Sim-to-Real: Designing Locomotion Controller for Six-Legged Robot

ChenYu Yang[1], Yue Gao*, ChangDa Tian[2] and QingShan Yao[3]

***Abstract*— Legged robots can be utilized in unstructured environments, the locomotion and motion planning of legged robots are of great importance. Six-legged robots are more challenging to control due to its “mulitple-input, multiple-output, multiple end-effectors” properties. By utilizing sim-to-real technique, we can avoid the limitation of learning controller on a real physical system. In this project, we designed a hierarchical controller by utilizing deep reinforcement learning methods to perform the tasks of navigation through obstacles and locomotion control. The high level controller navigates through obstacles and generates the footholds, and the low level controller controls each leg to move the end-effector to the designated footholds. This combination of multiple controller learning and human design is more practical than end-to-end learning method because prior knowledge can be more efficiently transferred to the learning process. The applicability of the policy network is tested in both simulation and real physical robot with promising results.**

I. INTRODUCTION

Legged robots are a kind of bio-inspired robots which utilize mechanical legs for movements [1] [2]. Compared to wheeled or tracked robots, legged robots show significant adaptability to rough terrain and flexibility to perform different tasks. Six-legged robots(hexapods) have the advantages such as high stability, high load than two or four legged robots [3] [4]. These advantages mostly come from its static stability, which is due to at least three legs supporting its body, and holding its center of mass above the triangle formed by the footholds. In addition to its static stability, hexapod’s upper platform remains stable during motion, making it an ideal choice for mounting other equipment.

Developing the control system of the robot in a simulation environment instead of real environment is a widely used approach [5] [6] [7] [8] [9]. It can make the process quicker, cheaper and safer, as the experiment accidents in real environment may potentially damage people and robots. However, the controllers that perform well in the simulation environment may not transfer easily to their real world counterparts, due to all kinds of modeling error. Therefore, engineers have to bridge this “reality gap” when transferring polices from simulation to real environment [5] [6].

Deep reinforcement learning (DeepRL) is very eligible for developing control policies in simulation environments. It allows the robot agent to learn the control policies by itself through trail and error, instead of laborious tuning by human engineers. Thank to the increasing computational power and the simulation environments where the agent can trail and error cheaply, reinforcement learning has made remarkably feats in performing challenging tasks [10] [11].

[1]Chenyu Yang. Undergraduate with the school of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University(SJTU), Dongchuan Road 800, China yangcyself@sjtu.edu.cn

[2] Changda Tian.Undergraduate with the school of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University(SJTU), Dongchuan Road 800, China deepfluency@sjtu.edu.cn

[3]QingShan Yao. Undergraduate with the school of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University(SJTU), Dongchuan Road 800, China sjtuyqs@sjtu.edu.cn

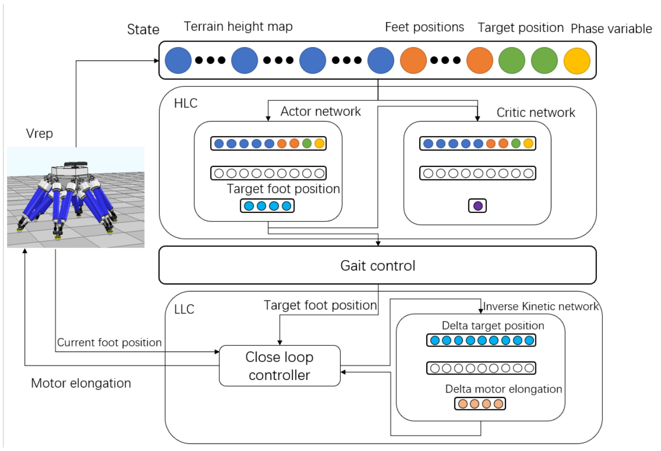

Fig. 1. The control system graph. The high level controller generate footholds positions and low level controller controls the robot’s movement to step into those footholds positions

This paper aims at training a motion and locomotion controller on a real hexapod robot. The control policies is learned in a simulation environment with DeepRL algorithms. The controller is designed to perform tasks of path planning to avoid obstacles and locomotion planning to control the robot’s posture. An overview of the controller is shown in Figure 1. The controller is a combination of neural network based modules and a traditional close loop control routine. The high level controller (HLC) is a neural network which is responsible for navigation and sending command on the low level controller (LLC). The LLC is a close loop controller which utilize a neural network to generate control signals. LLC receives the commands from HLC, and is responsible for execution of the command.

