Abstract—With the rise of computing power, using data-driven approaches for co-designing robots' morphology and controller has become a feasible way. Nevertheless, evaluating the fitness of the controller under each morphology is time-consuming. As a pioneering data-driven method, Co-adaptation utilizes a double-network mechanism with the aim of learning a Q function conditioned on morphology parameters to replace the traditional evaluation of a diverse set of candidates, thereby speeding up optimization. In this paper, we find that Co-adaptation ignores the existence of exploration error during training and state-action distribution shift during parameter transmitting, which hurt the performance. We propose the framework of the concurrent network that couples online and offline RL methods. By leveraging the behavior cloning term flexibly, we mitigate the impact of the above issues on the results. Simulation and physical experiments are performed to demonstrate that our proposed method outperforms baseline algorithms, which illustrates that the proposed method is an effective way of discovering the optimal combination of morphology and controller.

I. INTRODUCTION

A robot's performance relies on its mechanical structure as well as its proficiency in control, which are inherently coupled. Heavy human engineering is required to achieve the optimal combination of mechanical structure and controller previously. With the increased computing power, many data-driven algorithms [1]–[5] have emerged to address the co-design of robot's morphology and controller problems.

Among data-driven approaches, most of them [1]–[4] reply on a population of morphology candidates whose policies are necessary to be trained in the simulator to calculate fitness. Then the new morphology can be selected based on fitness. Such methods are time-consuming. Improving the efficiency of the optimizer is worthwhile. As a pioneering method, Co-adaptation [5] learns a Q-function conditioned on morphology parameters to evaluate the fitness of the candidates, thereby avoiding the need to train each candidate from scratch. Specifically, it uses two identical networks: individual network and population network with Actor-Critic architecture [6]. The former interacts with the environment under specific morphology, and the latter integrates the interactive data under different morphologies to provide the surrogate function for morphology optimization. The introduction of the double-network mechanism significantly improves data efficiency.

In this paper, we find that since the population network in Co-adaptation [5] is updated without direct interaction with the environment, exploration errors emerge in such an offline setting. It affects the estimation of fitness in the process of morphology optimization severely. Besides, Co-adaptation [5] utilizes the population network’s parameters to initialize the individual network to speed up training. Such procedures result in performance collapses due to sudden state-action distribution shifts. To address the above mentioned issues, we propose the concurrent networks that couple online and offline RL methods [7]. Specifically, a policy-constraint method [8] is utilized to train the population network offline for alleviating the exploration error, and an adaptive behavior cloning term [9] is used to train the individual network online for mitigating the state-action distribution shift during parameter transmitting. Furthermore, we perform two simulation tasks and one physical task to verify the effectiveness of the improvements. To summarize, our contributions are as follows:

1) We refer to the double-network mechanism proposed by Co-adaptation [5] to solve the co-design problem, and find there exist exploration errors and state-action distribution shifts problems, which are fatal to the performance. So we propose concurrent networks that couple online and offline RL methods, which is simple but effective to alleviate the impact of the aforementioned issues. Through the proposed method, the offline network can better use the online data for learning general policies, and the online network can better use the offline network’s parameters to speed up training.

2) We evaluate two typical legged robot locomotion tasks in simulation. Both tasks verify that the proposed method can greatly mitigate the problem in Co-adaptation [5] and improve the optimization performance.

3) A four-degree-of-freedom legged robot is constructed to perform validation on hardware. Through the simulation, it is found that the ratio of front and rear leg length and gait of the robot are significantly related to the locomotion speed, we have verified the two findings via a physical experiment.

II. RELATED WORKS

A. Co-design for robots

The joint optimization of morphology and policy can be divided into two categories: robot topological configuration
changing and unchanging.

In the first category, the co-design problem can be formulated as a bi-level optimization problem or formulated as a joint optimization problem, where morphology and control are optimized simultaneously. For the bi-level optimization methods, [1] maintains a distribution over designs and uses the reinforcement learning algorithm to optimize a control policy to maximize the expected reward over the design distribution. Co-adaptation [5] can also be classified into this category, by leveraging a Q function conditioned on morphology parameters, it eliminates the need to train a population of candidates. Its specific implementation is described in Section III-B. For the joint optimization approaches, early notable works mainly based on evolutionary approaches, such as [10]–[12]. Among recent works, [13] proposed a computational approach to perform co-design approaches, such as [10]–[12]. Among recent works, [13] expressed hard-泛化 difficult design distributions. Co-adaptation [5] can also be classified as a control policy to maximize the expected reward over the design distribution. Co-adaptation [5] is to represent the model prediction. The objective of the upper-level is to make the learned policy only access data similar to the robot and the environment can interact. The evaluation is depicted as follows:

$$J(\pi, \xi) = \mathbb{E} \left[ \sum_{t=0}^{H} \gamma^t r(s_t, a_t, \xi) | a_t \sim \pi(\cdot | s_t, \xi) \right]$$  (2)

where $\gamma$ is the control policy, $\xi$ is the morphology design parameters, $F(\cdot)$ and $J(\cdot)$ are objective functions of the upper and lower layers, respectively. The lower-level optimization is performed firstly to get the optimized controllers under the pre-defined morphology parameters, and then the upper-level optimization is performed to obtain the optimized morphology among the morphology parameters search space based on the fitness acquired by the optimized controllers.

