Investigating Overconstrained Quadruped Locomotion using Reinforcement Learning

Guojing Huang
Robotics Engineering
12111820

Jinda Dong
Feedback Controlling
12111829

Xinyan Ju
Automation
12111142

Junwei Lv
Robotics Engineering
12212928

Zhou Chen
Automation
12111812

Shengyang Ming
Robotics Engineering
12111717

***Abstract*—This paper investigates overconstrained locomotion on a particularly designed and manufatured overconstrained quadruped, from a learning perspective with the help of reinforcement learning, aiming to exploit its potential in motion and energy consumption to advance the field of legged robots. We use a overconstrained Bennett quadruped as the robotic carrier, which invented by the Bionic Design and Learning lab from SusTech. Then, we employed a large-scale, multi-terrain deep reinforcement learning framework to train the overconstrained quadruped for a comparative analysis of different training results in rewards, velocity and joint torques. Results show that the trained overconstrained quadruped is capable of walking over different terrains, including pyramid-like stairs, gravel, and changing slopes, at least as common quadruped robots do. It proves that the overconstrained quadruped can achieve the highest speed of $1.23m/s$ on rough environments, which is feasible and acceptable. This paper paves the path for furthur optimal trainning and sim-to-real process of overconstrained quadruped using reinforcement learning.**

***Index Terms*—Bennett linkage, reinforcement learning, locomotion**

I. Introduction

A. Limitations of traditional control of quadruped robots

In the classical control methods of quadruped robots, complex control algorithms are required to coordinate the movement of the four legs to ensure their stability and smoothness. It is very difficult to design and debug these controllers. At the same time, traditional controllers are usually designed for specific tasks and specific environments. Controllers for running and climbing stairs may have significantly different architectures and need to be designed and adjusted separately. As a controller designed for a specific robot in a given terrain, it is usually not directly used in another environment. Even if it is used in a similar environment, it usually requires a large number of parameters to be adjusted to adapt to various changes[1, 2]. In addition to the complex factors of the controller, traditional quadruped robots also often require relatively complex mechanical structures and high-precision mechanical and electronic components, which increases the difficulty and cost of design and manufacturing[3]. At the same time, due to the complexity of control and high energy consumption, classically controlled quadruped robots consume a lot of energy during movement, especially in complex terrain or high-speed movement. Low movement efficiency and high energy consumption. These disadvantages greatly limit the application of robots in the real world.

B. Optimizing quadruped robot control using reinforcement learning

In order to overcome the problems of poor adaptability to the environment and changes, complex controller parameter debugging, and low robustness of traditional quadruped robots, modern quadruped robots combine artificial intelligence, deep learning, and advanced sensing technology to improve their autonomy, adaptability, and efficiency, thereby achieving the following advantages:

1) Model simplification and robustness improvement: Reinforcement learning algorithms can be trained in a variety of different environments, allowing robots to adapt to a variety of terrains and obstacles. By continuously interacting with the environment and optimizing strategies, the robot is able to move efficiently in different environments without relying on large amounts of training data.[4] In addition, unlike traditional control algorithms that require precise physical and environmental models, reinforcement learning relies on experience and feedback, so it can learn and optimize even without an accurate model. At the same time, the robot's fault tolerance is greatly improved, and it can maintain stable operation even in the face of unexpected situations (such as sensor failure or environmental changes), which greatly improves the robustness and adaptability of the robot.

2) Eficiency improvement: Reinforcement learning maximizes rewards (such as speed, stability, energy efficiency, etc.) by finding the optimal action strategy[5], which makes the robot more efficient when performing tasks, reduces energy consumption, and extends battery life. At the same time, robots trained through reinforcement learning can respond quickly in real-time environments, adapt to dynamic changes, and improve the smoothness and stability of movement. This also allows robots to quickly switch between different tasks, sometimes simply through changing the reward function[4], such as switching from walking on flat ground to climbing stairs, and perform well in each task.

In this paper, we use reinforcement learning to train a quadruped robot using data collected by the proprioceptive IMU sensor, aiming to achieve robust control.

II. Related Work

A. Robot Platform and Leg Structure

Robot platform and leg structure are of vital importance for reinforcement learning. Though these robots are all called quadruped robots, there are still lots of differences on their robot platform and leg structure.

A team at Google used the Minitaur from Ghost Robotics, a quadruped robot with eight direct-drive actuators as their robot platform. The motors could be actuated through position control or a PWM signal. An STM32 ARM microcontroller is used to send commands to actuators, receive sensors, and perform simple computations, but is not enough to execute neural network policies learned from deep RL. Thus, an Nvidia Jetson TX2 is installed to perform neural network inference[6].

