2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Reinforcement Learning-based Hierarchical Control for Path Following of a Salamander-like Robot

Xueyou Zhang, Xian Guo, Yongchun Fang and Wei Zhu

***Abstract*— Path following is a challenging task for legged robots. In this paper, we present a hierarchical control architecture for path following of a quadruped salamander-like robot, in which, the tracking problem is decomposed into two sub-tasks: high-level policy learning based on the framework of reinforcement learning (RL) and low-level traditional controller design. More specifically, the high-level policy is learned in a physics simulator with a low-level controller designed in advance. To improve the tracking accuracy and to eliminate static errors, a soft Actor-Critic algorithm with state integral compensation is proposed. Additionally, to enhance the generalization and transferability, a compact state representation, which only contains the information of the target path and the abstract action similar to front-back and left-right, is proposed. The proposed algorithm is trained offline in the simulation environment and tested on the self-developed real quadruped salamander-like robot for different path following tasks. Simulation and experiments results validate the satisfactory performance of the proposed method.**

I. Introduction

The locomotion control design for quadruped robots is a long-standing research topic, from basic locomotion control to high-level path planning. The traditional control method of quadruped robots is based on dynamic and kinematic models. A control framework for a quadruped robot over rough terrain is proposed in [1] and [2]. A complete control framework from high level to low level is designed by planning the foot placement [3], [4], [5]. The trajectory of the robot is constructed by planning the robot's landing point. A method of adaptively updating the foothold according to the environmental state is proposed in [6]. There is also a method based on nonlinear model predictive control for real-time motion planning of legged robots [7]. In [8], an online three-dimensional map of the environment is first established, based on which, the most suitable footholds are searched through any-time-repairing, finally, by employing the floating-base inverse dynamics, real-time planning and control for the robot is successfully realized. Unfortunately, these traditional methods involve very complicated derivation, design, and manual parameter adjustment processes.

Recently, owing to the improvement of computation power, an incredible amount of applications of reinforcement learning-based methods for robot control have been successfully achieved [9]. Noting that reinforcement learning usually does not require much human intervention, these methods are hence much more intelligent for the control of robots. Unfortunately, this kind of design faces a great difficulty of transferring a policy neural network trained in a simulation environment to a practical robot [10]. Based on this observation, much effort has been focused on designing effective methods to apply the learned policy neural network to the real world. Through system identification, an accurate model of the actuator is set up to improve the reliability of the simulation environment and then reduce the gap between the simulation and real worlds [11]. After utilizing real data to train the actuator model, and adding noise to the training robot, the simulation environment then better describes the behavior of the real robot, which finally enables the method learned from the simulation world to be successfully applied to the real one [12]. There are also policy neural networks that are trained directly on real robots [13], which surely consumes more hardware costs. These practices provide great experience for solving high-level decision-making problems of robots by using reinforcement learning(RL) and then deploying them to the real world.

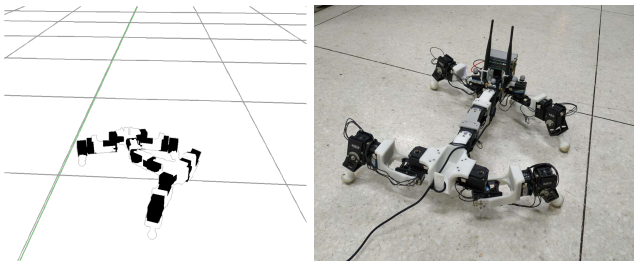

Fig. 1. The quadruped salamander-like robot. The simulation environment on the left is used to train the policy neural network, and the right one is used to verify the feasibility of the algorithm for a real robot.

Recently, the complex quadruped locomotion control problem, which requires high-level decision-making, has received considerable interest on robotics community. A hierarchical control framework is first proposed in [3], and a method of using hierarchical controller to achieve highly dynamic gait on MIT Cheetah is reported in [14]. In some latest results, reinforcement learning has been combined with hierarchical control to achieve path tracking control of quadruped robots, with the idea of extending ordinary reinforcement learning to hierarchical reinforcement learning, and training low-level network to execute high-level commands through supervised learning [15]. For instance, a hierarchical reinforcement

This research was jointly supported by the National Natural Science Foundation of China (61873132), Natural Science Foundation of Tianjin (19JCQNJC03200), Science Foundation of Science and Technology on Complex System Control and Intelligent Agent Cooperative Laboratory (192003).

