Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

Co-optimization of Morphology and Behavior of Modular Robots via Hierarchical Deep Reinforcement Learning

Jieqiang Sun[1], Meibao Yao[1,*], Xueming Xiao[2], Zhibing Xie[1], Bo Zheng[3]

***Abstract*—Modular robots hold the promise of changing their shape and even dimension to adapt to various tasks and environments. To realize this superiority, it is essential to find the appropriate morphology and its corresponding behavior simultaneously to ensure optimality of the reconfiguration. However, achieving co-optimization is challenging because robotic configuration and motion are interactive and coupled with each other, as well as their optimization processes. To this end, we proposed a co-optimization framework based on hierarchical Deep Reinforcement Learning (DRL), consisting of a configuration model and a motion model based on the Twin Delayed Deep Deterministic policy gradient algorithm (TD3). The two network models update asynchronously with a shared reward to ensure co-optimality. We conduct simulations and experiments with the Webots platform to validate the proposed framework, and the preliminary results show that it yields high quality optimization schemes and thus allows modular robots to be more adaptive to dynamic and multi-task scenarios.**

I. Introduction

Modular robots are a collection of autonomous machines composed of homogeneous or heterogeneous modules that can reconfigure their shape to perform various tasks and adapt to different environments. This superiority is achieved by optimizing the reconfiguration process to yield the appropriate morphology and corresponding behavior. As observed from the evolutionary biology, living creature's behavior results from a synergy of the body and brain to their interact with the environment. A concurrent optimization of robot configuration and motion could ensure a global optimum of reconfiguration schemes by enlarging the candidate space, thereby achieving a higher level of adaptability to various surroundings. However, co-optimization is challenging because robotic motion is generally dependent on the morphology, and their optimization processes are coupled with each other. As a result, early research mostly focused solely on optimizing the configuration or motion of modular robots separately, and we first investigated the aspects of configuration design and motion control as follows.

[1]Intelligent Robotics Lab (IRL), School of Artificial Intelligence, Jilin University, Changchun, 130012, China; Engineering Research Center of Knowledge-Driven Human-Machine Intelligence, Ministry of Education, China.

[2]CVIR Lab, Changchun University of Science and Technology, Changchun, 130012, China.

[3]Shanghai Aerospace Control Technology Institute, Shanghai, 201109, China.

[*]Meibao Yao is corresponding author. Email: meibaoyao@jlu.edu.cn

Recent developments in system design and hardware implementation facilitated increasingly versatile modular robotic systems, along with improved configuration design methods contingent to them [1]–[7]. An ROS2Learn framework based on reinforcement learning was proposed to optimize configuration of the Modular Articulated Robotic Arm (MARA) established in the work [8]. A DQN-best-first algorithm was suggested as a search heuristic to add desired parts to the robotic arm. However, the robotic configuration is generally optimized in terms of reachability, dexterity and other kinematic metrics, while the efficiency-related indicies such as time-saving and energy metrics characterized by system dynamics and motion are just as significant to the task-oriented scenarios.

The other aspect to enhance adaptability of modular robots is optimized motion control in various tasks. Early work on this issue focused on designing rule library attributed to expertise using specified grammas in separate tasks such as docking and overcoming obstacles [9]–[13]. To ensure the continuity of motion implementation in more long-term tasks, logics of conditional judgment (if-then) and for-loops were utilized in path planning and locomotion tasks [14]–[17]. In more complex tasks such as navigation and finding a target, the mapping of segmented task goals to action sequences become much less obvious. A modular robotic system named SMORES-EP equipped with an RGB-D camera can preprocess visual information into high level features and annotations, and further map them to motion commands. However, this correspondence is still based on expertise and cannot guarantee optimality of the motion control and planning in modular robots.

To avoid sub-optimality and limitations brought by empirical design, there are a growing number of evolutionary and learning-based methods to optimize motion of modular robots [18], [19]. Genetic algorithm (GA) was utilized to search and optimize locomotion gaits of modular robotic systems of the M-Tran, iMobot and UBot robots in various environments [20]–[22]. Motion control models based on Central Pattern Generator (CPG) was employed to yield cyclical motion of the actuators [23], [24], and GA and reinforcement learning were applied to optimize the network parameters of CPG [25], [26]. The TRPO and DDPG algorithms were used to train simple rowing and crawling motions of Snapbot with a fixed configuration [27]. In the above study, researchers

optimize the motion control strategy of modular robots in a variety of ways, but often limited to a fixed morphology and ignore the interaction between motion and the robotic body.