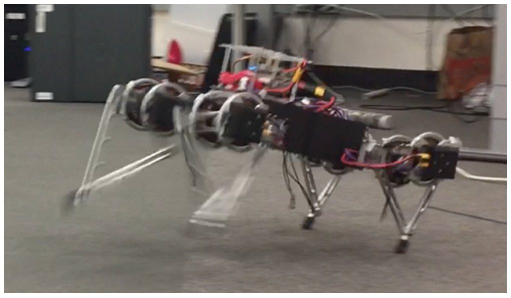


Fig. 1: The Minitaur from Ghost Robotics.

Meanwhile, a team from China used Doggo as their robot platform. The Doggo robot is an open source, complex quadruped robot from Stanford, which uses a closed-chain leg structure with two degrees of freedom for each leg structure . As its complex dynamic structure, they do not directly model it in kinematics, but analyze its leg structure and the trot gait as a prior knowledge. Each leg is composed of two links, with two degrees of freedom, which are connected by a rotating shaft, and the two links form a closed chain structure[7].

Fig. 2: The Doggo, an open source robot from Stanford

A research supported by National Key Research and Development Plan of China used a cheetah-like robot called Yobogo as their robot platform. It's a quadruped robot with 12 direct drive actuators. Each leg is controlled by three actuators that allow the foot to move in 3D space[2].

B. *Reinforcement Learning in Locomotion Problems*

Tremendous progress in deep reinforcement learning have been applied on the quadruped robots in dealing with the locomotion problems. For example, Yang et al. [4] proposed a method to make the training process faster and safer, but it requires a slew of computing ability. A team at Google used to train a quadruped robot with eight direct-drive actuators [6] on the Minitaur from Ghost Robotics (a robot platform), with a faster and lowe-cost way.

Fig. 3: The Yobogo

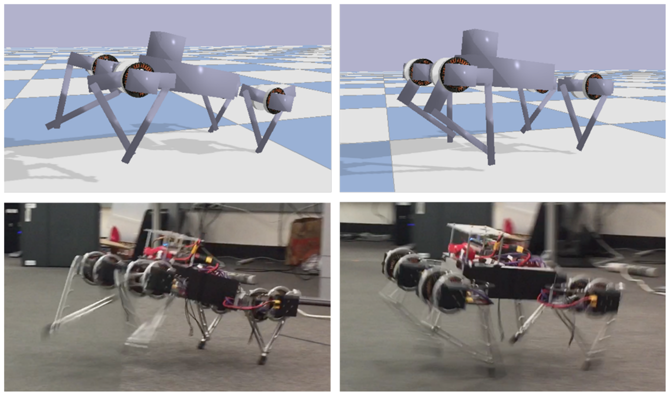


Fig. 4: The simulated and the real Minitaurs learned to gallop using deep reinforcement learning.[6]

Efficient and high-quality state estimation needs to be applied on the quadruped robot. In the previous studies, excluding redundant terms[6] or symolifying the process of system simulation[2] are examined to be good solutions. In a previous study[6], The yaw of the base provided by the IMU sensor drifts quickly, and the direct result of the velosity from the motor contains too much noise, leading to higher uncertainty. So using position control to actuate the motors of the Minitaur can somehow deal with this problem.

Besides, the design of the observation space also plays an important role to narrow the reality gap. If the observation space is high dimensional, the learned policy can easily overfit the simulated environment, which makes it difficult to transfer to the real robots. ‘ The team tested the system with galloping and trotting. And Fig. 5 provides the results. Using hyper parameters search to obtain the relative best hyper parameters, and every 25 steps they collect the simulated experience for policy update in parallel.

A team[2] trained the robot step by step, that is to first test the framework by training the robot to walk forward on a flat ground, and later increase the complexity of the environment to better the adaptability of the framework on complex terrains.

To better the performance of the adaption to variety of terrians, actor-critic methods have been applied widely. Peng et

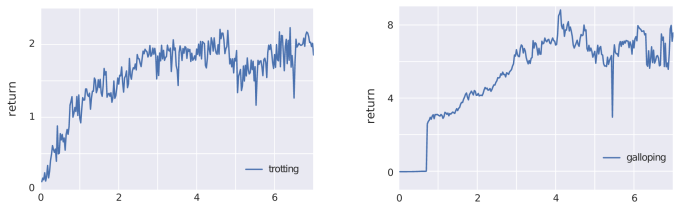


Fig. 5: The learning curves of trotting and galloping[6].

al.[8] proposed a mixture of actor-critic experts method to perceive the physical environment based on limited prior knowledge. Besides, some end-to-end controller training strategies are also proved to be in good use[9].