Xueyou Zhang, Xian Guo, Yongchun Fang, and Wei Zhu are with the Institute of Robotics and Automatic Information System, College of Artificial Intelligence, Nankai University, China. Corresponding author: Xian Guo, e-mail: guoxian@nankai.edu.cn

learning method is proposed in [16] to realize path tracking for quadrupeds, wherein both high-level and lower-level policies are represented by neural networks. Unfortunately, both networks are end-to-end trained based on their respective rewards, which may take too much training time.

Inspired by recent results, we present a reinforcement learning-based hierarchical control framework for the self-developed salamander-like robot (see Fig. 1), for which high-level policy networks are trained with reinforcement learning to provide complex global decisions, while low-level presents the traditional controller to implement the commands issued by the high-level policy neural networks. Compared with other designs, the reported control framework takes full advantages of the characteristics for both reinforcement learning and traditional control algorithms. That is, high-level neural networks are usually suitable to process slowly-changing, high-dimensional information to make global decisions, while low-level controllers are good at tracking specific commands. In addition, for the specific task of straight paths tracking, state integral compensation is introduced into the soft Actor-Critic algorithm, which, as supported by experimental results, achieves significant improvement when compared with the ordinary soft Actor-Critic algorithm.

The contributions of the paper is summarized as follows:

1) deep reinforcement learning is successfully combined with traditional control to develop a high-performance hierarchical tracking control framework, which is directly utilized on a practical salamander-like robot;
2) both compact state space and abstract action representation are proposed to improve the generalization ability of reinforcement learning;
3) integral compensation is introduced into the state space to reduce the tracking error for specific trajectories, whose performance is verified by experimental results.

The remaining parts are organized as follows. The proposed whole hierarchical control framework is presented in Section II. Then, the design for the states, actions, rewards, together with the construction of simulation environments and the training of high-level policy neural networks, is carefully described in Section III. Next, we introduce the underlying controller, including the leg controller, the spine controller, and the coordination mechanism between the leg and the spine in Section IV. After that, the entire framework is deployed to the self-developed salamander robot to verify its feasibility and generalization capabilities in Section V. We finally conclude the paper and discuss future work in section VI.

II. Hierarchical Control Structure

The hierarchical control structure is shown in Fig. 2, where the high-level policy is a neural network trained by the soft Actor-Critic algorithm, while the low-level controller includes the spine controller and the leg controller consisting of the leg trajectory generator and the inverse kinematics solver. On one hand, high-level policy receives reward $r(s_{t-1}, a_{t-1})$ and the high-level states s_t, including robot position/orientation and target path information, from the environment. On the other hand, low-level controller receives the commands a_t, consisting of stride lengths and spine offset from the high-level policy network, and then generates and outputs the desired motor positions to the robot. Finally, we discretize the target trajectory into a series of scattered points, and based on which, the immediate reward is defined as the distance from the robot to these target points.

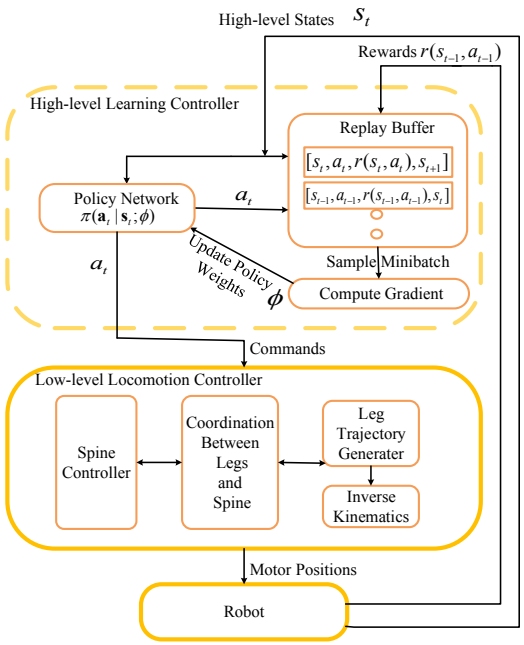

Fig. 2. Hierarchical control structure. The high-level policy receives reward $r(s_{t\ 1}, a_{t\ 1})$ and high-level states s_t, including robot position and orientation and target path information, from the environment. The low-level receives commands a_t, such as stride lengths and spine offset, from the high-level and outputs motor positions.