With the development of more advanced representation and optimization techniques, research on the co-optimization issue has been facilitated over the past decades. Evolutionary algorithms(EA) are used for the co-design of morphology and control of soft tensegrity modular robots [28]. The parameters of morphology and motion were encoded to genomes, and Multi-Objective Genetic Algorithm (MOGA) was proposed to optimize the performance of modular robots for a steering task [29]. Experimental results verified that the co-evolution can yield better reconfiguration schemes compared with optimizing motion on a fixed morphology. However, the verification scenario is relatively simple without fully considering the performance of modular robots in multi-task scenarios.

In this work we present a DRL-based framework with a hierarchical structure to optimize configuration and motion concurrently in multiple tasks including climbing stairs, avoiding obstacles, and finding a target. To cope with the scenario complexity, we integrate an RGB-D camera into the robotic system for higher-level online perception and scene understanding. The hierarchical framework contains a configuration model and a motion model based on TD3 that update asynchronously with a shared reward, and finally yields co-optimized schemes of configuration and motion.

In summary, the main contributions of our work are as follows:

1) We propose a hierarchical DRL-based framework that concurrently optimizes morphology and motion of modular robots with depth images as input, enhancing their adaptability to complex environments.
2) We propose a motion generation module based on TD3 that can adapt to various scenes by incorporating motion modes and optimizing separate groups of motion parameters, and its effectiveness is demonstrated in multi-task scenarios.
3) We conduct simulations and experiments with the Webots platform, and demonstrate that the proposed algorithm can yield high-quality reconfiguration schemes for modular robots in randomized and multi-task environments.

II. Methodology

A. Problem Formulation

To achieve the co-optimization, we first concurrently describe the robotic morphology and behavior as a tuple $\mathcal{H} = \{\boldsymbol{C}, \boldsymbol{M}\}$, where $\boldsymbol{C}$ denotes the robotic configuration and $\boldsymbol{M}$ represents its motion strategy. The configuration of a modular robot can be modeled as $\boldsymbol{C} = \{\boldsymbol{C_a}, \boldsymbol{C_s}\}$, where $\boldsymbol{C_a}$(10×10 matrix) describes the connection relationship among modules in the global structure, and $\boldsymbol{C_s}$(10×5 matrix) describes the connected ports of local modules, see Fig.1 as an example of a chain-type modular robot. $\boldsymbol{C_a}$ is represented by a binary 0-1 matrix, where $\boldsymbol{C_a}(i,j) = 1$ denotes that module i and module j are connected, and otherwise not connected. $\boldsymbol{C_s}$ contains five elements to describe the connectors of a pair of modular individuals, as $(module_i, module_j, face_i, face_j, connection)$. Taking four modules as an example, each row of $\boldsymbol{C_s}$ in Fig.1 shows a complete description of one connected pair. We can utilize $\boldsymbol{C_s}$ matrix to store information of every connection in a robotic configuration. On the other hand, the motion strategy

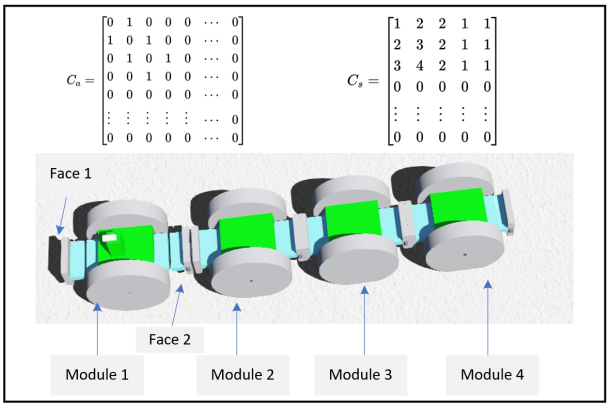

Fig. 1: Modeling the configuration of a chain-type modular robot with 4 modules.