The interaction between the robot and the environment can be modeled as an extension of the Markov Decision Process (MDP) conditioned by morphology parameters $\xi$, which can be represented by $(S, A, P, R, \gamma)$, where $S$ and $A$ represent state and action space, $P$ and $R$ denote dynamics and reward function, and $\gamma \in [0, 1)$ indicates the discount factor. At each time step $t$, an agent selects an action $a_t \in A$ under state $s_t \in S$ according to policy $\pi(\cdot | s_t, \xi)$ and receives a reward $r_t = r(s_t, a_t, \xi)$. The environment transfer to a new state $s_{t+1}$ following the transition model $p(s_{t+1} | s_t, a_t, \xi)$.

The objective of lower-level optimization is to optimize the control policy to maximize the expectation of the accumulative rewards conditioned on a specific morphology parameter $\xi$. The evaluation is depicted as follows:

$$F(\pi^*(\xi), \xi) = \mathbb{E} \left[ \sum_{t=0}^{H} \gamma^t r(s_t, a_t, \xi) | a_t \sim \pi^*(\cdot | s_t, \xi) \right]$$  (3)

In (3), $\pi$ is fixed and $\xi$ is parameter to be optimized.
B. Co-adaptation

Co-adaptation [5] claims its main difference from other methods is that it does not require simulators to calculate the fitness of the agent, but a trained Q function conditioned on morphology parameters serves as the objective function for the morphology optimization process. Specifically, two identical networks are utilized to establish the double-network mechanism, which is called individual network and population network, respectively. The individual network interacts with the environment, and the interaction tuples \((s_t, a_t, s_{t+1}, r_t | \xi)\) are stored in \(D_{\text{ind}}\) and \(D_{\text{pop}}\). During each iteration of morphology optimization, \(D_{\text{ind}}\) is cleared, and \(D_{\text{pop}}\) stores all the interaction tuples under different morphologies.

In the training process, most of the training data of individual network come from \(D_{\text{ind}}\), and that of population network all come from \(D_{\text{pop}}\). Since \(D_{\text{pop}}\) stores interaction tuples under different morphology design parameters, the population network can better generalize across different morphology designs. When facing new morphology designs, the population network is expected to provide an estimation of Q-values as the surrogate function for morphology optimization. Specifically, the Q-value of the initial state \(s_0\) is utilized. Therefore, the objective of morphology optimization (3) can be reformulated as:

\[
\xi^* \approx \arg \max_{\xi} \mathbb{E}[Q_{\text{pop}}(s_0, a_0, \xi)|a_0 \sim \pi_{\text{pop}}(\cdot|s_0, \xi)] \tag{4}
\]

In this way, Co-adaptation [5] transforms the problem of finding the optimal morphological parameters into training networks that can predict the Q-values of different morphology for a given state and action. In practice, both the individual network and population network are trained by SAC algorithm [34], and the morphology is optimized by the particle swarm optimization (PSO) method.

IV. METHOD

A. Policy-constraint method for offline RL

In Co-adaptation [5], when training the population network, it’s necessary to perform value estimation as shown below:

\[
Q_{\text{pop}}(s_t, a_t, \xi) \leftarrow r_t + \gamma Q_{\text{pop}}(s_{t+1}, a_{t+1}, \xi) \tag{5}
\]

The policy function \(\pi_{\text{pop}}\) obtains the virtual next action \(a_{t+1}\) according to the next state \(s_{t+1}\) and morphology parameters \(\xi\), which come from interaction tuple collected by the individual network. Then \(Q_{\text{pop}}(s_{t+1}, a_{t+1}, \xi)\) can be acquired by Q function \(Q_{\text{pop}}\), and the target Q-value \(Q_{\text{pop}}(s_t, a_t, \xi)\) can be calculated. If population network interacts with the environment itself, when the Q function \(Q_{\text{pop}}\) overestimates the state-action pair, the policy \(\pi_{\text{pop}}\) may collect data in the uncertainty region, and the erroneous value estimate can be corrected. Nevertheless, since the population network is updated by the data from the static dataset, \(a_{t+1}\) selected by policy function \(\pi_{\text{pop}}\) may be sub-optimal, and the distribution of \((s_{t+1}, a_{t+1}, \xi)\) may be quite different from that of the replay buffer, resulting in an incorrect estimation of the target Q-value and the failure of the Q-learning based algorithm SAC [34]. This process is called exploration error. Thus, the Q function \(Q_{\text{pop}}\) can not provide a well surrogate function for morphology optimization, which is fatal for the Co-adaptation [5] algorithm.