The hierarchical control framework, together with its states and actions, is carefully designed to increase the applicability and generality of the system. Specifically, the high-layer network needs to be trained only once, then it can be directly transferred to other new task scenarios, which can vastly help to save computation resources and training time. At first, the action generated from the high-level network is defined abstractly, such as moving forward, backward, left and right, which is effective for any path. Based on which, the low-level component then constructs the foot end trajectory and spine swing angle. Finally, the inverse kinematics is solved to obtain all motor positions which are then sent to the robot.

After obtaining the position and orientation of the robot in the world coordinate system through external sensors, the robot coordinate system is set up on the robot. Take the coordinates of the target points in the robot coordinate system as the states, which is regarded as the high-level states. That is, the distance error from the robot to the target point is used as the input of the high-level network. The smaller the error, the greater the reward. Hence, the goal of updating high-level policy neural network is to obtain optimal strategy

to minimize the error. During the path following process, the target path is first discretized into a series of scatter points, and a part of them are selected into the pre-set sliding window, whose coordinates in the robot coordinate system are sent as inputs to train the high-level network. Noting that any path can be composed of many sets of scattered points, so that the trained high-level network can be transferred to track other paths without re-training.

III. HIGH-LEVEL LEARNING CONTROLLER

A. Background

The control problem of RL in this paper can be modeled as a Markov decision process (MDP), which is defined by a tuple ($\mathcal{S}$, $\mathcal{A}$, p, r), with $\mathcal{S}$ being the continuous state space, $\mathcal{A}$ representing the continuous action space, p standing for the state transition probability, and r denoting the reward from the environment on each transition. Let $\rho_\pi(\mathbf{s}_t)$ and $\rho_\pi(\mathbf{s}_t, \mathbf{a}_t)$ denote the state and state-action marginals of the trajectory distribution included by a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$. The goal of the studied problem is to learn a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$ so as to maximize the objective of soft Actor-Critic reinforcement learning which is defined as the expectation of rewards and entropy [17]:

$$\pi^* = \arg\max_\pi \sum_t \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[r\left(\mathbf{s}_t,\mathbf{a}_t\right) + \alpha\mathcal{H}\left(\pi\left(\cdot|\mathbf{s}_t\right)\right)\right] \tag{1}$$

where α is the temperature parameter that weights the relative importance of the entropy term and the reward. The maxmium entropy objective can effectively increase the scope of exploration and speed up the convergence of RL.

B. State Space

As for the problem of path following, the target path is discretized into a series of scatter points, then a sliding window is utilized to include a succession of points, whose positions relative to the robot are selected as the high-level observations expressed as follows:

$$\mathbf{s}_t = \left\{{}^R P_1^t, {}^R P_2^t, {}^R P_3^t, ..., {}^R P_n^t\right\} \tag{2}$$

$${}^R P_i^t = ({}^R x_i^t, {}^R y_i^t), \quad i = 1, 2, 3, ..., n \tag{3}$$

with n being the number of points in the sliding window, and ${}^R P_i^t$ representing the positions of the i-th point in the coordinate system of the robot. During the path following process, the motion of the sliding window is determined by the distance from the robot to the first point in the current window. When the distance meets the pre-required conditions, the sliding window moves forward accordingly. The task of path following completes when all the points in the target path have been scanned. The measurement which drifts quickly is not considered as the part of states, so as to make it easier to apply the trained network to real robots [11].

In addition, there exists notable steady-state error during the path tracking process if one dimension of the target trajectory remains constant. For instance, if the target trajectory is selected as a straight line perpendicular to the axis of robot's coordinate system, there always exists tracking error and the robot is unable to precisely follow the target trajectory. In order to effectively eliminate the steady-state error, the integral compensation is introduced into the state as follows [18]:

$$\mathbf{s}_t' = \left\{\mathbf{s}_t, {}^R P_{com}^t\right\} \tag{4}$$

$${}^R P_{com}^t = (\sum_{k=1}^t {}^R x_1^k, \sum_{k=1}^t {}^R y_1^k) \tag{5}$$

where ${}^R P_{com}^t$ is the integral compensation representing the cumulative sum of the coordinates of the first point in the sliding window.