$\boldsymbol{M}$ of the modular robot is represented by a multi-layer neural network, where the measured values of each sensor are fed and outputs signals as processed motor control commands that executed by all modules.

The modular robots can change their number of modules and connections when confronting with different scenes in a scenario, then the robots will implement various tasks in correspondence to the scenes. In this work modular robots are deployed in multi-task scenario, including climbing stairs, avoiding obstacles and finding targets. In this process, we consider the following three elements for evaluation: time consumption $T(\mathcal{H}) = t_{use}/t_{init}$, task completion rate $D(\mathcal{H}) = d_{finish}/d_{init}$, and energy consumption $E(\mathcal{H}) = E_{final}/E_{init}$. Therefore, we pose this problem as a multi-objective optimization to maximize the comprehensive task completion rate while minimizing the energy and time consumed by the modular robots. In summary, we obtain the objective function as:

$$F(\mathcal{H}) = -\omega_t T(\mathcal{H}) - \omega_e E(\mathcal{H}) + \omega_d D(\mathcal{H}). \tag{1}$$

where $\omega_t, \omega_e, \omega_d$ are user-set weights that can be used to adjust the impact of the multiple objective on the results. We hope to find an optimal solution $\mathcal{H}^* = (\boldsymbol{C}^*, \boldsymbol{M}^*)$ to maximize the objective function:

$$\mathcal{H}^* = \arg\max_{\mathcal{H}} F(\mathcal{H}). \tag{2}$$

In this work, our goal is to search for this optimal solution for a chain-type mobile modular robot by co-optimizing the robotic configuration and motion.

B. Method Overview

We propose a hierarchical DRL-based framework to cope with the co-optimization problem described by Eq. 2. The schematic is shown in Fig.2, consisting of a scene classifier and an optimizer. To enable the robot to change its configuration when facing up with different scenes, we devise a scene classifier ahead of the optimizer. We pre-trained the scene classification model based on ResNet18, with the sensed depth image as input, and the scene identified result is recorded as P_i, the next result as $P_i^{'}$. We then input the environmental information, including the depth image and the result of the classifier, and the robotic states to the optimizer to generate the optimized robotic configuration and motion.

The right side of the schematic illustrates the upper level of the optimizer, the configuration model, and the lower level shows the motion model. The sequential selection of robotic configuration in different scenes in a scenario conforms to a semi-Markov Decision Process (SMDP). At each scene switching point when $P_i \neq P_i^{'}$, with a given state $s_t \in S_c$, the configuration model chooses an action $a_t \in A_c$ to select a configuration based on the policy $\pi : S_c \rightarrow A_c$ to obtain the reward r_t and the new state $s_{t+1} \in S_c$. This process can be expressed as a tuple $\tau_1 = (\mathcal{S}_c, \mathcal{A}_c, p, r)$ with state transition probability $p\left(s_{t+1} \mid (s_t, a_t)\right)$. Since the time-intervals of state transitions of the configuration model is flexible, this process can be described as SMDP.

The lower level motion model generates a motion control command $a_m \in \mathcal{A}_m$ at each time step t, based on the specified configuration, with $s_m \in \mathcal{S}_m$ as input, updating the motion network parameters by reward $r(s_m, a_m)$ until multi-task terminated; the evaluation metric is calculated and used to update the configuration model. This process can be described as a classical Markov Decision Process (MDP), expressed as a tuple $\tau_2 = (\mathcal{S}_m, \mathcal{A}_m, p, r)$. The difference with the configuration model is whether the time-intervals of the signal output is fixed or not. The neural network of the two models is trained simultaneously but updated asynchronously via a shared reward.

The state space $\mathcal{S}$ consists of two parts, the state space $\mathcal{S}_c$ for determining robotic configuration, solely containing environmental perception, and the state space $\mathcal{S}_m$ for determining robotic motion, containing environmental information and the robot state. The action space $\mathcal{A}$ also consists of two parts, the candidate configuration space $\mathcal{A}_c$ and the motion space $\mathcal{A}_m$ containing continuous motion control commands. The goal of co-optimization is to find the optimal policy of choosing the robotic configuration from $\mathcal{A}_c$ and motion strategy from $\mathcal{A}_m$, which maximizes Equation (1).

