EXPLORATIONS ON THE RAPID MOTOR ADAPTATION METHOD

Keya Hu
Department of Computer Science
Shanghai Jiao Tong University

ABSTRACT

Quadruped robots have gained increasing practical applications. Traditionally, training legged robots involves developing a reinforcement learning (RL) controller in a simulated environment and transferring it to the real world, a process that faces challenges due to discrepancies between simulation and reality. This paper leverages the Rapid Motor Adaptation (RMA) method to address these challenges. RMA uses two modules for training in simulation: a base policy and an adaptation module, which allows smooth real-world transfer without extensive physical trials. To enhance the RMA method, three experimental settings were explored: varying policy structures using different hidden layers in MLP and CNN, adjusting the number of reuse epochs for trajectory data to optimize training, and increasing training terrain difficulty to improve adaptability to diverse environments.

1 INTRODUCTION

The quadruped robot has made much progress and gained more and more practical applications nowadays. The standard paradigm of training the legged robots is to train an RL-based controller in a physics simulation environment and then transfer to the real world using various sim-to-real techniques Tobin et al. (2017); Hwangbo et al. (2019). However, the transfer is quite challenging since the real world is considerably different from the physics simulator, and the physics simulator can not accurately capture the physics of the real world. It's also infeasible for the quadruped robot to collect walking data in the real world since even 3-5 mins of walking may cause damage to the robot if it fails often.

In our paper, we adopt the rapid motor adaptation(RMA) Kumar et al. (2021) method to solve this problem. The RMA method proposes two modules to train the quadruped robot in the simulation environment: a base policy and an adaptation module. The training is separated into two stages. In the first stage, it trains the base policy of the legged robots. In the second stage, it trains the adaptation module, which takes the historical actions and states as input and outputs the environment's latent vector to help the quadruped robot decide its next action. By using this strategy, the quadruped robot can be trained only in the simulation environment and directly transfer to the real world smoothly without carrying out multiple experiments in the physical world.

To better utilize the RMA method, we do three settings of experiments to improve the given baseline: try different policy structures, test the effects of the number of reuse epochs for the group of trajectory data of every parameter, and use different training terrains to improve its performance.

1. We try different hidden layers of the multilayer perceptron (MLP) and convolutional neural networks (CNN) to find the best network structure for the policy function to improve its moving ability and reduce the training time.
2. We vary the number of reuse epochs to see its effects on the training learning curve and gain the best resuing number for training.
3. We slightly increase the difficulty of the training terrains to help the four-legged robot dogs better adapt to various kinds of environments.

2 METHODS

2.1 PPO FOR POLICY LEARNING

We utilize the proximal policy optimization(PPO) Schulman et al. (2017) algorithm to learn the policy model of the legged dog. The PPO algorithm is based on the actor-critic training structure. The actor function, which means the policy function, takes the previous trajectories as inputs and outputs the action of the next step to maximize the rewards. The critic function, which means the value function, takes the current state as input and outputs the estimated advantage of the state. The actor and critic functions share the same network structure.

The PPO algorithm is based on the policy gradient method,

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (R(\tau^n) - b) \nabla \log p_\theta(a_t^n | s_t^n)$$

where $b = V_\phi(s_t) \approx \mathbb{E}[R(\tau)]$. Here, b serves as a baseline to prevent the possibility that $R(\tau^n)$ is always positive Fei (2022). $A^\theta(s_t, a_t) = R(\tau^n) - b$ is the advantage function, which represents the average advantage of taking action compared to other actions. In my code, $b = V_\phi(s_t)$ is the critic function.

If we simply adapt the policy gradient method, the obvious shortcoming will be the slow update of the parameter since we should interact with the environment to collect the new data for each parameter update. Therefore, we can utilize an off-policy way to update the policy's parameters. We can reuse the collected data for several times. We denote the policy parameter we use when we collect the data as $\theta^{'}$ and the parameter now as θ. If we want to reuse the old data, we should utilize the important sampling to modify the bias between them. The policy gradient with important sampling is written as

$$\nabla \bar{R}_\theta = \mathbb{E}_{\tau \sim p_{\theta^{'}}(\tau)} [\frac{p_\theta(\tau)}{p_{\theta'}(\tau)} (R(\tau^n) - b) \nabla \log p_\theta(\tau)] = \sum_{t=1}^{T} \frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A_t(s_t, a_t)$$

Also, the difference between p_θ and $p_{\theta'}$ should not be too large to apply the important sampling. Therefore, we introduce KL divergence as a penalty Weinan Zhang & Yu (2022) for the distribution difference between the two versions of parameters. Then, we will get the gradient we use in our experiment.

$$\nabla \bar{R}_\theta = \sum_{t=1}^{T} \frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A_t(s_t, a_t) - \lambda KL[\theta, \theta^{'}]$$

The pseudo-code of the PPO algorithm is shown here: Algorithm 1. We use this algorithm to train the actor and critic functions.