C. Increasing the Robustness of the Controllers

Although reinforcement learning has great potential in learning complex motion strategies from scratch and automatically adjusting the robot's motion control through simple reward signals, it does not require a lot of manual intervention and debugging. However, it cannot be widely promoted in real environments due to the "reality gap"[10, 5]. Therefore, a large number of studies have adopted a series of methods to improve the performance and robustness of the controllers.

One way to improve the robustness of the controller and thus reduce the reality gap is by simulating perturbations that may be encountered during training. Pinto et al.[10] introduced an adversarial agent to simulate various possible disturbances during training, namely Robust Adversarial Reinforcement Learning (RARL), which maximized the failure probability of the main agent by applying disturbance forces. As shown in Fig. 7, the RARL strategy exhibits remarkable robustness in dealing with variations in friction coefficient and mass. Many studies use randomized dynamic parameters[5, 6, 11, 12, 13] or domain randomization[14] techniques in the training process. Multiple parameters such as mass, friction, and inertia in physical simulation are randomized to ensure the robustness of the control strategy under different conditions. Rusu et al.[15] use progressive networks to bridge the reality gap, transferring policies learned in simulation to the real world, and demonstrate successful learning of tasks from raw visual input on a fully driven robotic manipulator. In [6], they compared baseline simulation with random perturbations during training, and the result is shown in Fig. 6.

From another perspective, many studies have tried to improve the robustness by optimizing observation space and control strategy. Tan et al.[5] designed a compact observation space to reduce the risk of overfitting of the control strategy in the simulation environment. A smaller observation space can make the observation distribution in simulation and reality more consistent, thereby narrowing the gap between reality and reality. The RLOC (Reinforcement Learning and Optimal Control) module established by Gangapurwala et al.[11] introduces the Footstep Planning strategy to generate the expected footstep position suitable for the current terrain, map the robot state and terrain information to the expected foot position,

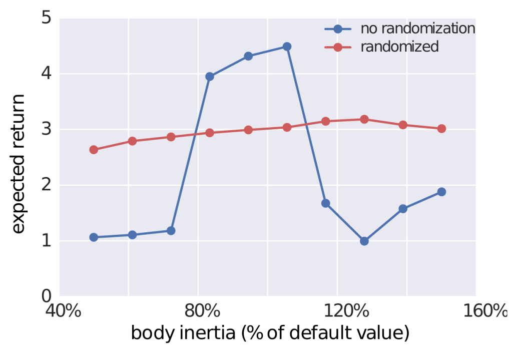


Fig. 6: Performance comparison of controllers that are trained with (red) and without (blue) randomization and tested with different body inertia[6].

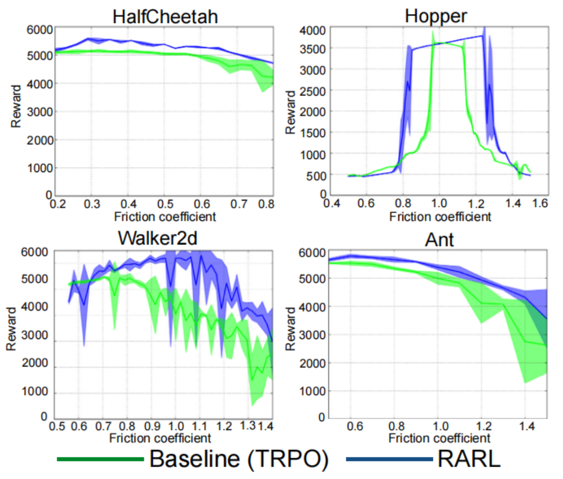


Fig. 7: Average reward values of RARL strategies under different friction coefficients. The performance of the baseline strategy (TRPO) decreases significantly, while the RARL strategy shows higher robustness.[10]

avoid complex combinatorial search, and naturally emerge adaptive behaviors during training, improving the robot's ability to move on irregular terrain. Figure shows the success rate of each planning strategy under different terrains, indicating that this strategy significantly improves the robustness of the system on complex terrains.

III. Data

A. Overconstrained Robot Model

Overconstrained linkage is a special class of linkages with more degrees of freedom than predicted by the degrees of freedom formula. Bennett linkage is over-constrained, and its degree of freedom is -2, as calculated by Kutzbach Grubler's mobility criterion. However, it can still move thanks to its particular geometrical design, which has a single DOF. The Bennett linkage is a single closed-loop four-bar spatial mechanism whose two neighboring joint axes are neither parallel nor perpendicular. When the joint axes are parallel, the Bennett mechanism is transformed into a planar four-bar mechanism. The geometrical conditions for the establishment of the Bennett mechanism are:

$$a_{12} = a_{34} = a, a_{23} = a_{41} = b \quad (1)$$

$$\alpha_{12} = \alpha_{34} = \alpha, \alpha_{23} = \alpha_{41} = \beta \quad (2)$$

$$\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \zeta \quad (3)$$

$$R_i = 0 (i = 1, 2, 3, 4) \quad (4)$$

Shown in Fig. 8(a) and Fig. 8(b) are, respectively, physical and model versions of the overconstrained quadruped provided by the Design and Learning laboratory supervised by Prof. C.Song, Which links are all designed as the alternative from following the geometric conditions defined by the Bennett linkage. Table I shows some key physical parameters of overconstrained quadruped, laying the groundwork for the smooth running of our project.