It is noted that the co-optimization process consists of several iterations of 1+m decision-making steps. In each iteration, the robot configuration is determined at the first time step, and the next m time steps execute the action commands output by the motion model. When the scene classifier judges terrain switching, the optimization will proceed to the next iteration, and the configuration model transfers to the next state and choose a new robot configuration. With the hierarchical framework at two levels, the co-optimization of robotic configuration and motion for multi-task scenarios can be realized. We describe details of the two models below.

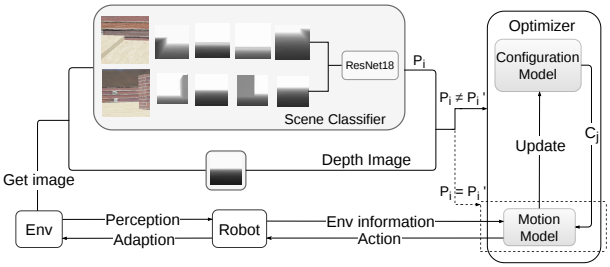

Fig. 2: Overview of the proposed DRL-based co-optimization framework, mainly consisting of two parts: a scene classifier and an optimizer that includes a configuration model and a motion model.

Algorithm 1: Configuration model optimization algorithm

Initialize :
Replay memory $\mathcal{D}$ to capacity $\mathcal{N}$;
Action-value function Q with random weight θ;
Target action-value function $\hat{Q}$ with weight $\hat{\theta} = \theta$;
for episode = 1, M **do**
  Initialize sequence $S_0 = \{O_0\}$ and preprocessed sequence $\phi_0 = \phi(S_0)$
  for t = 1, T **do**
    With probability ϵ select a random action A_t
    otherwise select A_t = argmax$_a Q(\phi(S_t), a; \theta)$;
    Execute action A_t to update the matrices C_s and C_a, observe reward R_t and data observation O_{t+1};
    episode terminate set D_t = 1, else D_t = 0;
    Set $S_{t+1} = \{S_t, A_t, O_{t+1}\}$ and preprocess $\phi_{t+1} = \phi(S_{t+1})$;
    Store transition $(\phi_t, A_t, R_t, D_t, \phi_{t+1})$ in D;
    Sample random mini batch of transitions$(\phi_i, A_i, R_i, D_i, \phi_i^{'})$;
    If D_i = 0, set $Y_i = R_i + \gamma \text{max}_{a'} \hat{Q}(\phi_i^{'}, a^{'}; \hat{\theta})$ else $Y_i = R_i$;
    Perform a gradient descent step on $(Y_i - Q(\phi_i A_i, \theta))^2$ with respect to the Network parameters θ;
    Every K steps reset $\hat{Q} = Q$
  end for
end for

C. Configuration Model

The configuration model maps from a depth image observed by the robot to a selected configuration, and the goal is to yield an optimal robotic configuration $\boldsymbol{C}^*$ based on the current scene. To achieve this, we design a neural network for the configuration model containing a Convolution Neural

Networks (CNNs) encoder, as depicted in Fig.3. Both of the first two convolutional layers have a kernel size of 5*5 with a padding set at 2, 32 and 64 feature maps, respectively. Each convolutional layer is followed by a maximum pooling layer of 2*2. The first fully connected layer has 256 hidden units and the second outputs a vector represents the Q value evaluated for each configuration, whose dimension n is determined by the configuration space of the modular robots. The scene changing judged by the scene classifier triggers a new signal output of the configuration model. The trigger mechanism is designed as: each iteration accounts for several time steps, the value set as 100 in our simulation, the last 10 observed depth images in the iteration is passed through the scene classification model which performs classification, if more than 70% of the results are consistent, then the scene classifier outputs the final result. The selected $\boldsymbol{C^*}$ will be used to update the matrices $\boldsymbol{C_s}$ and $\boldsymbol{C_a}$ to indicate the change of the robot configuration.