2.2 RMA FOR ADAPTATION

We use the method of rapid motor adaptation(RMA) Kumar et al. (2021) to train the adaptation model that helps the legged robots only trained in the simulation environment to have better performance in the real environment. The RMA method can help the legged robots perform well without training them in the real environment for many iterations, which are not only time-consuming but also expensive. The structure of RMA is shown in Figure 1. The training part includes two phases. The first phase in our code is called the train teacher phase, and the second phase is called the training student phase.

The training-teacher phase. The method will train an environment encoder μ and the actor-critic network in this phase. The environment encoder μ takes the environment vector $e_t \in \mathbb{R}^{17}$ as input and outputs the encoded latent vector z_t.

$$z_t = \mu(e_t)$$

The actor function, which is the policy function π then takes the latent vector z_t, the last action $a_{t-1} \in \mathbb{R}^{12}$ and the current state $x_t \in \mathbb{R}^{30}$ as inputs and output the next action a_t.

$$a_t = \pi(x_t, a_{t-1}, z_t)$$

Algorithm 1 Proximal Policy Optimization

for $i \in \{1, \ldots, N\}$ **do**
  Run policy π_θ for T timesteps, collecting $\{s_t, a_t, r_t\}$
  Estimate advantages $A_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} - V_\phi(s_t)$
  $\pi_{\text{old}} \leftarrow \pi_\theta$
  for $j \in \{1, \ldots, M\}$ **do**
    $J_{\text{PPO}}(\theta) = \sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} A_t - \lambda KL[\pi_{\theta_{\text{old}}}|\pi_\theta]$
    Update θ by a gradient method w.r.t. $J_{\text{PPO}}(\theta)$
  end for
  for $j \in \{1, \ldots, B\}$ **do**
    $L_{\text{BL}}(\phi) = -\sum_{t=1}^{T} \left(\sum_{t'>t} \gamma^{t'-t} r_{t'} - V_\phi(s_t) \right)^2$
    Update ϕ by a gradient method w.r.t. $L_{\text{BL}}(\phi)$
  end for
  if $KL[\pi_{\theta_{\text{old}}}|\pi_\theta] > \beta_{\text{high}} KL_{\text{target}}$ **then**
    $\lambda \leftarrow \alpha\lambda$
  else if $KL[\pi_{\theta_{\text{old}}}|\pi_\theta] < \beta_{\text{low}} KL_{\text{target}}$ **then**
    $\lambda \leftarrow \lambda/\alpha$
  end if
end for

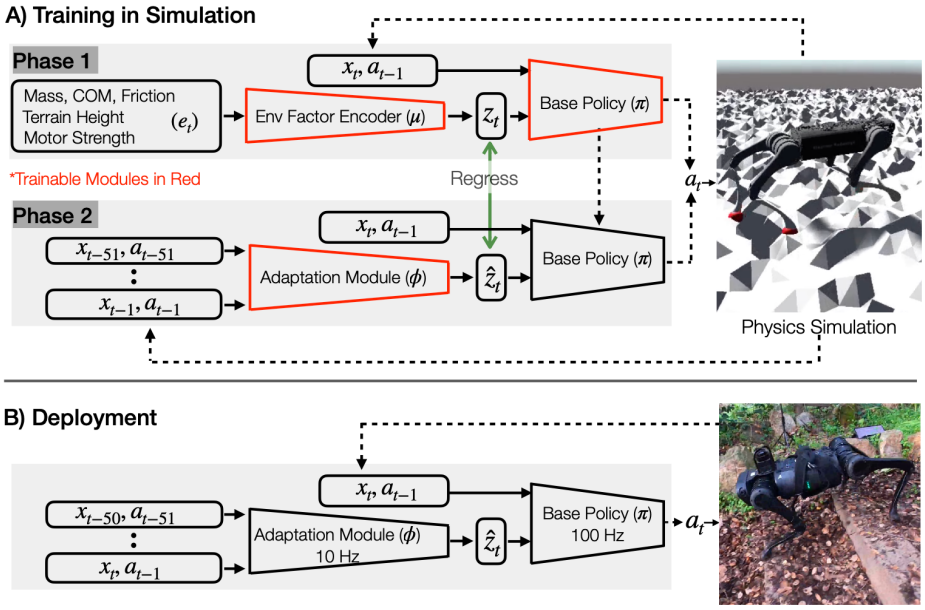


Figure 1: The structure of the training phases of the RMA method.