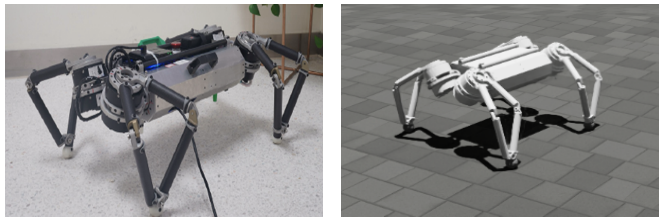

(a) physical robot (b) simulation model

Fig. 8: Two different versions of overconstrained quadruped

TABLE I: Key parameters of overconstrained quadruped

Property	Parameter
Length*Width*height(m)	0.49x0.45x0.32
DOF	12
Mass(kg)	13.8
Actuator	Unitree A1
Maximum joint torque(N·m)	33.5
Maximum joint speed(rad/s)	21
Torque constant(N·m/A)	0.9287

Because the simulation platform Nvidia Isaac Sim requires the *usd* model to simulate, we imported the *urdf* file into Isaac Sim and converted it to the *usd* format, which also contains some basic physical properties like mass and moment of inertia. To ensure the feasible running of the simulation as well as to reduce the computational burden, we preprocessed the robot model. Firstly, each joint motor is given a 100 of k_p and 10 of k_d. Then we reserved necessary collisions on the base, tips, and the 8 links closer to the ground, marked by green, as Fig. 9 shows.

B. Simulation Platform and Framework

All our work is done on Nvidia's Isaac Sim, which provides much more modern and advanced learning-based simulation environments. We use ORBIT as our learning frame to acquire a feasible reinforcement learning training interface or method and choose rsl_rl as the one.

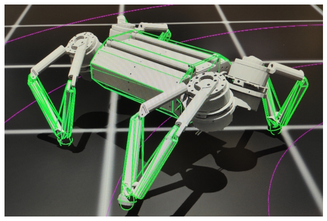


Fig. 9: Necessary collisions of the overconstrained quadruped

NVIDIA Isaac Sim is an extensible robotics simulation platform that gives you a faster, better way to design, test, and train AI-based robots. It's powered by Omniverse to deliver scalable, photorealistic, and physically accurate virtual environments for building high-fidelity simulations.

ORBIT is a unified simulation framework for interactive robot learning environments, powered by NVIDIA Isaac Sim. It offers a modular design to easily and efficiently create robotic environments with photo-realistic scenes, and fast and accurate rigid and soft body simulation. With ORBIT, we are allowed to train reinforcement learning policies and collect large demonstration datasets from hand-crafted or expert solutions in a matter of minutes by leveraging GPU-based parallelization.

RSL_RL is a fast and simple implementation of reinforcement learning algorithms designed to run fully on GPU. Based on the PPO algorithm (Proximal Policy Optimization), We can solve the problem of strategy optimization in a continuous action space. The stability of updates is ensured by limiting the distance between the new policy and the old policy in each update. This is particularly important for the case where updates are performed in a continuous action space. PPO works as follows: first, the data in the environment, including states, actions, and rewards, are collected by executing the current policy. The current policy and value function are then used to compute an estimate of the advantage of each state-action pair, i.e., the advantage of that action relative to the average. The strategy is then updated by maximizing the agent's expected cumulative reward over the past collected data. Finally, a new action is executed in the environment using the updated policy, and then data collection continues with the next round of updates. PPO is efficient and stable and ensures stability by limiting the magnitude of policy updates, which helps prevent the policy from diverging or converging to a suboptimal solution during training. The algorithm is also relatively simple, easy to implement and tune, and performs well in many real-world problems.

IV. Methods

We are going to simulate a controller that can be applied to the quadruped robot for moving in complex terrains using reinforcement learning. We propose the following ideas to obtain the RL based controller.