We utilize a policy-constraint method (TD3-BC) [8] to handle the exploration error problem. Although simple, it can still achieve or exceed other complex state-of-the-art offline RL methods [29], [30]. Specifically, when calculating the Actor’s loss function of population network, we add the behavior cloning term to promote the actions obtained by the policy to approach the action in the dataset, thereby reducing the error estimation of the Q-value. The update formulate is
as follows (In practical implementation, we concatenate $\xi$ and state $s_t$ together. For notational clarity, we conceal $\xi$ in the remainder of the paper):

$$J_\pi(\phi') = -E_{(s_t,a_t) \sim D_{ind}} \left[ \frac{Q_\theta(s_t, \pi_\phi(s_t))}{\sum (s_{i}, a_{i}) [Q_\theta(s_i, a_i)]} - \alpha (\pi_\phi'(s_t) - a_t)^2 \right]$$  \hspace{1cm} (6)

where $\theta'$ and $\phi'$ represent the network parameters of Critic and Actor of the population network, respectively. To balance the values of the two terms in (6), the Q-value is normalized, and $\alpha$ is added to control the weights of the behavior cloning term, we use $\alpha = 0.4$ in our experiments.

B. Adaptive behavior cloning term for offline-to-online

To speed up training, Co-adaption [5] duplicates the population network’s parameters to initialize the individual network in each morphology optimization iteration. Since the population network is trained offline and individual networks are trained online, this can be considered an offline-to-online problem [9], [25], [35]. During morphology optimization, the individual network is certain to encounter an unfamiliar state-action regime as the morphology parameters are changed. Hence we deem it can fit the Q function when meeting a new morphology configuration.

IV-A. [9] points out that adaptively adjusting the weight of the behavior cloning term can prevent severe performance drop in the initial stage and accelerate the online training speed later. Inspired by this, we also add the behavior cloning term to the Actor’s loss function of individual network. The difference is that the weight of the behavior cloning term in the population network is fixed, while that of the individual network can be adjusted automatically as the training process.

Adaptively choosing the weighting is difficult, and requires a hand-crafted heuristic. In this work, based on the idea of [9], the control mechanism similar to the proportional-derivative (PD) controller is utilized. To separate from that of the population network, we set the weights of the behavior cloning term of the individual network to be $\beta$, and the loss function is shown below:

$$J_\pi(\phi) = -E_{(s_t,a_t) \sim D_{ind}} \left[ \frac{Q_\theta(s_t, \pi_\phi(s_t))}{\sum (s_{i}, a_{i}) [Q_\theta(s_i, a_i)]} - \beta (\pi_\phi(s_t) - a_t)^2 \right]$$  \hspace{1cm} (7)

Specifically, $\beta$ is composed of two components, the proportional term is determined by the difference between the current return $R_{current}$ and target return $R_{target}$, and the derivative term is determined by the changes in returns between the current episode $R_{current}$ and last episode $R_{last}$.

The formula is as follows:

$$\Delta \beta = K_p (R_{current} - R_{target}) + K_d \cdot \max(0, R_{last} - R_{current})$$  \hspace{1cm} (8)

where $K_p$ and $K_d$ are weights of two terms, $R_{target}$ is a hyperparameter that needs to be set manually according to different tasks. For each morphology configuration, we train hundreds of episodes. In each episode, the agent first interacts with the environment for pre-defined steps and calculates the cumulative rewards $R_{current}$ in this episode meanwhile. Then $R_{last}$ is updated with $R_{current}$ of the previous episode. Next we can get $\Delta \beta$ according to E.q.(8), then $\beta$ in E.q.(7) can be updated. $\beta$ is constant within each episode and changes as the episode progress.

C. Bayesian optimization for morphology selection

Among the concurrent networks, the population network is trained to synthesize data from different morphologies. Hence we deem it can fit the Q function when meeting a new morphology configuration.

$$F(\pi^*(\xi), \xi) \approx F(\pi_{pop}, \xi) = E[Q_{pop}(s_0, a_0), \pi_{pop} | \Psi]$$  \hspace{1cm} (9)

By utilizing (9), we approximate a model $M : \xi \mapsto F(\pi_{pop}, \xi)$, which is usually modeled with Gaussian process. The learned model $M$ is utilized to calculate the acquisition function $\psi_i(\xi)$, specifically, Gaussian Process Upper Confidence Bound (GP-UCB) [36] is adopted in our method. In each round of optimization, the optimization results are as follows:

$$b_i = \arg \max_{\xi \in \Xi} E[Q_{pop}(s_0, a_0), \pi_{pop} | \Psi]$$  \hspace{1cm} (10)

where $\mu_i(\xi)$ and $\sigma_i^2(\xi)$ are mean and variance of model $M$ respectively, $\kappa$ is a hyperparameter, which is used to controls the trade-off between exploration and exploitation.

In this section, we design several experiments to answer the following questions:

1) Does the policy-constraint method alleviate the exploration error caused by the population network not interacting with the environment?
2) Does the adaptive behavior cloning term mitigate the performance collapse caused by the sudden state-action distribution shift when using the population network’s parameters to initialize the individual network?

3) Does the proposed method significantly improve the optimization performance compared to the Co-adaptation [5] method?

4) Does the rule found in the simulator by the proposed method still hold in the physical experiments?

A. Legged robot tasks in Gym

Setup: We consider two typical legged robot tasks based on MuJoCo, wrapped in OpenAI Gym API: HalfCheetah and Ant. The former is a 2D motion task, while the latter is a 3D motion task. We modify the length of the robots’ legs by changing the XML file based on the xmltodict library. The training is performed on an NVIDIA Geforce GTX 2080ti GPU. A detailed description of the environment and the hyperparameters of the algorithm can be found in the Appendix [A][B].