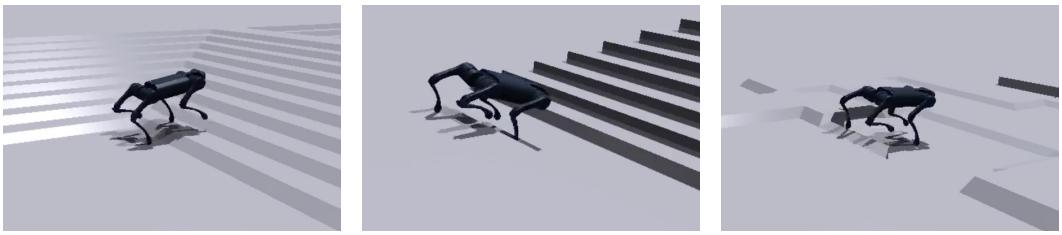
The critic function, which is the value function V, takes the same input as the policy function to output the estimated average advantage value.

$$b_t = V(x_t, a_{t-1}, z_t)$$

We use the PPO algorithm to update the actor-critic function during the training-teacher phase. Meanwhile, we also update the parameters of the environment encoder. We train these two modules for 2000 iterations.

The training-student phase. In this phase, we will train an adaptation module ϕ to take the actions $a_{t-k:t-1}$ and the recent history of the robot's states $x_{t-k:t-1}$ to generate $\hat{z}_t$ which is an estimate of the latent vector of the true environment vector z_t.

$$\hat{z}_t = \phi(x_{t-k:t-1}, a_{t-k:t-1})$$



(a) Stairs up terrain. (b) Stairs down terrain. (c) Discrete terrain.

Figure 2: Three types of terrain we used in our experiments.

We train the adaptation module ϕ by minimizing the following loss.

$$\mathcal{L}(\hat{z}_t, z_t) = \text{MSE}(\hat{z}_t, z_t) = ||\hat{z}_t - z_t||^2$$

where $z_t = \mu(e_t)$ and μ is the environment encoder we train in the training-teacher phase.

The deployment phase. The method directly deploys the legged dog with the well-trained policy function and the adaptation module in the real environment. In the experiment, we only do the simulation part instead of the real environment because of the limitation of time and device.

3 EXPERIMENT SETTING

We use the a1 quadruped robot dog as our experiment entity. The simulation environment is based on the Isaac Gym to capture the physical interactions in the real world. We use the simulator to generate five types of terrain: smooth slope, rough slope, stairs up, stairs down, and discrete. Among these terrains, we find that the smooth slope and rough slope are too easy and can not separate the performance of different policy structures. Also, it is unlikely for the real world to be a simple slope. Therefore, in the following experiments, we only test our structures on stairs up, stairs down, and discrete terrains, as Figure 2 shows. Among all these terrains, the discrete terrain seems to be the most difficult task for a1 robot dogs.

We use two ways to evaluate the performance of our trained robots. The first evaluation metric is the reward tracking linear velocity, which means

$$\exp\left(-c \cdot ||v_{\text{real}} - v_{\text{command}}||^2\right)$$

where c is a tracking velocity constant. It demonstrates its ability to move forward while encountering various kinds of obstacles. The second evaluation metric is the gaits of the legged robot dogs, which are visualized by the Isaac Gym. We take the gaits that appear more normal and help the dog to adapt to various environments as the good ones.

In our experiments, we will research three parts of the RMA algorithm. Firstly, we will do experiments on various structures of multilayer perception(MLP)Haykin (1994) and convolutional networks (CNN) Kim (2014). Secondly, we want to find a suitable number of epochs for reusing the data collected for one specific parameter. Therefore, we will vary the epochs and see their learning curves' performance. Lastly, we balanced the portion of different types of terrains during training to gain a better performance against the baseline setting. We mainly change the portion of stairs-up and discrete terrains since they are the most challenging kinds of terrains in the tasks for a1 legged robot dogs.

4 EXPERIMENTAL RESULT

4.1 THE PERFORMANCE OF DIFFERENT POLICY NETWORK STRUCTURES

We set various policy structures to find the one most suitable for legged robots. We choose the most common structures, MLP, and CNN structures, to do experiments.