A. Reinforcement Learning Algorithm

In classic reinforcement learning, the environment is formulated as a Markov decision process described by the tuple

(S, A, p, r, γ) with S to be the state space, A to be the action space, p to be the state transition probability, r to be the obtained reward and γ to be the discount factor[16]. The process of the Markov is shown in (5), where $P_{ss'}$ is the state transition probability matrix.

$$P_{ss'} = P[S_{t+1} = s'|S_t = s] \tag{5}$$

The bellman equation reveals the recursive relationship between cost function and strategy in reinforcement learning[17], describes as (6), where $\pi(s)$ represents the probability distribution in state s choosing strategy a, and $R(s, a)$ stands for the current reward, s' is the new state add γ is the discount factor.

$$V(s) = E_{a\ \pi(s)}[R(s, a) + \gamma V(s')] \tag{6}$$

In order to find the policy $\pi(a|s) = P[A_t = a|S_t = s]$ that can maximize the reward, we following the basic MDP (Markov Decision Process) process[18] to implement the reinforcement learning in our model. Q-learning is a model-free method widely used in solving the MDP problems in reinforcement learning[19], using the state-action value function (Q function). We follow the Q-learning method to choose strategies that can maximize the final reward in (7) to calculate the expected rewards Q in a given state s to s' corresponds to taking the action a. Here α is the learning rate, a hyper-parameter.

$$\begin{aligned} Q^{new}(s, a) =& (1 - \alpha)Q(s, a) \\ &+ \alpha(R(s', a) + \gamma \max Q(s', a)) \end{aligned} \tag{7}$$

To maximum the adaptability of our model, we choose the rotation speed of the joints on the legs of the quadruped robot to be the control objects where the actions take on, and the center linear velocity to be the observation.

To better the performance of our model, we choose penalization terms appropriately, leading to our reward function shown in Table. III.

With our sophisticated design of the reinforcement algorithm, our RL-based model successfully converges and the simulating result show a good performance.

B. Training Method

The training process for classic reinforcement learning (shown in Fig. 10) starts with the simulation environment, and pass the current states S_t and reward R_t to the agent, following the action a_t. Then the action (the angular velocities of the joints) will be applied to the simulation environment (our quadruped robot) and step to the next state s_{t+1} and reward r_{t+1}. We follow the basic framework of the reinforcement learning, and apply novel strategies to better the performance.

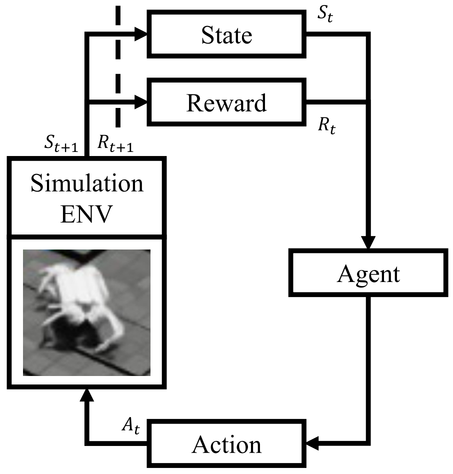


Fig. 10: Flow chat for the classic training process.

To better the performance of the model, especially the latent real-time-world performance in the future, we applied actor-critic method to training the policy. This is a two-time-scale algorithm in which the critic uses TD learning with a linear approximation architecture and the actor is updated in an approximate gradient direction based on information provided by the critic[20]. This method allow it possible to update the strategy and the cost function at the same time, which is of high-efficiency. The flow chat of our actor-critic training method is provided in Fig. 11, and part of the parameters are shown in Table. II, where an ELU activation function[21] is described in (8).

$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right) & \text{if } x \leq 0 \end{cases} \tag{8}$$

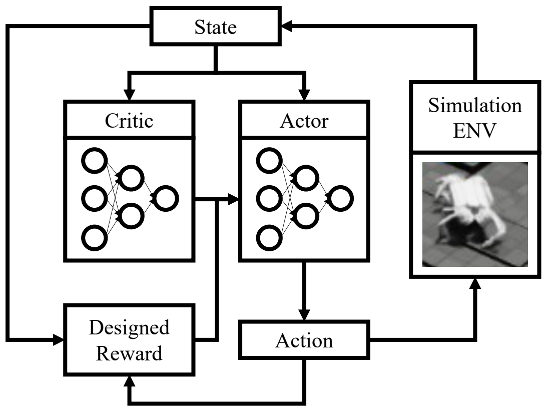


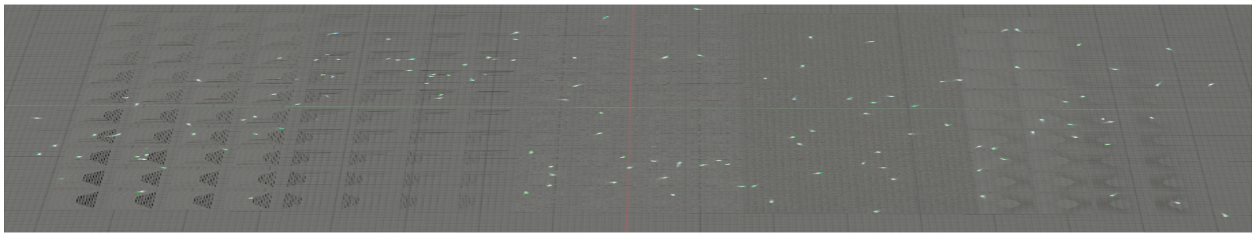
Fig. 11: Flow chat for the training process, using actor-critic strategy.