II. RELATED WORK

A. Modular Control

When solving the problem of robot control, modular control method is widely used because of its practicability and low coupling.The whole task of the robot is divided into subtasks, and each subtask will be handled by a model. These modules interact with each other by means of dynamic operations and heuristic functions. For example, the modular structure in [12] contains three modules. The first module calculates the positions of the legs at the next moment, and the second module calculates the paths that the legs move to the above positions. The last module controls the legs to follow the paths to the specified positions. Similarly, work in [13] uses a controller with two modules. One is a planning model which computes an optimal path to reach the target point, the other is a tracking model which follows the path and move the robot. As mentioned above, these modular methods are popular and easy to control has problems like the under-utilization of the robot's own state and environmental information. The higher module makes decisions based only on a simple model of the ability of the lower module. The difference between the real abilities and configuration of the lower module and that of the model believed in higher module may leads to impracticable actions for the lower module to perform.

B. *Deep Reinforcement Learning Methods*

Deep Reinforcement Learning techniques is suitable for solving decision-making optimization problems. A robot can be seen as an agent in the face of a specific situation, it should take an action plan to achieve its goal, in other word, to maximize the benefits. So, deep reinforcement learning methods suit robotics well and becomes popular in robotics [11]. The robot learns a control policy in a trail-and-error manner, observing input states, choosing actions to interact with environment, and tries to maximize the score of a reward function [14]. There are many fully-developed algorithms to tackle the problems of continuous action space, such as DDPG, TRPO and PPO. Based on actor-critic DDPG can give the next action in continuous actions [15]. TRPO can make the policy always become better [16]. PPO has higher learning efficiency [17]. These algorithms enables the policies for many tasks to be learned, such as navigating [18], dribbling a ball [10], or recover from fallen states [19] [20].

C. *Sim-to-real*

In order to transfer the policies learned from the simulation environment to the real environment, many approaches has been proposed to bridge the "reality gap". [21] [22] [23] [24] These approaches can be roughly divided into two classes. The first is to improve the simulation fidelity [21] [22]. As some characteristic are hard to model, a data-driven way, knows as system identification is usually used. The second approach is to build more robust controllers that can adapt the variations of system properties. The robust controllers can be trained by adding noise to the dynamic system [5] [6] [9], the policies and the observations. However, the perfectly accurate model is always impossible and the noise on the system may prevent the agent from learning policies that best exert the hardware abilities. As a result, this two kinds of approaches are often combined together. For example, J. Hwangbo et al. [8] developed a success quadrupedal that can run at fast speed and recover from falling down. They identify the model parameter from the actual experiments and add noise to train a robust controller. Despite the impressive results, these works fails to design robust control system from the structure's point of view. That is, the abstractions in the modular control system can be leveraged to make the policy not prune to affected by the"reality gap."

III. OVERVIEW

The control system structure can be seen as fig 1. The hexapod mainly contains a body and six legs. Each leg is driven by three motors, having a three degree of DOF. [25]. As shown in 2, for each leg, the elongation of three P motors $[l_1, l_2, l_3]$ affects position of the foot tips. In this paper, we denote BCS as the Body coordinate system, as opposite to the Global coordinate system GCS. For each leg, we use $[x, y, z]$ to denote the position in its BCS, and $[l_1, l_2, l_3]$ to represent the elongation of the motors.

