the visualization result is as Figure 2. Then I concatenate the continuous images into a video.

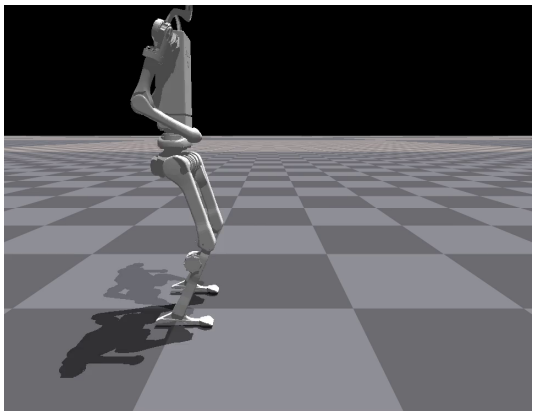

Figure 1: Visualization

In my work, I obtain a 20-second video for each simulated robot with fps(frames per second)=10.

Parameters After my adjustment, I finally set 19 actions and 69 observations, and choose the control type as P (position). It trains the environment for 1500 learning iterations, with $T = 24$(timesteps per iteration) . Other detailed parameters for the joints can be found in my github repo.

3.2 Results and Analysis

The results of this work is mostly displayed in video. All the videos used in this mini-paper can be found in github repo.

Overall Demonstration After training, most of the robots in the simulation environment can walk smoothly with its whole body. The joints of the upper and lower parts of some robots can move with a well-matched frequency.

A typical example of an ordinary robot is as shown in Video 1:

As the video shows, the whole body of this robot is acting together when moving forward.

Balance and Coordination I modify the configuration for the rewards on the robot orientation, which means increase the coefficient of corresponding reward function. This modification can be found in the class "rewards" here.It ends up working well for the balance of the robots.

Here is a typical example in Video 2.

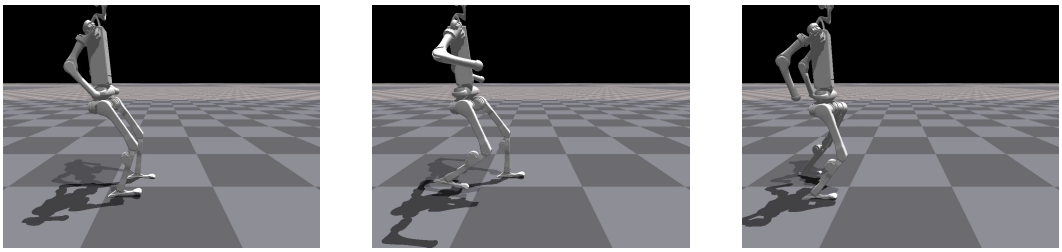

Figure 2: This is a partial screenshot from a video. A robot that is about to fall backward can adjust its balance in time by the extent to which it moves its legs backward.

The first video is the baseline. Based on the movement trends of these two robots, they are about to fall backward. And evidently the second one makes a noticeable retreat with its left leg, which quickly restores its balance.

Parameters for training During the training process, I have modified some hyperparameters for better performance. There is one interesting phenomenon when T(timesteps per iteration) is larger. Larger timesteps in one iteration mean more data collected in each actor, which will bring more stable value functions and advantage functions, and benefit a lot for the balance.

Take the robot shown in the last part as an example. For the robot in Video 2, its actions turn to Video 3 when T is changed from 24 to 50.

It is obvious that in the latter video, the robot reacts more quickly to the crisis of falling backwards. This findings can be a good direction in future for better performance.

4 Future Work

There are lots of future work to do based on this fundamental work.

- The results above are shown in a simulation environment. I can attempt to apply the training results to a real robot.
- The configuration of the scene is just flat floor. More complex terrain designs can be considered, such as steps, steeper ground and so on.
- Except PPO, there are many other reinforcement learning algorithms can be adopted in this work, for example, SAC[**?**] and RMA[**?**]. I can attempt some and find out a better algorithm for this project.

5 Conclusions

This work achieved the upright walking and balance maintenance of a bipedal robot in a simulated environment through modeling and training. The parameters of the model are adjusted to a certain extent and it finally achieves good performance. Some findings are helpful for further optimization.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[2] P Moritz M I Jordan Schulman, S Levine and P Abbeel. Trust region policy optimization. *arXiv preprint arXiv:502.05477*, 2015.

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[4] Deepak Pathak Jitendra Malik Ashish Kumar, Zipeng Fu. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.