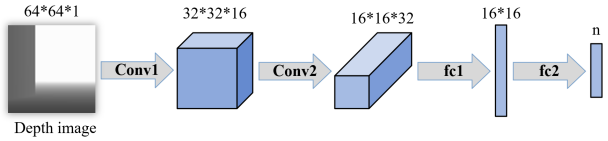

Fig. 3: Network architecture of the configuration model.

Our configuration model inherits some basic components of the DQN algorithm [30], and the optimization process is shown in Algorithm 1. The convolutional neural network generates the target Q-value according to the input state S, and can evaluate the Q-value of the next state according to the target Q-value in the current state. Adding the experience playback mechanism and the target network to the model can greatly reduce the correlation among data. When entering the state s, we can get the Q values of all actions. The Q-learning-based algorithm is updated as below:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_{A_{t+1}} Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right) \right]. \quad (3)$$

During the training process, the historical data is continuously collected and stored in an experience buffer, and then some data is sampled from the buffer to apply the Mini-Batch gradient descent to update the Q-value. In this way, the optimal configuration is obtained.

D. Motion Model

To accurately control the modular robot and enhance its adaptability to various task scenarios, the motion model should output control commands continuously. In addition, there may be a large number of boundary actions required in the process of implementing the task. For example, when the robot is moving, full speed is often the optimal strategy with minimal time consumption. In view of the above two considerations, we propose a motion model (see Algorithm 2) based on the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [31], which adds noise to the action output and crops it to a certain range, leading to continuous boundary actions for exploration. Fig.4 shows an overview of the network architecture of the motion model and Fig.5 illustrates the actor network.

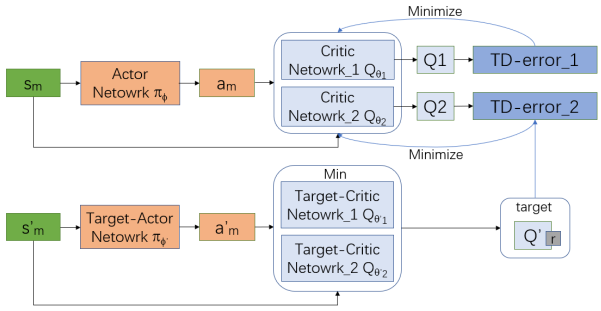

Fig. 4: The overview of the network architecture of the motion model.

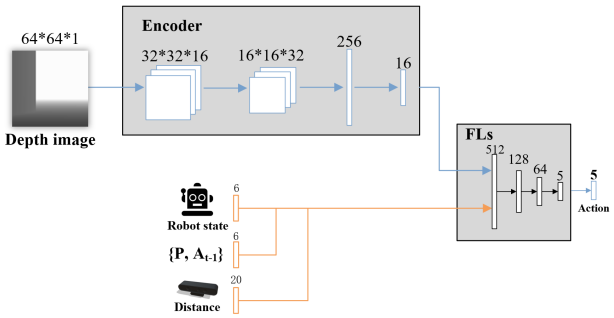

Fig. 5: The actor network architecture of the motion model.

Network architecture: Our motion model is based on the actor-critic network architecture, in which two critic networks output the state-action values, i.e., Q-values, and we utilize the smaller value to update the actor network. We use the target network to output the target Q values to update the critic network (see Fig.4). The actor network consists of a Convolution Neural Network (CNN) encoder and four Fully Connected Layers (FLs). The critic networks are similar to the actor networks and the target actions are directly input to the FLs. The encoder architecture is similar to the configuration model but with a 16-dimension output in the last layer. A depth image is fed into the encoder as input, and each pixel of the depth image represents a specific distance measurement, ranging from 0.01 to 1. The robot states, composed of a proportion of the remaining electricity (1-D), current position (3-D) and current speed (2-D), combined to another vector including depth image after feature extraction, previous reward r_{t-1}, previous action a_{t-1} and distance sensor data, are also input to the FLs successively after data processing (see Fig.5).