C. Action Space

In the hierarchical control structure shown in Fig. 2, the outputs of the high-level policy network, which are also taken as the inputs for the low-level controller, are abstract commands mainly including moving forward, backward, left and right, rather than specific ones such as the desired position/velocity for the motors, which can generally represent controls for any path, as shown in Fig. 3. In this work, the action space of high-level controller is defined by the stride length of the left and right legs, as well as the offset of the spine, explicitly expressed as follows:

$$\mathbf{a}_t = \{l_{left}, l_{right}, \varphi\} \tag{6}$$

where l_{left} and l_{right} represent the stride length of the left and right legs, respectively, and φ is the spine offset variable. When l_{left} is greater than l_{right}, the robot tends to move to the right, or vice versa.

D. Reward Function

With the goal of moving the robot to the target points covered by the sliding window, the reward function is selected as follows:

$$r = -\sum_{i=1}^n k_i[({}^R x_i^t)^2 + ({}^R y_i^t)^2] \tag{7}$$

where k_i is a weight coefficient used to judge the importance of each target point in the sliding window, and $({}^R x_i^t, {}^R y_i^t)$ representing the positions of the i-th target point in the coordinate system of the robot. As i increases, the corresponding target point becomes farther from the robot, accordingly, its weight coefficient gradually decreases. Noting that the n target points in the sliding window imply a forward looking distance to the tracking path, based on which, the robot can decide which direction to move in advance. The reward is defined as negative, so that to encourage the robot to reach the target position with least number of control steps.

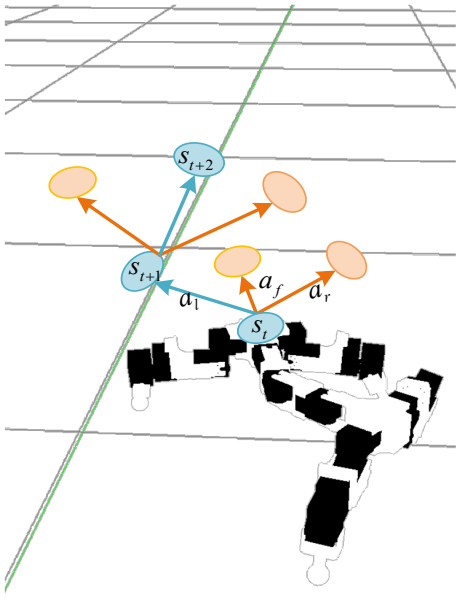

Fig. 3. The action moves the robot from one state to the next. The policy of the high-level network is to find a sequence of actions to make the robot follow the target path. The green straight line is the target path. The blue ellipse indicates the optimal state of the robot after selecting the best action, and the orange ellipse indicates the state after selecting other actions.

IV. LOW-LEVEL LOCOMOTION CONTROLLER

The low-level locomotion controller of the robot consists of leg controller, spine controller and the coordination mechanism between the legs and the spine [19]. Specifically, the kinematic model of each leg is set up, based on which, the motor positions of the joints can be calculated by the utilization of the target trajectory for each leg. Meanwhile, the motion of the spine is taken as a sinusoidal function, whose amplitude, frequency, and bias are independently designed. The movement of the spine and legs is well coordinated to keep the center of gravity of the robot within the polygon formed by the supporting legs.

Each leg has four joints, and the angle of each joint can be obtained by solving the inverse kinematics to make the end-effector follow the constructed target trajectory. Unfortunately, the pseudo-inverse method is unstable when near the singularity. Based on this observation, the inverse kinematics is obtained by solving the following optimization problem:

$$\underset{\Delta \mathbf{q}}{\text{minimize}} \|\Delta \mathbf{p} - \mathbf{J}\Delta \mathbf{q}\|^2 + \lambda \|\Delta \mathbf{q}\|^2 \tag{8}$$

where J is the Jacobian matrix of the leg, $\Delta q = q_t - q$ is defined as the difference between the target and the actual position of the joint, $\Delta p = p_t - p$ is defined as the difference between the target position of the leg end and the actual position, λ is a varying damping factor aiming to protect the motor at the singularity:

$$\lambda = \begin{cases} \lambda_0 & : \sigma_j = 0 \\ 0 & : \text{ others} \end{cases} \tag{9}$$

with λ_0 being a constant, and σ_j denoting the singular point of J. The optimization problem (8) can be solved by calculating the partial derivative of Δq and then making it equal to zero, which yields the solution as follows:

$$\Delta \mathrm{q} = \left(\mathrm{J}^T \mathrm{J} + \lambda I\right)^{-1} \mathrm{J}^T \Delta \mathrm{p} \tag{10}$$