Policy-constraint method analysis: In this experiment, we manually set four groups of morphological parameters (HalfCheetah 1-4, Ant 1-4 in Fig.2). The population network adopts three different implementations, the first is the proposed method (Population-TD3BC), and the second method BCQ (Population-BCQ) is another offline RL method [20], which adopts a generative model to generate actions that are expected within the distribution range of actions in the replay buffer. The third only uses the TD3 method (Population-TD3), which is without offline settings. The rewards acquired by individual network and three kinds of population network are shown in Fig.3. The training starts with the first group of morphology parameters and ends with the fourth group. For the sake of fairness, the individual network is trained by the TD3 algorithm [37], and the individual network’s parameters in each group are not initialized by the population network anymore, but continue to be trained based on that of the previous group.

For HalfCheetah-1 and Ant-1, the individual network and population network use the same data for training (because only the data under the first group of morphological parameters are utilized at the beginning). It can be found that the individual network has the highest rewards, indicating that the exploration error problem does exist. By comparing the following groups of data, we find that as the training progress, the rewards of Population-TD3BC gradually approach the rewards of the individual network and even surpass the individual network in the fourth iteration. It can be found that the rewards of Population-BCQ are lower than that of Population-TD3. We speculate that the introduction of additional generative networks may cause the training slowdown. The above experiments demonstrate that the proposed method can greatly reduce the exploration error problem caused by the offline setting so that the population network can provide a more reliable estimation for the upper-layer morphology optimization.

Adaptive behavior cloning term analysis: We conduct four comparative experiments to answer the second question. The first is to initialize the individual network with random parameters (No Copy), the second is to directly transfer the population network’s parameters to the individual network (Direct Copy), and the third is to fix $\beta$ in (7), to reduce the performance drop caused by the distribution shift (Fixed Term), the last one is the proposed method (Adaptive Term). The training starts from the first group of morphology parameters and ends with the fifth group in Fig.2. The results of the second group to the fifth group are plotted in Fig.3 (since there are no parameters transmitting in the first group, so we don’t plot it).

From Fig.4, we find that the initial rewards of the four methods are similar in the second group. From the third group, it can be found that among the initial rewards, No Copy has the lowest rewards. Due to the distribution shift, the Direct Copy’s initial rewards are neither high. The initial rewards of Adaptive Term and Fixed Term are relatively high, proving that these two methods can alleviate the performance drop caused by the distribution shift. However, as the fixed behavior cloning term limits the exploration of the agents, the rewards at the end epoch are not as high as the proposed method. This set of experiments shows that the proposed method can alleviate the initial performance drop as well
as make the agent keep a high degree of exploration to keep the rewards of the end higher than other methods. It is worth noting that although the parameters of Direct Copy, Fixed Term, and Adaptive Term are copied from the same population network, the curves in Fig 4 are obtained from the evaluating stage (after the training stage), and the initial network parameters are changed after the training stage, so the initial rewards may not have the same values, which can also be seen in other offline-to-online works [9], [35].

**Morphology optimization results:** We compare the morphology optimization performance of the proposed method with four baselines. The cumulative rewards under optimized morphology with the corresponding controller are shown in Tab.I (# represent iterations). The first baseline is Coadapt.SP, which is the original implementation of Coadaptation [5]. The second baseline is Coadapt.TP, which replaces the original SAC algorithm with the TD3 algorithm (By adjusting the hyperparameters, we want to ensure the results of Coadapt.SP can be reproduced). The third baseline is Coadapt.TB, which replaces PSO in Coadapt.TP by Bayesian optimization. The fourth baseline is Random Sampling, which sample designs uniformly at random with the parameter ranges, which can be considered as the lower bound of morphology optimization.

We perform independent T-tests between Coadapt.SP and Coadapt.TP among three tasks, and all the p-values are higher than the threshold 0.05, which proves that by selecting hyperparameters, we guarantee the results are independent of the algorithm selection. Furthermore, we perform independent T-tests between Coadapt.TP and Coadapt.TB among three tasks, the p-values of the Four-legged robot and Halfcheetah are less than 0.05, and that of Ant is greater than 0.05. And all the mean values of Coadapt.TB are greater than Coadapt.TP. Therefore, we can conclude that Bayesian optimization is more suitable for our task most time. The p-values between Coadapt.TB and the Proposed method are all less than the threshold, which proves that the introduction of flexible behavior cloning term is indeed effective in our task. Besides, the rewards of the proposed method have a relatively steady upward trend as the optimization process, which also shows the effectiveness of the improvements compared to other baselines. As the lower bound of optimization, the Random Sampling method has the lowest rewards, which is under our expectations.

**B. Legged robot task in real world**

In this section, we consider a four-legged robot to verify that the proposed method is still applicable in the physical world. For the lower level, we combine RL with Central Pattern Generator (CPG) [38], [39], and define the action of RL as the phase difference of CPG to train gaits. A simulation model is built firstly that is the same as the physical robot, then the optimization algorithm is performed in the simulator. By analyzing the results obtained in the simulation, we find that the ratio of front and rear leg length and gait of the robot are significantly related to locomotion speed. Further, we prove these two factors in the physical experiment. More details can be seen in Appendix.