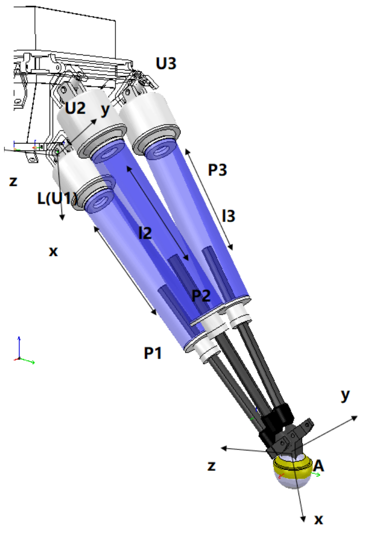

Fig. 2. Leg mechanism of our six-legged robot.

The control system is divided into two components. The high-level controller (HLC) is a neural network that decides the direction of motion and the place of footholds. The low-level controller(LLC) is a neural network that learns to put the feet to the target positions by controlling the motors of the robot, following a hand crafted gait. The HLC and LLC form a two level control hierarchy where the HLC takes in the environment information and command on the LLC. Receiving the command of HLC, the LLC makes a series of movements so that the robot change to desired posture. HLC is blocked until the LLC finishes. HLC and LLC are executed in parallel with the simulation environment. The controllers generate control signals in real time during the simulation. This hierarchical structure simplifies the task of the HLC, enables it to explore policies relevant to the task efficiently.

The input to the HLC consists of the observation,o_H, the high-level-goal g_H, and a phase variable ϕ. It outputs the position to place the end-effector in the next step a_H. The o_H contains the information of the robot's current posture, and the observation of the surrounding terrain. In this paper,

g_H is about the target which the robot should get close to. ϕ is a binary variable that represents which group of legs is the swing leg in this turn. When called by HLC, the LLC controls the robot to take a step in free-rectangular-three-gait. The three feet in one group move independently towards their intermediate target position at the same time. The feet move along a rectangular trajectory, maximizing the ability of stepping over obstacles.The actions of LLC are sent to the simulation environment, which returns the new goal and observations after executing the action. The environment calculates a reward for HLC. The HLC is trained with DDPG algorithm.

IV. HIGH LEVEL CONTROLLER

A. HLC state

The HLC input observation o_H contains the hexapod's configuration, the terrain map, the phase variable. The hexapod's configuration is described by positions of the six feet tips in the body coordinate system. As shown in figure3, the terrain map is a 2D height map of terrain sampled on the regular grid. The terrain map consists of two parts, internal and external. The internal part is fine-grained sampled to ensure the robot see small obstacles and the external part extends the field of observation and enables the robot to plan the path to bypass the larger obstacles. The phase variable is binary and indicates which group of legs are the swing legs in this step. The total observation creates a 1757D vector with 1744 samples in the terrain map and 12 feet positions and 1 phase variable.

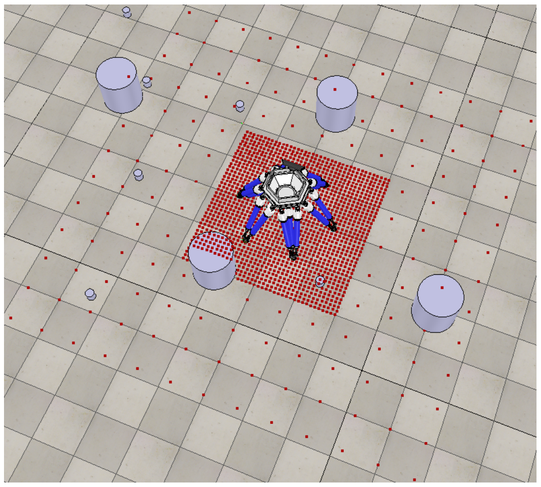

Fig. 3. The terrain features consist of a 2D height map of the terrain sampled on a regular grid. The internal part of the height map has a resolution of grid size 0.05*0.05m and occupies an area of 2*2m. The external part of the map has a resolution of 0.4*0.4m and occupies area of 4.8*4.8m

B. HLC action

The HLC action space represents all positions of the footprint in the body coordinate system. The HLC is the target position in relation to the body. The action contains three positions, representing the target positions of the three swing legs. As for the z positions of the target and the feet are all considered 0, the positions are all 2D vectors. Thus, the target is 2D vector and the action in 6D action space.