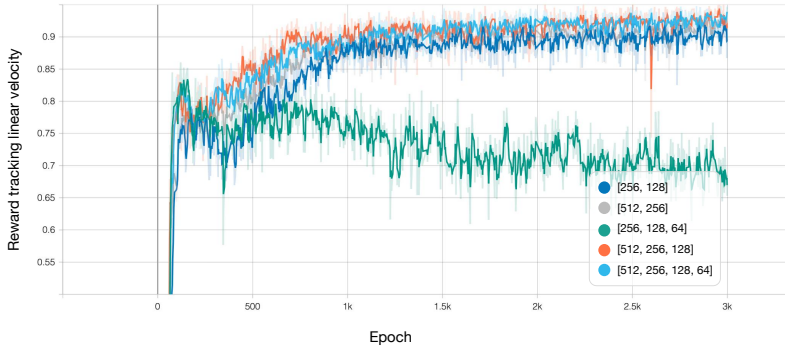


Figure 3: The learning curve of training teacher phase with different structures of MLP.

CNN structure / rew_tracking_lin_vel	500	1000	2000	3000
CNN with linear	**0.8295**	**0.8646**	**0.8887**	**0.9071**
CNN without linear	0.5523	0.5075	0.5890	0.6078

Table 1: The reward of tracking linear velocity of each training epoch of CNN structure network with and without the linear layer.

4.1.1 MLP STRUCTURE

We try different policy and value networks structures. We vary the hidden layers of the MLP structure and trace its learning curve while training as Figure 3 shows.

We can see that the performance of various MLP structures is quite stable with various hidden layers except $[256, 128, 64]$, which results in the worst performance. Among all these hyperparameters, $[512, 256]$ seems to be the best hyperparameter compared to the baseline $[512, 256, 128]$. Its final reward of tracking linear velocity is 0.9209 while the baseline is 0.9143, and it tasks 1 hour 1 min while the baseline takes 1 hour 15 min to finish training. Therefore, it not only reaches better performance in the last epoch but also trains faster with a simpler structure.

4.1.2 CNN STRUCTURE

We also apply the CNN structure to the policy network. We set two hidden CNN blocks, the first layer with 16 blocks, each block has kernel size$= 8$ and stride$= 2$, and the second layer with 8 blocks, each block has kernel size$= 4$ and stride$= 1$. We apply a linear projector with dimension 512 before the CNN layers for one structure, and the other one directly applies CNN blocks after the input. We get the results in Table 1, which shows the reward of tracking linear velocity of training epoch $500, 1000, 2000, 3000$.

The table demonstrates that the CNN structure can not perform well without the linear layer. With the linear structure, the CNN structure can achieve a performance similar to that of the MLP structure. Therefore, the linear layer seems quite important for the legged robots, in the following experiments, we will adopt MLP as the base structure of the policy function.

4.2 THE EFFECTS OF THE NUMBER OF LEARNING EPOCHS FOR ONE DATA COLLECTION.

To better utilize the collected data, the PPO algorithm will use each data trajectory of each parameter more than one time, as mentioned in the method section. Therefore, in our experiments, we also analyze how many epochs of learning can best benefit our learning. We set the epochs to $1, 2, 5, 10, 20$ and use a linear structured policy network for training. Figure 4 shows the learning curve of the reward tracking linear velocity.

The figure demonstrates that the best number of epochs are ranging between $[5, 10]$. The baseline adapts the setting of epoch$= 5$. We find that epoch$= 10$ can reach the final reward of tracking

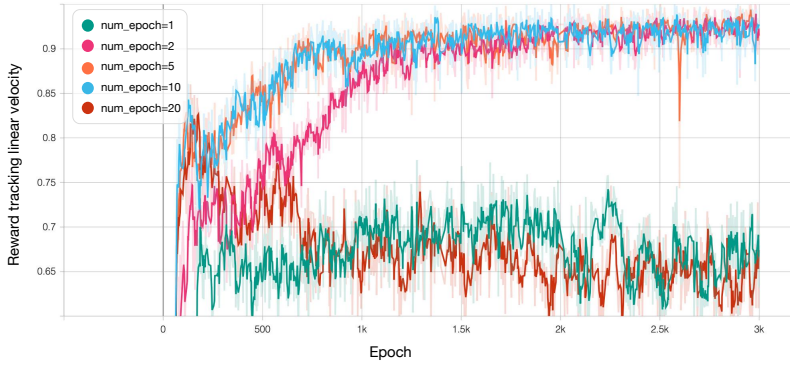


Figure 4: Learning curve for the reward for tracking of linear velocity commands of different learning numbers of epochs for each collected data.