The optimization results are shown in Tab.II which exhibits similar performance as that of HalfCheetah and Ant tasks. To verify the effectiveness of morphology optimization results, we select two optimized morphology parameters: front leg length and rear leg length as the x-axis and y-axis, respectively, and plot them in Fig 5(b). By analyzing it, we find that all optimized points remain at the upper left corner of the line $y = x$, predicting that the robots run faster when the rear legs are longer than the front legs. For the gait optimization results, we compare the distances traveled by the trained gait and the three classical gaits simultaneously under the same morphology and record the results in Fig 5(c). Among all gaits, the trained gait achieves the furthest distance. Furthermore, we find that under the flat ground, when the RL algorithm converges, the CPG parameters tend to be static, so there is no need to obtain states during physical deployment and only needs the optimized CPG parameters to generate motion instructions.

Fig. 5(a) shows the physical experimental results. To show the results more clearly, we plot the distances that the robot moves as a line graph in Fig 5(d). For the classic gait, we select the trot gait with the fastest movement speed for comparison. From the second and third row of Fig 5(a), it can be found that under the trot gait, the robot moves...
faster when the front legs are shorter than the rear legs (the intersection in Fig.4(d) is due to ground slip), which verifies the results of the morphological optimization in the simulation experiments. By comparing the results in the first and the second row in Fig.[5(a)], it can be found that under the same morphological parameters, the trained gait is faster than the trot gait, thus validating the results of policy optimization.

VI. CONCLUSIONS

In this paper, we refer to the double-network architecture proposed by [5] to solve the robot co-design problem under unchangeable topologies. Aim at exploration error and state-action domain shift problems existing in [5], based on the policy-constraint method [8], we propose a simple but effective method to solve them. Furthermore, we also verify the effectiveness of the optimized results in a physical environment, the limitation of our method is that we are able to optimize the morphology and control optimization results obtained in the simulator are still effective, which are still very effective on the real robot with a simple control system. In the future work, we will continue to optimize the physical robot and install some sensors to form a closed-loop control system to adapt to the changing environment, such as locomotion in the presence of uneven terrain, obstacle, variations in friction, etc.

REFERENCES

[1] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter, “Jointly learning to construct and control agents using deep reinforcement learning,” in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 7908–7915.
[2] T. Wang, Y. Zhou, S. Filder, and J. Ba, “Neural graph evolution: Towards efficient automatic robot design,” arXiv preprint arXiv:1906.05370, 2019.
[3] D. J. Hejna III, P. Abbeel, and L. Pinto, “Task-agnostic morphology optimization,” arXiv preprint arXiv:2102.13100, 2021.
[4] A. Gupta, S. Saremi, S. Ganguli, and L. Fei-Fei, “Embodied intelligence via learning and evolution,” Nature communications, vol. 12, no. 1, pp. 1–12, 2021.
[5] K. S. Luck, H. B. Amor, and R. Calandra, “Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning,” in Conference on Robot Learning. PMLR, 2020, pp. 854–869.
[6] V. Konda and J. Tsitsiklis, “Actor-critic algorithms,” Advances in neural information processing systems, vol. 12, 1999.
[7] J. Schrittwieser, T. Hubert, A. Mandhane, M. Barekatain, I. Antonoglou, and D. Silver, “Human-level performance in the Atari learning environment using deep reinforcement learning,” in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 9798–9805.
[8] S. Fujimoto and S. S. Gu, “A minimalist approach to offline reinforcement learning,” in 2021 International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 6259–6266.
[9] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and D. Rus, “Robogrammar: graph grammar for terrain-adapted robot design,” in Proceedings of the 21st annual conference on Computer graphics and interactive techniques, 1994, pp. 15–22.
[10] H. Lipson and J. B. Pollack, “Automatic design and manufacture of robotic lifeforms,” Nature, vol. 406, no. 6799, pp. 974–978, 2000.
[11] S. Ha, S. Coros, A. Alspach, J. Kim, and K. Yamane, “Joint optimization of robot design and motion parameters using the implicit function theorem.” in Robotics: Science and systems, vol. 8, 2017.
[12] T. Chen, Z. He, and M. Ciocarlie, “Hardware as policy: Mechanical and computational optimization using deep reinforcement learning,” arXiv preprint arXiv:2008.04460, 2020.
[13] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, “Chainqueen: A real-time differentiable physical simulator for soft robotics,” in 2019 International conference on robotics and automation (ICRA). IEEE, 2019, pp. 6263–6271.
[14] P. Ma, T. Du, J. Z. Zhang, K. Wu, A. Spielberg, R. K. Katzschmann, and W. Matusik, “Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation,” ACM Transactions on Graphics (TOG), vol. 40, no. 4, pp. 1–14, 2021.
[15] J. Xu, T. Chen, L. Zlokapa, M. Foshey, W. Matusik, S. Sueda, and P. Agrawal, “An end-to-end differentiable framework for contact-aware robot design,” arXiv preprint arXiv:2107.07501, 2021.
[16] A. Zhao, J. Xu, M. Konakovik-Lukovic, J. Hughes, A. Spielberg, D. Rus, and W. Matusik, “Robogrammar: graph grammar for terrain-adapted robot design,” arXiv preprint arXiv:2008.04460, 2020.
[17] S. Fujimoto, D. Meger, and D. Precup, “Off-policy deep reinforcement learning over behavioral cloning?” in 2021 International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 9863–9869.
[18] A. Kumar, J. Hong, A. Singh, and S. Levine, “When should we prefer offline reinforcement learning over behavioral cloning?” arXiv preprint arXiv:2204.05618, 2022.
Fig. 5. Physical experiment results. (a) Screenshots at consecutive time intervals of physical robots. The first row is the trained gait when the front legs are shorter than the rear legs, the second row is the trot gait when the front legs are shorter than the rear legs, and the third row is the trot gait when the front legs are longer than rear legs. (b) The morphology optimization result, the $x$-axis is the length of the front legs, and the $y$-axis is the length of the rear legs. The color of points represent the iterations of morphology optimization. (c) The distances traveled by three classical gaits and gait learned by the proposed method under the same morphology. (d) The visualization of (a), the $x$-axis is the time, and the $y$-axis is the distance from the robot’s center of mass to the start position.