The movement of the spine can change the position of the center of gravity, besides, it can also increase the speed of movement. Considering these facts, the controller of the spine is set as follows:

$$q_{s,i}(t) = a_i \cos\left(2\pi f t + \phi_i\right) + \varphi_i, \quad i = 1, 2, 3 \tag{11}$$

where a_i is the amplitude of the sinusoidal signal for the i-th spine joint, f is the oscillation frequency of the spine, ϕ_i is the initial phase of the i-th joint, and φ_i is the offset variable:

$$\phi_1 = 0, \phi_2 = \pi, \phi_3 = 0 \tag{12}$$

$$\varphi_i = \begin{cases} 0 & , \text{ when going straight} \\ \varphi_{0,i} & , \text{ when turning} \end{cases} \tag{13}$$

with $\varphi_{0,i}$ determined by the turning radius.

Similar to salamander crawling, the robot advances with a walking gait, for which the relative phases are given as:

$$\psi_{RF} = 0, \psi_{LH} = 0.25, \psi_{LF} = 0.5, \psi_{RH} = 0.75 \tag{14}$$

with the notation RF, LH, LF and RH standing for the right forelimb, the left hindlimb, the left forelimb, and the right hindlimb, respectively.

V. EXPERIMENTAL RESULTS AND ANALYSIS

A. End-to-end flat policy learning

As a comparison to the designed hierarchical control framework, we have also trained an end-to-end flat policy, with the results shown in video `https://youtu.be/BnAN09oxb2Y`. The inputs of the end-to-end policy network are the states of the robot, including all joint positions, robot position, robot attitude, and robot acceleration, while the outputs are the specific positions of the joint motors of the robot. As for the reward, it is defined positive when the robot moves along the desired path, while imposing negative cost whenever it deviates from the given path. At the same time, some important guiding indices are adopted into the rewarding function, such as the landing posture of the robot. The algorithm for the end-to-end flat policy training is selected as the Deep Deterministic Policy Gradient algorithm [9].

There are three drawbacks for the end-to-end flat policy. 1) It is usually difficult to transfer the controller learned in the simulation to real world. Because the flat policy of the end-to-end deep RL requires both stable movement of the robot and reliable path following, the policy network needs to utilize many states as the inputs to the robot. Besides, some important sensor data, such as speed and acceleration, are considerably different between simulation and real worlds. Due to the aforementioned reasons, it is very difficult to apply the policy network trained in simulation scenario to

real world. 2) For the end-to-end flat policy, its outputs of joint motor change dramatically. However, practical actuators cannot respond to output joint angles sufficiently fast, which brings much difficulty for algorithm implementation. 3) The end-to-end flat policy can only follow the well-trained path, implying that the generalization ability of the trained strategy network is not satisfactory.

B. *Offline traing*

The full-scale model of robot is designed in Gazebo, a robotic physics simulation engine with all kinds of sensors, where the high-level policy is trained. Both policy and value functions are represented by fully connected neural networks, with all the hyper-parameters summarized in Table I. The training process completes when it executes the permitted number of steps or the robot reaches the target position. To demonstrate the training process, all the rewards from each iteration are recorded and the obtained learning curve is shown in Fig. 4. It can be seen that after 300 iterations of training, the reward of the policy network almost reaches its maximum, indicating that the network parameters have well converged. It takes about 5 hours to execute the 300 iterations for training the robot. Also, it needs to point out that in Fig. 4, the troughs at the 1000th and 4000th iterations are the starting points for re-training when using the saved network parameters.