C. HLC task

The HLC is trained to navigate through some randomly placed barriers and walls to reach the goal. The obstacles are placed on a horizontal plane, some of them are short, which can be stepped over by the robot, while others are not. A target location is placed randomly in the map and is changed randomly to another place whenever the robot get within 0.5m around the target. The HLC goal $g_H = (x_{tar}, y_{tar})$ is represented by the target position in the robot's coordinate system. Since the policy is not provided with an explicit path toward the goal as input, it must learn to recognize the path from the terrain map T, deciding whether to bypass or to step over the obstacles, and plan its footsteps accordingly.

Figure4 shows a snapshot of the task. For the simplicity of implementation, we only use cylinders of different size as obstacles in toy environment for training. We use other kinds of obstacles for testing in v-rep environment.

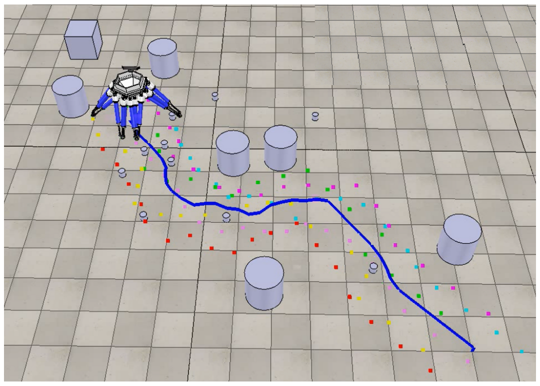

Fig. 4. The snapshot of the task. The blue line traces the trajectory of the robot and the dots of different colors represents the location where the feet made contact with the ground. The cylinders are obstacles and the cube marks the location of the target

D. HLC reward

The reward system for the agent contains two parts, the reward for approaching the target, and the penalty for falling down or collision with obstacles. For the action a in iteration t, which transform the state s into $s^{'}$, the reward can be represented as:

$$\begin{aligned} r(s,a,s',t) &= \alpha_{to} * (\max(0, \hat{d}(t) - d_{tar}(s'))) \\ &+ \alpha_{off} * \min(0, d_{tar}(s) - d_{tar}(s')) + \; r_{collid} \end{aligned} \quad (1)$$

Where d_{tar} represents the distance between the center of robot's body and the target on the horizontal plane, and $\hat{d}(t)$ is the shortest distance the agent ever get in this episode, $\hat{d}(t) = \min(\hat{d}(t-1), d_{tar}(s))$. When the robot get closer to the target, it is likely to get a positive reward, and it gets a negative reward when the distance between it and the target get larger. Note that this reward function ensures that the largest reward for the agent is bounded given a map even when the max number of iterations of one episode is not limited. In this way, the agent cannot get reward by moving back and forth. It has to keep getting closer to the target in order to earn positive rewards. α_{off} and α_{to} are two factors and are set to $\alpha_{to} = 2$, $\alpha_{off} = 1$ in our experiments.

The collide penalty r_{collid} is 0 at most times and a constant P whenever a collision with the obstacles in the environment is detected, or the robot fell down. The robot is also regarded as collide when it is in some dangerous state, for example, the legs are crossing each other or the footholds are too close. In our experiments the constant P is set to -20

E. HLC Network

The neural networks of HLC are 4-layer fully connected(FC) neural networks. Although we have height map that could be handled by Convolutional neural networks, the height map do not contain as many feature information as a picture, a simple FC network is enough. Moreover, we cannot lose the exact positions of obstacles in pooling layers. Thus, we use FC network to process the height map, the robot posture, the goal, and the phase variable together.