[22] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine, “Stabilizing off-policy q-learning via bootstrapping error reduction,” Advances in Neural Information Processing Systems, vol. 32, 2019.

[23] N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, N. Heess, and M. Riedmiller, “Keep doing what worked: Behavioral modelling priors for offline reinforcement learning,” arXiv preprint arXiv:2002.08396, 2020.

[24] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, “Advantage-weighted regression: Simple and scalable off-policy reinforcement learning,” arXiv preprint arXiv:1910.00177, 2019.

[25] A. Naïr, A. Gupta, M. Dalal, and S. Levine, “Awac: Accelerating online reinforcement learning with offline datasets,” arXiv preprint arXiv:2006.09359, 2020.

[26] Y. Liu, A. Swaminathan, A. Agarwal, and E. Brunskill, “Off-policy policy gradient with state distribution correction,” arXiv preprint arXiv:1904.08473, 2019.

[27] A. Swaminathan and T. Joachims, “Batch learning from logged bandit feedback through counterfactual risk minimization,” The Journal of Machine Learning Research, vol. 16, no. 1, pp. 1731–1755, 2015.

[28] O. Nachum, B. Dai, I. Kostrikov, Y. Chow, L. Li, and D. Schuurmans, “Algaeldice: Policy gradient from arbitrary experience,” arXiv preprint arXiv:1912.02974, 2019.

[29] A. Kumar, A. Zhou, G. Tucker, and S. Levine, “Conservative q-learning for offline reinforcement learning,” Advances in Neural Information Processing Systems, vol. 33, pp. 1179–1191, 2020.

[30] I. Kostrikov, R. Fergus, J. Tompson, and O. Nachum, “Offline reinforcement learning with fisher divergence critic regularization,” in International Conference on Machine Learning. PMLR, 2021, pp. 5774–5783.

[31] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, “Combo: Conservative offline model-based policy optimization,” Advances in Neural Information Processing Systems, vol. 34, 2021.

[32] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims, “Morel: Model-based offline reinforcement learning,” Advances in neural information processing systems, vol. 33, pp. 21810–21823, 2020.

[33] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma, “Mopo: Model-based offline policy optimization,” Advances in Neural Information Processing Systems, vol. 33, pp. 14129–14142, 2020.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in International conference on machine learning. PMLR, 2018, pp. 1861–1870.

[35] S. Lee, Y. Seo, K. Lee, P. Abbeel, and J. Shin, “Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble,” in Conference on Robot Learning. PMLR, 2022, pp. 1702–1712.

[36] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, “Gaussian process optimization in the bandit setting: No regret and experimental design,” arXiv preprint arXiv:0912.3995, 2009.

[37] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in International conference on machine learning. PMLR, 2018, pp. 1587–1596.

[38] C. Wang, G. Xie, L. Wang, and M. Cao, “Cpg-based locomotion control of a robotic fish: Using linear oscillators and reducing control parameters via pso,” International Journal of Innovative Computing Information and Control, vol. 7, no. 7B, pp. 4237–4249, 2011.

[39] A. Crespi, D. Lachat, A. Pasquier, and A. J. Ijspeert, “Controlling swimming and crawling in a fish robot using a central pattern generator,” Autonomous Robots, vol. 25, no. 1, pp. 3–13, 2008.
APPENDIX

A. Training Details
For the individual network and population network, their Critic loss functions are:

\[ J_{Q_{\text{pop}}}(\theta_{t}^i) = E_{(s_t,a_t) \sim D_{\text{pop}}(\cdot)} \frac{1}{2}(r(s_t,a_t) + \gamma \min_{i=1,2} Q_{\theta}(s_{t+1}, \pi_{\phi}(a_{t+1} | s_{t+1}) + \varepsilon) - Q_{\theta_{t}^i}(s_t,a_t))^2 \]  