TABLE I

HIGH-LEVEL NETWORK HYPERPARAMETERS

Parameter	Value
discount	0.99
learning rate	2×10^{-3}
number of hidden layers	3
number of hidden units per layer	32
number of samples per minibatch	64
entropy target	-3
nonlineatity	Relu
target smoothing coefficient	0.005
replay buffer size	10^6

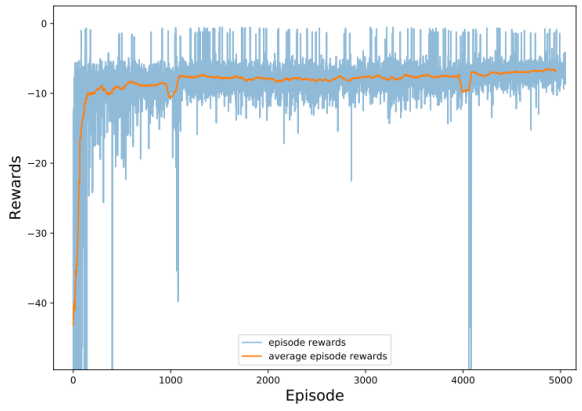

Fig. 4. The learning curve of robot following straight path.

To test the performance of the trained high-level policy network, the straight path is first followed in simulation environment. Further, considering that neural networks are usually sensitive to different state inputs, we test the scenario for three sets of path following, and all the obtained results are provided in Fig. 5, with different types of lines showing these three tests. Since the width of the robot is 0.467 m, and the maximum distance between the center of the robot and the straight path is less than 0.3 m, the robot follows well along the target path, under different state inputs. In fact, the legged robot relies on the friction between the legs and the ground to move forward, and there is a body swing during the movement. Even with these factors, the robot does not deviate from the target path due to the successful training process.

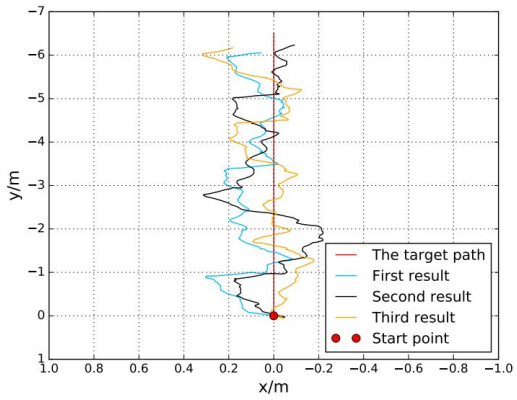

Fig. 5. The three trajectories of the robot follow a straight line in simulation environment.

To further test the generalization ability of the network, we try to transfer the high-level policy network, which have been trained on straight target path, to follow other more complex paths. To this end, a sinusoid path is utilized as the target curve to track. The parameters of the high-level policy network tracking the curve are consistent with the parameters of the high-level policy network tracking the straight line. Even thought the sinusoidal curve presents a steeper slope, and the higher-level policy network has never been trained by such a data set, it still yields satisfactory tracking performance. As clearly shown in Fig. 6, still with three different tests, the robot follows the target path successfully, and it can turn along the target path with sharp turning, which verifies that the developed hierarchical control framework presents good generalization capabilities, owing to the specific design that we consider the target path as many tiny line segments when expressing the state of the high-level policy network.

C. *Hardware Validation*

We apply the high-level policy network trained in simulation environment to the self-developed salamander robot and record its actual trajectory. The experiment environment is shown in Fig. 7, where we use a motion capture system

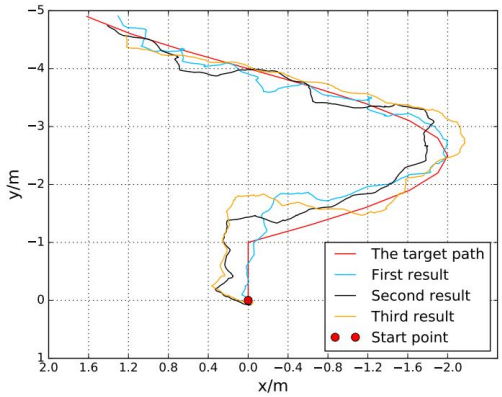

Fig. 6. The three trajectories of the robot follow a sine curve in simulation environment. The high-level policy network is trained with robots following straight paths.

(Qualysis) to measure the position and orientation of the robot. Thus, the coordinates of the target point in the robot coordinate system are obtained through the homogeneous transformation, and then used as the input for the high-level policy network. The high-level policy network runs on the PC and the low-level controller runs on TX2 carried on the robot. The action output from the high-level network is published to the underlying system through the wireless network.