TABLE II: Parameters in actor and critic.

Hidden Layers in Actor	Hidden Layers in Critic	Activation Function
[512, 256, 128]	[512, 256, 128]	ELU function

TABLE III: Part of the Reward Terms and Explanation

Reward Terms	Explanation	Formula	Weight
track_lin_vel_xy_exp	Reward tracking of linear velocity commands (xy axes)	$\exp\{-\frac{\lvert v_x - v_x^{cmd}\rvert^2 + \lvert v_y - v_y^{cmd}\rvert^2}{0.25}\}$	1.5
track_ang_vel_z_exp	Reward tracking of angular velocity commands (yaw)	$\exp\{-\frac{\lvert\omega_{yaw} - \omega_{yaw}^{cmd}\rvert^2}{0.25}\}$	0.75
feet_air_time	Reward long steps taken by the feet	$\sum\{(t_{air} - 0.5) * (t_{air} > 0)\}$	0.01
flat_orientation_l2	Penalize non-flat base orientation	$\lvert\vec{g}_x^{base}\rvert^2 + \lvert\vec{g}_y^{base}\rvert^2$	-2.5
ang_vel_xy_l2	Penalize xy-axis base angular velocity	$\lvert\omega_{yaw}\rvert^2$	-0.01
action_rate_l2	Penalize the rate of change of the actions	$\lvert a_t - a_{t+1}\rvert^2$	-0.005


(a)

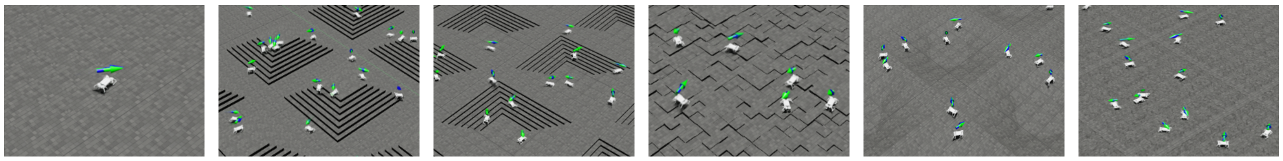

(b)

Fig. 12: **Simulation setup and individual training environment**. (a) Panoramic aerial view of multi-terrain simulation environment; (b) individual training environments: flat ground, pyramid terrain, inverted pyramid terrain, discrete random height field, continuous uniform random height field and gravel, from left to right respectively.

V. Experiments

We use deep reinforcement learning on our own workstation with Nvidia 3080Ti GPU to conduct large-scale simultaneous training of overconstrained locomotion. Our experiments aim to train a version of a policy that will enable overconstrained quadruped robots to perform highly adaptive and velocity-tracking locomotion in complex simulation environments. According to the previous introduction, we use the ORBIT framework to train velocity-tracking locomotion, where the robot's observation space includes the linear velocity, angular velocity, joint positions, joint velocities of the robot's base, and sampling heights of the terrain. We set the initial angles of hip motors, coaxial joint motors 1 (dof2) and coaxial joint motors 2 (dof3) to respectively $45°$, $25°$ and $20°$ so that the initial state of the robot resembles the posture of a reptile with a lower center of gravity and higher stability.

The multi-terrain training environment we use includes flat ground, pyramid terrain, inverted pyramid terrain, discrete random height field, continuous uniform random height field and gravel, etc. and the difficulty can be automatically modified according to the actual training situation. The simulation environment is set up as Fig. 12, a large scene with 20-by-10 grids surrounded by flat ground extended to the edge of the scene. Each grid is a terrain type of 10-by-10m square. The simulation process is like a process of breaking through and upgrading, with 4096 robots evenly divided into groups deployed randomly at the center of the most manageable level of terrain blocks. Each robot is given a command of random position and heading direction [x, y, yaw] between -0.5 and 0.5 m, -3.14 and 3.14 *rad* to learn how to obtain and maintain a feasible or excellent kinematic performance. Each training session goes through 2,000 iterations to ensure the robot has enough time to learn and adapt to the environment.