(11)

\[ J_{Q_{\text{ind}}}(\theta_{t}) = E_{(s_t,a_t) \sim D_{\text{ind}}(\cdot)} \frac{1}{2}(r(s_t,a_t) + \gamma \min_{i=1,2} Q_{\theta}(s_{t+1}, \pi_{\phi}(a_{t+1} | s_{t+1}) + \varepsilon) - Q_{\theta_{t}^i}(s_t,a_t))^2 \]  

(12)

where \( \phi_{t} = \tau \phi_{t} + (1 - \tau)\bar{\phi} \), \( \tau \phi = \tau \phi + (1 - \tau)\bar{\phi} \), which represent target networks for Critic and Actor respectively. The subscript \( i = 1, 2 \) represents the number of the twin Q network, \( \theta \) and \( \theta' \) represent the parameters of Critic of individual network and population network, respectively. \( \varepsilon \sim \text{clip}(N(0, \sigma), -c, c) \) stands for random Gaussian noise, \( \sigma \) denotes the variance of the noise, and \( c \) represents the clip value.

Algorithm 1 Bayesian Optimization Augmented by the Concurrent Networks

1: Initialize replay buffers: \( D_{\text{pop}}, D_{\text{ind}}, D_{\text{init}} \)
2: for each iteration do
3: Initialize and empty \( D_{\text{ind}} \);
4: \( \xi = \xi_{\text{new}} \);
5: for every training episode do
6: for \( t \) in episode length \( T \) do
7: Interact with the environment: \( a_t \sim \pi_{\text{ind}}(s_t) + N(0, \sigma_{\text{exp1}}) \);
8: Observe next state \( s_{t+1} \) and reward \( r_t \);
9: Store transition \( (s_t, a_t, r_t, s_{t+1}) \) to \( D_{\text{pop}} \) and \( D_{\text{ind}} \);
10: Store initial states \( s_0 \) to \( D_{\text{init}} \);
11: end for
12: Set \( R_{\text{last}} = R_{\text{current}} \) and \( R_{\text{current}} = \sum_{t=0}^{T} r_t \);
13: Update behavior cloning term \( \beta \) according to Eq.\,(8);
14: for \( n \) in update numbers do
15: Train \( Q_{\text{pop}} \) and \( \pi_{\text{pop}} \) with random batches from \( D_{\text{pop}} \) according to Eq.\,(11) and \,(9);
16: Train \( Q_{\text{ind}} \) and \( \pi_{\text{ind}} \) with random batches from \( D_{\text{ind}} \) according to Eq.\,(12) and \,(7);
17: end for
18: end for
19: for \( i \) in BO update numbers do
20: Find \( \xi_i \) by optimizing acquisition function over the GP according to Eq.\,(10);
21: Sample initial states \( s_0 \) from \( D_{\text{init}} \);
22: Calculate the objective value \( F(\pi_{\text{pop}}, \xi_i) \) according to Eq.\,(9);
23: Augment \( D_{\text{BO}}^{\xi_i} = \{ D_{\text{BO}}^{\xi_i-1}, (\xi_i, F(\pi_{\text{pop}}, \xi_i)) \} \) and update the GP;
24: end for
25: \( \xi_{\text{new}} = \arg \max_{i} F(\pi_{\text{pop}}, \xi_i) \)
26: end for

In summary, the pseudo-code of the whole algorithm is described in Algorithm 1

Besides, all the hyperparameters and network architectures utilized in the experiments are listed in Tab. II.

| Hyperparameters | Value                  |
|-----------------|------------------------|
| Optimizer       | Adam                   |
| Learning rate   | \( 3 \times 10^{-4} \) |
| Target update weight | \( 5 \times 10^{-3} \) |
| Batch size      | 256                    |
| Policy noise std | 0.2                    |
| Policy noise clip | 0.5                    |
| Policy update frequency | 2                     |
| Capability of \( D_{\text{pop}} \) | \( 1 \times 10^{6} \) |
| Capability of \( D_{\text{ind}} \) | \( 1 \times 10^{7} \) |
| Length of episode | \( 1 \times 10^{5} \) |

| Offline | \( \alpha \) | \( 0.4 \) |
|----------|-------------|-----------|
| \( K_p \) | \( 3 \times 10^{-5} \) |
| \( K_d \) | \( 8 \times 10^{-5} \) |

| Architecture | Network depth | 3 |
|-------------|--------------|---|
|             | Network width | 200 |

| Bayesian | Bayesian optimization steps | 30 |
|----------|----------------------------|----|
|          | Random exploration steps   | 30 |

B. Simulation Environment Details

HalfCheetah. The objective of HalfCheetah is to move forward as fast as possible while minimizing the action cost. It has an 18-dimensional state space consisting of body position, quaternion of body, joint position, body linear velocity, body angular velocity, and joint velocity. Actions have six dimensions and are torque applied to six joints. The rewards is set to \( r_t = \frac{x_{t+1} - x_t}{0.05} - 0.1 \| a_t \|_2^2 \). Where \( x_t \) is the position of \( x \)-axis at time step \( t \). The morphology parameters that we modify are the length of the front and rear legs (6 dimensions in total). The original morphology parameters of the agents are the same as the Gym. The modification range of the length of each part is \([0.5 - 1.5]\).