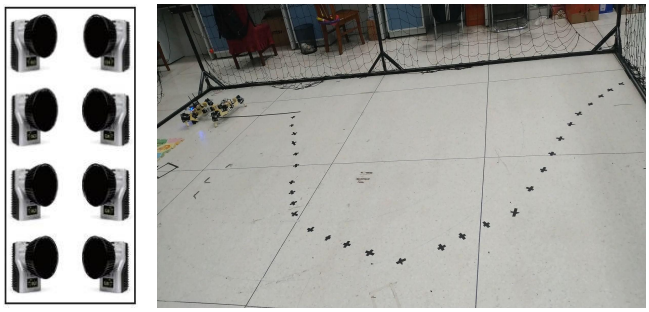

Fig. 7. Experiment environment. The robot is positioned by the external sensor motion capture system on the left. Black dots on the ground are points on the target path and are only used to show experimental results in the video.

For the first set of experiment, the designed reinforcement learning-based hierarchical control algorithm is utilized to control the salamander robot to follow a given straight path. Considering the steady-state error problem described previously, both the ordinary soft Actor-Critic (SAC) algorithm, and the soft Actor-Critic algorithm with integral compensation (SAC-IC) specifically designed for straight path, are implemented for the path tracking task. Meanwhile, a proportional-integral controller (PI) is also implemented to provide experimental comparison. For all of these three methods, the collected results are demonstrated in Fig. 8. It can be seen from the curves that all of the three algorithms can enable the robot to follow the given straight path successfully, yet the SAC-IC algorithm performs significant better than the other two ones due to the integral compensation component. Also, it should be pointed out that for the PI controller, it takes more time to adjust the control parameters, and for different paths, it is usually necessary to re-tune the control parameters which causes much inconvenience for practical applications. On the contrary, for the designed reinforcement learning-based hierarchical control method, once trained, there is no need to manually adjust parameters for different tasks, since the policy is generated by interacting with the environment. Besides, the input of the neural network can be set of sufficiently high dimension to cover numerous task scenarios, which subsequently improves the generalization ability of the reinforcement learning-based control algorithm.

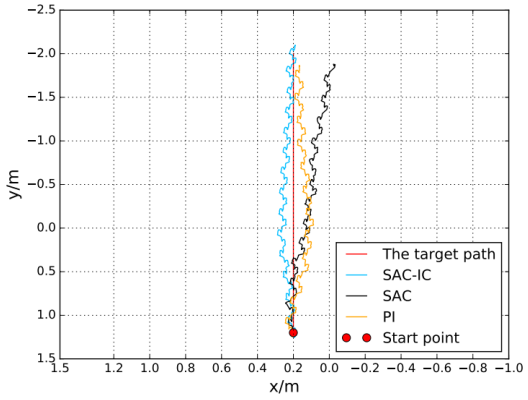

Fig. 8. The three trajectories of the robot follow a straight path in real word. The three trajectories were implemented using soft Actor-Critic (SAC) algorithm, soft Actor-Critic algorithm with integral compensation (SAC-IC), and proportional integration algorithm (PI).

For the second set of experiment, we still utilize the designed reinforcement learning-based control algorithm and the comparative PI controller to make the robot follow some sinusoidal curve. For the sake of brevity, the SAC algorithm without integral compensation is adopted as the reinforcement learning-based method. To obtain best results, the parameters for the PI controller have been re-tuned, which yields different values from the ones utilized in the first experiment. Yet, for the reinforcement learning-based method, the structure and parameters of the system are kept exactly the same with those in the first experiment, which indicates the convenience of this method. The collected results for these two control methods are presented in Fig. 9, from which it can be seen that, the trajectory generated by the learning method is apparently closer to the target path, mainly due to the reason that the neural network can input a lot of information, and the input set of target points can play a predictive role for tracking control.

Besides straight path and sinusoidal curves, the proposed reinforcement learning-based control algorithm can also be utilized to enable the robot to track other paths without re-training, which clearly indicates the feasibility of the proposed hierarchical control framework.