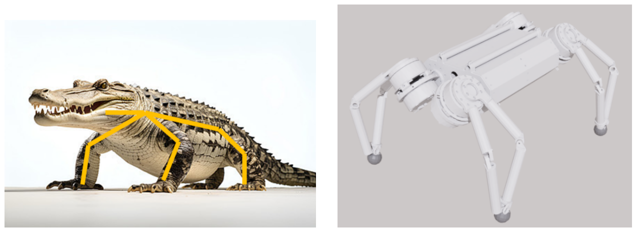
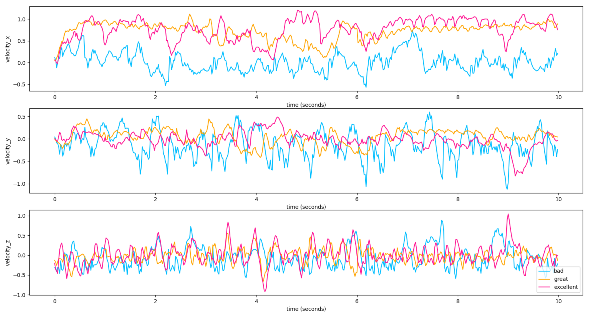
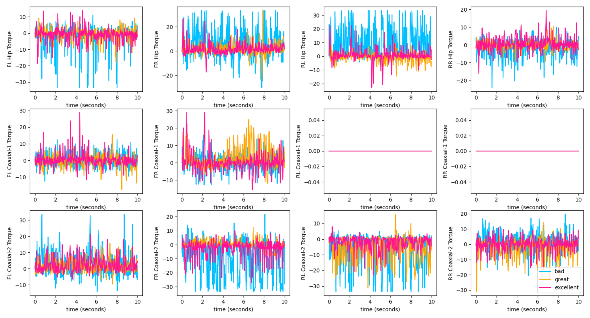


Fig. 13: **Two different versions of overconstrained quadruped**. (a) Reptile schematic; (b) Initial state of overconstrained quadruped



(a)



(b)

Fig. 14: Comparison among three training results of (a) velocity and (b) torque

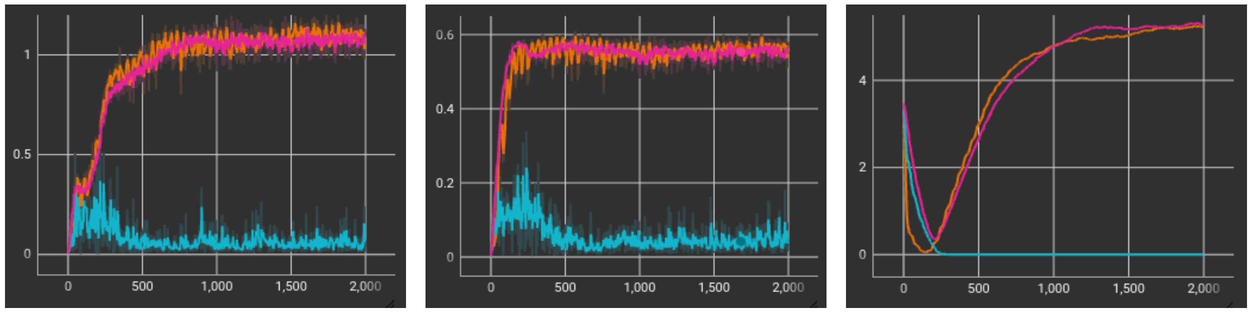


Fig. 15: Comparison of typical reward terms.

VI. Results

We begin by conducting large-scale, multi-terrain deep reinforcement learning of overconstrained locomotion using the setup illustrated in Fig. 12. According to the reward terms in Table II, where *track_lin_vel_xy_exp* and *track_ang_vel_z_exp* these two rewards are most representative of whether the control policy excellent or not, that has been trained by the control results obtained from deep reinforcement learning training. Fig. **??**(a) and Fig. **??**(b) show the velocity tracking comparison of the three training results, respectively. The three excellent, great, and bad training results are plotted as pink, orange, and blue correspondingly. It can be seen from Fig. **??**. The velocity tracking of the two excellent and great training results increases rapidly within about the first 200 steps, proving that the parameter settings are reasonable and overconstrained quadruped in a fast learning period. After 200 steps, the velocity tracking of the yaw angle (i.e., z-axis) is basically in a stable fluctuation state, indicating that the overall posture of the quadruped has been stabilized after 200 steps, and the overall ability to follow the given heading inputs, and eventually stabilized at between 0.55 and 0.56. The velocity tracking of the xy-axis also slows down the growth rate and gradually stabilizes at about 1.08 after about 800 steps. The failed training result has a small increase in the initial 200 steps, then decreases gradually, and finally falls back to a value close to 0, which proves that the training parameters are wrong and the training result is invalid.