V. LOW LEVEL CONTROLLER

A. LLC structure

The LLC is a close loop controller which consults a neural network to generate control signals until the foot arrives at the target position. When called by HLC, the LLC calculates the difference between the target leg position and the current leg position, queries its neural network for control signal, and sends the control signal to the robot's motor, updates the new position and repeats the procedure until the difference to target leg position is below a limit.

B. LLC network

Thanks to the central symmetric structure of the robot, we can train a smaller network which takes in and out information of only one leg, and apply it to six legs. The input of the neural network is the current length of the three motors $[l_1, l_2, l_3]$ and the difference between the target leg position and the current leg position $[\Delta x, \Delta y, \Delta z]$. The output of the neural network is the difference of the length of the three motors $[\Delta l_1, \Delta l_2, \Delta l_3]$

The neural network of LLC is a simple 4 layer fully connected network with each layer has a dimension of 128. It is trained in a supervised manner with the data collected from the robot in the vrep simulation environment.

VI. EXPERIMENTS

The supplemental videos give the most detailed information about the motions of the robot. In the simulation environment, the hexapod is 0.7m to 0.85m tall and has a mass of 290kg. We use the ground friction of $\mu = 1$. The robot's motion is calculated by the inverse kinematics functions supported by v-rep and simulated with newton physics engine [26] with dt=10ms . All neural networks are built and trained with Pytorch [27]. The value of output actions of the network are in range of $[-0.1, 0.1]$.

The batch size for training the controller is 128, retrieved from experience replay memory D, which records 10k most recent tuples. The neural networks of actor are trained with Adam optimizer with learning rate=0.0001, momentum factors β_1 = 0.9, β_2 = 0.999. The networks of critic are trained with learning rate=0.001, and the momentum factors the same as actor's. The controller use a discount factor γ = 0.99. The noise for exploration added on the robot's action is a value following OrnsteinUhlenbeck process [28]with parameters(μ=0,θ = 0.15,σ=0.2). Besides the noise for exploration, the Gaussian random noise with standard deviation = 0.01 is added on the input and output of the network to make the system more robust. The controller is trained for approximately 10k episodes, requiring about one day. All computations are performed with Pytorch framework.

A. Toy environment

As the simulation in v-rep is too costly, we built a toy environment to accelerate the training process. Instead of doing kinetic simulation in the real v-rep environment, we simulate the robot motions in our toy environment, where no kinetic calculation performed. Our simulation environment is based on the following assumptions and rules:

- All the feet stepping on the ground are fixed unless it is lifted up.
- When any part of the robot collides with the other obstacle in the environment, the simulation reports a clash and stop.

Because the robot always has at least three feet stepping on the ground and the center of weight is in the triangle formed by the feet, the robot is stable at any moment. This static stability allows us to save the calculation of the momentum and therefore optimize the speed of simulation. The two assumptions above enable us to implement the toy environment. We can calculate the robot's position according to the position of its fixed feet and the relative position between body and feet. The toy environment shares the same configuration as the real v-rep environment, and have little error comparing with that with physical simulation. We train the agent with our toy-environment and then test it in the v-rep environment to see the result.

B. RESULTS

1) LLC performance: We collected the LLC performance data in the robot running episodes. The average LLC error is about 0.0403261 in the vrep environment. This error is the distance between the target position and the actual position under the control of the LLC. This precision is enough to allow the hexapod walk smoothly in the vrep environment.

2) HLC performance: Figure 5 shows the learning curve of the HLC . In the v-rep simulation experiments, the robot learns to step over low obstacles and bypass the tall ones. Note the robot does not have the global information and therefore cannot decide a global optimal plan to the target. We can see from the trajectory of the robot that the policy has some kind of arbitrariness as the movements are not directly pointing to the target. Some kind of fine planned behaviors are observed, as the robot learns to adjust its step length when approaching a low barrier, in order to make it possible to step it over.