Ant. The objective of Ant is to move forward as fast as possible while minimizing the action cost. It has a 41-dimensional state space consisting of body position, quaternion of body, joint position, body linear velocity, body angular velocity, joint velocity, the cartesian orientation of body frame, and cartesian position of body frame. Actions have eight dimensions and are torque applied to eight joints. The rewards is set to \( r_t = \frac{x_{t+1} - x_t}{0.05} - 0.5 \| a_t \|_2^2 + 1.0 \). Where \( x_t \) is the position of \( x \)-axis at time step \( t \). The morphology parameters that we modify are the length of the thighs and calves of four legs (8 dimensions in total). The original morphology parameters of the agents are the same as the Gym. The modification range of the length of each part is \([0.5 - 1.5]\).

C. Physical Experiment Details
1) Simulation Model: The four-legged robot is constructed using URDF files, which contain the appearance,
physical properties, and joint types of the robot. Therefore, we can easily modify the robot parameters according to the physical environment. The physics engine utilizes Pybullet, which is friendly to URDF files. We need to modify the structural parameters of the robot in real-time according to the results of the optimization algorithm, so we don’t use Mesh files to form the morphological structure, but used cylinders and cubes to form the four-legged robot.

2) Simulation Environment.: The objective of the four-legged robot is to move forward as fast as possible while minimizing the action cost. It has a 36-dimensional state space consisting of body position, body orientation, body linear velocity, body angular velocity, joint position, joint position history, joint velocity, and joint velocity history. Action has four dimensions and are CPG parameters. The reward function is designed to \( r_t = \frac{x_t}{1000} - \frac{x_0}{1000} - 0.5\|a_t\|^2 + 1.0 \). Where \( x_t \) is the position of x-axis at time step \( t \). The morphology parameters that we modify are the length and width of the body, the length and radius of the four legs. To keep the robot symmetrical, we set the length and radius of the left and right leg of the robot to always be consistent (6 dimensions in total).

3) Physical Robot.: The four-legged robot is composed of body parts and four legs, which are all 3D printed with the material of polylactic acid (PLA). The robot is controlled by the main control unit (ATmega328) located at the back of the body. The four legs of the robot are connected to the body with four servo motors (MG90S), which provide torque from 2.0kg/id to 2.8kg/cm. Two rechargeable 3.7 volts cylindrical lithium batteries (LR14500) are used as the power modules of the robot. Besides, two adjustable boost circuit modules (SX 1308 DC-DC) are utilized to increase the voltage of one lithium battery to 5 volts for the motors and that of the other lithium battery to 6-9 volts for the main control chip. Moreover, the legs are designed to be detachable and assemblable, which greatly facilitates the experiment of morphology optimization.

4) Central Pattern Generator.: Central pattern generators (CPGs) are neural circuits found in nearly all vertebrates, which can produce coordinated patterns of rhythmic movements. It can be modeled as a network of coupled non-linear oscillators where the dynamics of the network can be determined by the set of differential equations.

\[
\dot{\phi}_i = 2\pi f_i + \sum_{j \in \Omega_i} \mu_{ij}(\phi_j - \phi_i - \varphi_{ij}) \\
\dot{r}_i = a_r(R_i - r_i) - 2a_r \dot{r}_i \\
\dot{x}_i = a_x(X_i - x_i) - 2a_x \dot{x}_i \\
\dot{\theta}_i = x_i + r_i \cos(\phi_i)
\]

where \( \phi_i, r_i \) and \( x_i \) are three state variables, which represent the phase, amplitude, and offset of each oscillator. The variable \( \theta_i \) is the output of the oscillator, which are the position control commands in our experiments. The parameters \( f_i, R_i, \) and \( X_i \) are control parameters for the desired frequency, amplitude, and offset of each oscillator. \( \mu_{ij} \) is the coupling weights that change how the oscillator influence each other. The constant \( a_r \) and \( a_x \) are constant positive gains and allow us to control how quickly the amplitude and offset variables can be modulated. \( \Omega_i \) is the set of all oscillators that can have a coupling effect on oscillator \( i \). And the parameter \( \varphi_{ij} \) is the desired phase bias between oscillator \( i \) and \( j \), which is utilized to determine the gaits. Moreover, the subscripts \( i = 1,2,3 \) and \( 4 \) represent the left front leg, right front leg, left rear leg, and right rear leg of the legged robot, respectively.

Our purpose is to train the gaits of the robots, which is determined by \( \varphi_{ij} \), so we fixed other parameters, \( f_i = f = 10Hz, R_i = R = 0.4rad, X_i = X = 0.04rad, a_r = 20, a_x = 20, \mu_{ii} = 0 \) (i.e., all oscillators have no self-couplings), \( \mu_{ij} = \mu = 20 \) for \( i \neq j \). Up to now, we have obtained a simplified CPG model with only \( \varphi_i \) to be determined, which is \( 4 \times 4 \) matrix, the value of each item can be calculated with the following formula:

\[
\varphi_{ij} = \varphi_j - \varphi_i
\]

So the whole values that we need to learn are four \( \varphi_i \) (for \( i=1,2,3,4 \)).