Fig. **??**(c) is a journey map of the difficulty of the terrain that the robot can go over during the three training processes. It can be seen that the overall process of change is a rapid decline and then a slow rise, similar to a game scenario: A rookie begins to play the game and is not able to beat the level, but in the game has been continuously carried out to learn the skills, improve the level, and eventually can break through more and more levels. The excellent training result drops to a nadir at around 150 steps, and then more and more terrain can be crossed. At the same time, the great training result hits rock bottom at around 250 moves, and eventually, both training results can cross terrain levels of level 5. In contrast, the bad training result's ability to cross drops all the way down, losing the ability to cross any obstacle at around 200 steps.

Fig. 14 shows the comparisons of the velocity along each direction and the torque of the motors at each joint for the three different training results, respectively. As can be seen from Fig. 14(a), the highest speed achievable by the excellent training result in the x-axis direction is 1.23 m/s, which is faster than great and bad most of the time. In the y-axis direction, all three training results fluctuate up and down around 0 because the robot's locomotion strategy is mainly along the x-axis as seen from the visualization. Meanwhile, the excellent training result also has a higher response velocity in the z-axis, which proves that this result is more rapid in response to the environment height change compared to the other two. And Fig.14(b) shows the comparison of the output torque of the three motors, hip, coaxial-1 (to control the leg swinging back and forth), and coaxial-2 (to control the leg's changes of configuration), on the four legs of the robot ('FL' is the left front leg, 'FR' is the right front leg, 'RL' is the left rear leg, and 'RR' is the right rear leg). It can be seen that the excellent training result has basically greater output torque than the GREAT RESULT for each joint and each motor, consistent with the result of its superior kinematic performance. The torque fluctuation of the bad result is too large because the robot cannot keep standing or walking properly, which causes the system to input commands to the motors to continuously increase the torque, which has a greater impact on the damage of the motors and the stability of the robot's movement, and therefore is not considered to be a feasible result.

VII. Conclusion

Our project validates the feasibility of training overconstrained quadruped and investigating overconstrained locomotion based on reinforcement learning. What's more, the training results can implement overconstrained robots that can also go over all the challenging obstacles or terrains, including pyramid-like stairs, gravel, and changing slopes, as common quadruped robots do. It proves that the overconstrained quadruped can achieve the highest speed of $1.23m/s$ on rough environments. The current results are relatively reasonable and acceptable considering the time and resources. In the future, we will further optimize the scheme and architecture to give the robot better motion performance and more elegant motion posture so that the robot can intelligently and autonomously switch motion states when encountering more complex and challenging environments. At the same time, we will develop a general 3D-printed material-based over-constrained quadruped robot and migrate the simulation training results to complete the development of sim-to-real.

VIII. Acknowledgements

We would like to thank Prof. Chaoyang Song for teaching this project and providing guidance on the project idea, Zishang Zhang for his help in building the code framework, data extraction, the whole training process of this project, Haoran Sun for his help and guidance on the robot model preprocessing, data processingtraining settings and debug threads, Ronghan Xu for his answer on Bennett linkage and kinematics of over-constrained robots, and all the members of the project for their hard work and dedication in this semester. We would also like to thank Nvidia for inventing such an excellent simulation platform, Isaac Sim.

References

[1] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[2] W. Tan, X. Fang, W. Zhang, R. Song, T. Chen, Y. Zheng, and Y. Li, "A hierarchical framework for quadruped locomotion based on reinforcement learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 8462–8468.

[3] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2245–2252.

[4] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," in *Conference on Robot Learning*. PMLR, 2020, pp. 1–10.

[5] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[6] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[7] B. Hu, S. Shao, Z. Cao, Q. Xiao, Q. Li, and C. Ma, "Learning a faster locomotion gait for a quadruped robot with model-free deep reinforcement learning," in *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2019, pp. 1097–1102.

[8] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *Acm transactions on graphics (tog)*, vol. 36, no. 4, pp. 1–13, 2017.

[9] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, "Rapid locomotion via reinforcement learning," *The International Journal of Robotics Research*, vol. 43, no. 4, pp. 572–587, 2024.

[10] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2817–2826.

[11] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 2908–2927, 2022.

[12] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5307–5314.

[13] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.

[14] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. 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)*. IEEE, 2017, pp. 23–30.

[15] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on Robot Learning*. PMLR, 2017, pp. 262–270.

[16] J. R. Norris, *Markov chains*. Cambridge university press, 1998, no. 2.

[17] R. Bellman, "Dynamic programming," *science*, vol. 153, no. 3731, pp. 34–37, 1966.

[18] D. T. Hoang, N. V. Huynh, D. N. Nguyen, E. Hossain, and D. Niyato, *Markov Decision Process and Reinforcement Learning*, 2023, pp. 25–36.

[19] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[20] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[21] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.