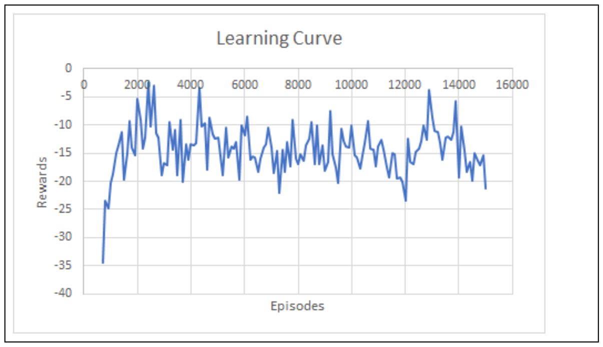

Fig. 5. The learning curve of the controller. The policy learned at about 2500 Episodes reaches its optimal. The policies generated later have no obvious difference in the way they take action.

We also designed the system to run the policy to control a real hexapod. We already have the terrain of the environment. We set on the robot an Intel RealSense camera and locate the robot via simultaneous localization and mapping (SLAM). From the prior terrain and the real-time positioning, we calculate the observed terrain and pass it to the controllers. The controllers generate signals as they do in vrep. The robot has a genuine inverse kinematic system to set the feet to the desired feet positions in relation to the body coordinate system. Figure 6 is a snap shot of the real environment experiment but the result of the real experiment can be best shown in the attached video clips. From which we can see that the the robot behave similarly as in the simulation environment, and doing properly in avoiding collision and approaching the target.

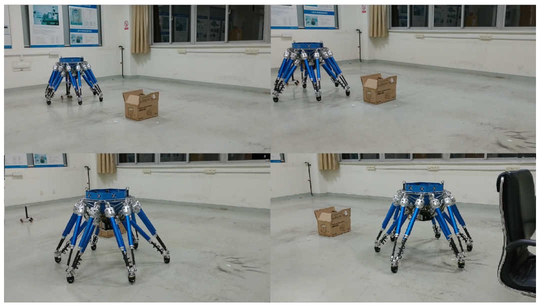

Fig. 6. A snap shot of the real environment experiment. The experiment can be seen from the attached video clips

C. Reward Function Comparisons

Work in [10] uses an reward function very different from ours, although we share some similarity in the tasks, one for controlling a bipedal toward target, and one for controlling a hexapod robot. To compare which reward function is more suitable for the problem in this paper, we modeled a reward function in light of theirs, as follows.

$$r(s,a,s',t) = \exp\left(-S^2\right) \tag{2}$$

$$S = min(0, u_{tar}^T(p(s') - p(s)) - \hat{v}) \tag{3}$$

where $p(s)$ is the planetary position of the robot's body in state s and $p(s') - p(s)$ is robot's body coordinate system displacement in that step on the horizontal plane and u_{tar}^T is a 2-D unit vector toward the target. In our experiments, $\hat{v}$ is set to 0.2, representing the desired step length by which the robot walk approach the target. The robot get penalized when it is moving slower than the desired motion.

The reward function for HLC in [10] uses speed instead of displacement. However, as our hexapod is static stable, speed is not an important factor for a HLC to consider, so we replace the speed with displacement in our adaption of reward function.

Figure 7 shows the learning curve of the agent under that reward system. This reward system is not as good as the system we used in our work. Some undesirable motions are observed as the trained agent prone to move back and forth in face of a barrier. Its intention can be explained as to avoid the danger of collide with the obstacle, which will leads to a penalty, while earn rewards in half of the actions. However, the reward system we used in previous work does not give the agent the opportunity of getting reward from this kind of behavior. The robot has to keep approaching the target in order to get the positive reward.