training terrain / rew_tracking_lin_vel	stairs up	stairs down	discrete
baseline	**0.9637**	0.9555	0.8773
balance discrete and stairs	0.9622	**0.9591**	**0.9184**
all discrete	0.1254	0.6508	0.6872
more discrete less stairs	0.6508	0.9036	0.7437

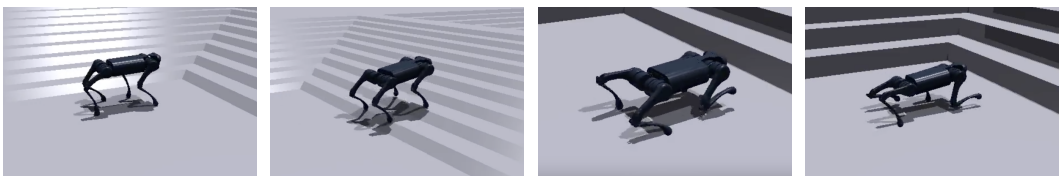
Table 2: The reward of tracking linear velocity of different training terrains.

linear velocity of 0.9256, which is slightly higher than the baseline with 0.9143. For the smaller number_epoch$= 2$, it takes more epochs to achieve better performance. On the one hand, if we do not reuse the collected data, the performance of the trained-legged robot can not be improved with the increment of the epochs since the data are not fully utilized. On the other hand, if we reuse one group of specific data for too many epochs, such as 50 epochs, their distribution will become increasingly different, which will also result in the training curve's poor performance.

4.3 Balance the difficulty of the training terrain

In our experiments, we find that discrete terrain seems to be the most difficult task for the trained robots. Therefore, we want to find out whether increasing the portion of discrete terrains in the training part will help the trained robots perform better in all three terrains. The results are shown in Table 2.

In the baseline setting, we set the terrain portion as 20% slope, 35% stairs up, 25% stairs down, and 20% discrete. In the balance discrete and stairs setting, we set the terrain portion as 50% stairs up, 10% stairs down, and 40% discrete. In the more discrete, less stairs setting, we set the terrain portion as 20% slope, 20% stairs up, 20% stairs down, and 40% discrete. We set the terrain portion as 100% discrete in the all discrete setting. Since discrete terrains are randomly generated, we do the experiments five times and average the results.



(a) Baseline. (b) Balance discrete. (c) More discrete. (d) All discrete.

Figure 5: The legged dogs' gaits after training on different terrains.

The table demonstrates that with balanced discrete and stairs-up terrains, the legged robots can achieve a similar great performance as a baseline and do much better on discrete terrains. However, the stairs-up terrains are quite important since if they become fewer, the legged robots' performance will drop dramatically on all terrains.

We also visualize the gaits of legged robots after training in Figure 5. We can see that with more discrete terrains, the legged robots will tend to lower their bodies to keep their balance. However, it will prevent the robots from climbing up more steep stairs. Only with the balanced discrete and stair terrains can the robots learn both to stand up to climb the stairs and to keep their gravity of center from failing when encountering discrete obstacles as Figure 5b shows.

5 Future Work

There are several directions that remain to be explored besides the experiments we have done.

1. Try more policy structures, such as RNN and LSTM. We can explore whether long-term memory will help us learn better policy functions.
2. Add more difficult terrains in the training to better equip the legged robots with the ability to transfer to various kinds of real-world environments.
3. Improve the structure of the adaptation module and do real-world experiments to verify its ability.

6 Conclusion

In the experiments, we use the RMA algorithm and do three experiments to improve its ability against the baseline. Firstly, the MLP structure shows a great advantage in the task. With a very simple hidden layer $[512, 256]$, it can achieve the best performance while using the least training time. Secondly, the number of reuse epochs can also affect the experiment results a lot. Too small and too large numbers are both not helpful. The best epoch seems to be ranging between 5 and 10. Lastly, increasing the portion of stairs up and discrete terrains during training in a balanced way can help the robots perform better in all kinds of terrains. It guides the robot dogs to stand up and move normally while maintaining a good balance when encountering discrete obstacles.

Acknowledgments

Thanks to Professor Weinan Zhang for teaching, teaching assistant Xialin He for giving advice, classmates Xinyao Li for helping to visualize the legged robots, and Yichao Zhong for giving advice on the evaluation metric.

References

Duoduo Fei. Ppo algorithm for reinforcement learning, 2022. URL `https://zhuanlan.zhihu.com/p/468828804`.

Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

Yoon Kim. Convolutional neural networks for sentence classification, 2014.

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

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

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Jian Sheng Weinan Zhang and Yong Yu. *Hands-On Reinforcement Learning*. 2022.