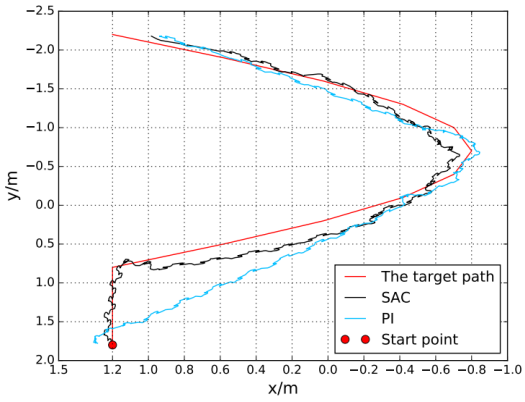

Fig. 9. The two trajectories of the robot follow a sine curve in real world. The two trajectories were implemented using soft Actor-Critic (SAC) algorithm and proportional integration (PI) algorithm.

VI. CONCLUSION

In this work, we investigate the combination of hierarchical control framework and reinforcement learning to deal with the robot control problem, where reinforcement learning is utilized to make decisions at upper level, while traditional controller are adopted at the lower level. That is, considering that the task of high-level decision-making only requires low-frequency sensor data, which is obviously more suitable for neural network learning methods, we thus develop the entire reinforcement learning process including state representation, action representation, reward design, simulation environment construction and training. Subsequently, the trained high-level policy neural network is successfully deployed to a real-world robot to fulfill different tasks. As supported by experimental results, the hierarchical control framework combined with reinforcement learning provides satisfactory performance, both for trained tasks and new ones.

Compared with other scheme, the reinforcement learning-based hierarchical control framework presents the following advantages: it does not require complex mathematical deduction or difficult design, besides, it can be conveniently applied to practical tasks and demonstrates good ability of adapting to new scenarios.

In the future, we will try to apply this control framework to more complex tasks, such as control a robot to move on rough terrain full of various dynamic obstacles.

REFERENCES

[1] John R Rebula, Peter D Neuhaus, Brian V Bonnlander, Matthew J Johnson, and Jerry E Pratt. A controller for the littledog quadruped walking on rough terrain. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1467–1473. IEEE, 2007.

[2] J Zico Kolter, Mike P Rodgers, and Andrew Y Ng. A control architecture for quadruped locomotion over rough terrain. In *2008 IEEE International Conference on Robotics and Automation*, pages 811–818. IEEE, 2008.

[3] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. Learning, planning, and control for quadruped locomotion over challenging terrain. *The International Journal of Robotics Research*, 30(2):236–258, 2011.

[4] Matt Zucker, Nathan Ratliff, Martin Stolle, Joel Chestnutt, J Andrew Bagnell, Christopher G Atkeson, and James Kuffner. Optimization and learning for rough terrain legged locomotion. *The International Journal of Robotics Research*, 30(2):175–191, 2011.

[5] J Zico Kolter, Pieter Abbeel, and Andrew Y Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. In *Advances in Neural Information Processing Systems*, pages 769–776, 2008.

[6] Alexander Winkler, Ioannis Havoutis, Stephane Bazeille, Jesus Ortiz, Michele Focchi, Rüdiger Dillmann, Darwin Caldwell, and Claudio Semini. Path planning with force-based foothold adaptation and virtual model control for torque controlled quadruped robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6476–6482. IEEE, 2014.

[7] Farbod Farshidian, Edo Jelavic, Asutosh Satapathy, Markus Giftthaler, and Jonas Buchli. Real-time motion planning of legged robots: A model predictive control approach. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 577–584. IEEE, 2017.

[8] Alexander W Winkler, Carlos Mastalli, Ioannis Havoutis, Michele Focchi, Darwin G Caldwell, and Claudio Semini. Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5148–5154. IEEE, 2015.

[9] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[11] 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. In *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018.

[12] 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.

[13] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[14] Dong Jin Hyun, Sangok Seok, Jongwoo Lee, and Sangbae Kim. High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the mit cheetah. *The International Journal of Robotics Research*, 33(11):1417–1445, 2014.

[15] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.

[16] Deepali Jain, Atil Iscen, and Ken Caluwaerts. Hierarchical reinforcement learning for quadruped locomotion. *arXiv preprint arXiv:1905.08926*, 2019.

[17] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[18] Yuanda Wang, Jia Sun, Haibo He, and Changyin Sun. Deterministic policy gradient with integral compensator for robust quadrotor control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[19] Xueyou Zhang, Yongchun Fang, Wei Zhu, and Xian Guo. A novel locomotion controller based on coordination between leg and spine for a quadruped salamander-like robot. In *2019 12th International Workshop on Robot Motion and Control (RoMoCo)*, pages 68–73. IEEE, 2019.