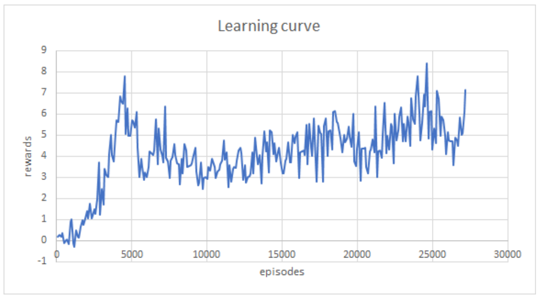

Fig. 7. The learning curve of the agent under the policy system in [10]

Figure8 shows the comparison of performance between the policy trained under the reward function in [10] and the policy trained under the reward function we used in this paper. The reward is calculated according to the adapted reward function in [10], and to make the result more meaningful, the simulation is carried out in the v-rep environment. As shown in this figure, our reward system generally yields better policy, even when the metric for comparing policies is the reward function adapted in [10]. This shows an interesting phenomenon that policies can perform better under one reward system when they are trained under another reward function instead of trained directly toward that reward function.

VII. Discussion and Conclusion

The learning of an agent with trail-and-error approach can be very costly, thus the idea of learning in a simulated

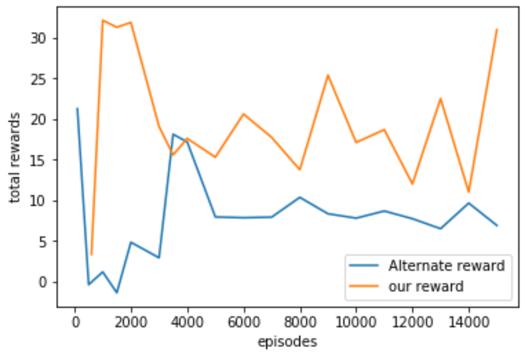

Fig. 8. The total reward of the policies trained under different reward functions. The **blue** line: the reward system we adapted in [10]. The **orange** line: the reward system we use in this paper. The experiment is carried out in vrep instead of toyrep. Because of the slowness of vrep simulation, the score of all models is get from only one run in a hand designed map. The map is much harder than that in the training environment to distinguish different models.

environment instead of a real one is often used [29]. In this work, we use the v-rep simulation environment as the toy environment of the real environment, and v-rep also has its toy environment in which the trail-and-error is performed.

The main concern for the v-rep simulation as the toy environment is the robustness of the hexapod. For the sake of the robustness, some Gaussian random noise are added to the robot's input and output. However, the main purpose for the toy environment of v-rep is the simulation efficiency, the improvement of speed exploit from the assumptions that simplify the kinetic simulation procedure. The experiments show that both toy environments derive good control policies for the environment it is simulating.

Moreover, one idea to make the agent trained in toy environment more robust and effective is to let the agent learn from episodes from both toy environment and the real environment. The weight of episodes in real environment should be higher, as they are more important and more costly to get. While we currently use only the episodes in toy environment, we intend to more fully investigate the training scheme with episodes from both toy and real environments.

Previous experiments have shown that different reward system may lead to different performance, although their goals are generally the same. Therefore, we can add some extra terms in the reward function to guide the learning of the agent. For example, we can add a term which encourage the agent in favor of some gait style. Or we can add a term which penalize the agent when it is too close to obstacle, making it more robust. While the models we used in this paper are trained with no such extra terms, we may do more research about the effects of these terms in future work.

In this paper, our low level controller only serves as the inverse kinetic calculator, which might seems unnecessary as the system itself has genuine inverse kinetic system. However, this learning-derived low level controller opens up the opportunities like self-learning and transfer-learning [30], and more complex functionalities. The low level controller can teach itself without knowing the exact configuration of the robot. It is easier to transplant a trained network to a new robot and preserve the higher levels. The low level controller with the force sensors can be helpful for hexapods on soft ground. We may further develop our low level controller in the following works.

In this paper we described a method to learn controllers by interacting with virtual environment. The controllers have hierarchical structures containing policies learned by reinforcement learning and polices developed by human. Overall, the method opens the door to learning-based approaches that enable flexible and cheap development of control policies. It allows for learning skills based on some level of human implementation, such as the low level gait controller, and in different kind of environment abstraction.

References

[1] Sangok Seok, Albert Wang, Meng Yee Chuah, David Otten, Jeffrey Lang, and Sangbae Kim. Design principles for highly efficient quadrupeds and implementation on the mit cheetah robot. In *2013 IEEE International Conference on Robotics and Automation*, pages 3307–3312. IEEE, 2013.

[2] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. Bigdog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008.

[3] Xochitl Yamile Sandoval-Castro, Mario Garcia-Murillo, Luis Alberto Perez-Resendiz, and Eduardo Castillo-Castaeda. Kinematics of hexapiderix - a six-legged robot - using screw theory. *International Journal of Advanced Robotic Systems*, 10(1):19, 2013.

[4] U. Schmucker, A. Schneider, and T. Ihme. Six-legged robot for service operations. In *Proceedings of the First Euromicro Workshop on Advanced Mobile Robots (EUROBOT '96)*, pages 135–142, Oct 1996.

[5] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. Technical report.

[6] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, Vincent Vanhoucke, Google Brain, and Google Deepmind. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. Technical report.

[7] Taeyoon Lee and Frank C Park. A Geometric Algorithm for Robust Multibody Inertial Parameter Identification. (1), 2018.

[8] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots, 2019.

[9] Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-CIO: Full-Body Dynamic Motion Planning that Transfers to Physical Humanoids. Technical report.

[10] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[11] Smruti Amarjyoti. Deep reinforcement learning for robotic manipulation-the state of the art. *arXiv preprint arXiv:1701.08878*, 2017.

[12] Marc H Raibert and John J Craig. Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 103(2):126–133, 1981.

[13] Michael Neunert, Farbod Farshidian, Alexander W Winkler, and Jonas Buchli. Trajectory optimization through contacts and automatic gait discovery for quadrupeds. *IEEE Robotics and Automation Letters*, 2(3):1502–1509, 2017.

[14] Hado Van Hasselt. Reinforcement learning in continuous state and action spaces. In *Reinforcement learning*, pages 207–251. Springer, 2012.

[15] David Balduzzi and Muhammad Ghifary. Compatible value gradients for reinforcement learning of continuous deep policies. *arXiv preprint arXiv:1509.03005*, 2015.

[16] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *Icml*, volume 37, pages 1889–1897, 2015.

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

[18] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Dynamic terrain traversal skills using reinforcement learning. *ACM Transactions on Graphics (TOG)*, 34(4):80, 2015.

[19] 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.

[20] Jun Morimoto and Kenji Doya. Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, 36(1):37–51, 2001.

[21] Duy Nguyen-Tuong, Matthias Seeger, and Jan Peters. Model Learning with Local Gaussian Process Regression. Technical report.

[22] Duy Nguyen-Tuong and Jan Peters. Learning Robot Dynamics for Computed Torque Control using Local Gaussian Processes Regression. 2008.

[23] Bongard Josh, Zykov Victor, and Lipson Hod. Resilient Machines Through Continuous Self-Modeling. 2011.

[24] M Neunert, T Boaventura, and J Buchli. Why off-the-shelf physics simulators fail in evaluating feedback controller performance-a case study for quadrupedal robots. Technical report, 2016.

[25] Y Pan. *Performance Design and Control Experiment of a Novel Hexapod Robot with PP Structure*. PhD thesis, Doctoral thesis, Shanghai Jiao Tong University, China, 2014.(in Chinese), 2014.

[26] Richard Fitzpatrick. Newtonian dynamics. 2011.

[27] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[28] Enrico Bibbona, Gianna Panfilo, and Patrizia Tavella. The ornstein–uhlenbeck process as a model of a low pass filtered white noise. *Metrologia*, 45(6):S117, 2008.

[29] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[30] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.