Motion mode: Different from conventional mobile robots, modular robots can adapt to a variety of scenarios, while their motion control requirements are more complex, with large

Algorithm 2: A TD3-based motion optimization algorithm

Initialize :
Critic network $Q_{\theta_1}, Q_{\theta_2}$, actor network π_ϕ with random parameters θ_1, θ_2, ϕ;
Target network $\theta_1^{'} \leftarrow \theta_1, \theta_2^{'} \leftarrow \theta_2, \phi^{'} \leftarrow \phi$;
Replay buffer β, learning rate of actor network α_1 and critic network α_2;
Update interval d, soft update parameter μ;
Configuration c output by Algorithm 1;
for each episode **do**
  while not end condition **do**
    Obtain motion state $s_m \in \mathcal{S}_m$;
    Obtain action with exploration noise and clip to (-1,1), $a_m \sim clip(\pi_\phi + \epsilon)$, $\epsilon \sim \mathcal{N}(0, \sigma)$;
    Execute action a_m in the specified motion mode;
    Obtain reward r and new state $s_{m,}^{'} \in \mathcal{S}_m$;
    Store transition tuple $(s_m, a_m, r, s_m^{'})$ in β;
    Sample mini-batch of N_m transitions $(s_m, a_m, r, s_m^{'})$ from β;
    Obtain the target action $a_m^{'} \leftarrow clip(\pi_{\phi'}(s_m^{'}) + \epsilon)$;
    Update critics θ_i by Equation (13);
    if episode mod d **then**
      Update ϕ by Equation (14);
      Update target networks:
      $\phi^{'} \leftarrow \mu\phi + (1-\mu)\phi^{'}$;
      $\theta_i^{'} \leftarrow \mu\theta_i + (1-\mu)\theta_i^{'}$;
    end if
  end while
end for

and time-varying controller dimensions. To cope with this, we integrate some design priori in the form of motion modes in the devise of motion model, which has the promise of improving network performance while accelerating network convergence. We design several motion modes such as stair terrain mode and flat terrain mode corresponding to multi-task scenarios discussed in this work, including going up stairs, avoiding obstacles in random positions, and finding targets. With different motion modes, the control commands given by the motion model will be executed by the robot in different manners.

To design the motion modes, we first analyze the action space and how the motion control commands are processed. There are five dimensions in the action space and each output component of the motion model is between [-1,1]. The first two dimensions are related to the motion speed of the robot. The remaining three dimensions are related to the climbing behavior of the robot.

In all scenarios, the modular robot will process the first two-dimension of the motion control command according to the following equation, we record $V_{straight}$ as the speed of the straight motion, $V_{turning}$ as the turning motion, to obtain the left and right wheel speed of each module:

$$V = \omega_1 * V_{\text{straight}} \pm \omega_2 * V_{\text{turning}} \tag{4}$$

However, in motion modes of different scenarios, the weights ω_1, ω_2 may be different, depending on the intensity and frequency of the turning behavior. For example, in the narrow terrain mode, ω_1 decreases and ω_2 increases compared to flat terrain mode.

In rough terrain and stair terrain motion modes, the remaining three-dimensional motion control commands will be processed into motor commands by the modular robot as follows:

$$P = \omega_3 * Asin(t + \omega_4 * B), \tag{5}$$

where A and B are the amplitude and phase of the signal, respectively. These two parameters are obtained by calculation of the last three dimension of the motion control command. t denotes the current time step relative to the initial time of an episode. Using Equation (5), we calculate the position control command P of the motor steering gear. These particular motor commands will only be executed by the modular robot in the motion mode requiring climbing action. For example, flat terrain motion mode will only execute the two-dimensional motion control command. Similarly, in different motion modes, the weights ω_3 and ω_4 may be different. In the simulation, we mainly use the stair terrain and flat terrain motion modes.

Reward function: The motion model and configuration model share the reward and update asynchronously. While the update of the configuration model considers the sum of the rewards of each terrain, the motion model updates itself at each time step. Our optimization goal of the motion model is to obtain a motion strategy that approaches the maximum of the objective function in Equation (1), which can trade-off among the multiple optimization goals $T(\mathcal{H}), E(\mathcal{H}), D(\mathcal{H})$. Therefore, the reward function of the lower TD3-based motion model is designed as follows:

$$R_1 = \tau_1 \cdot \Delta D + \tau_2 \cdot \cos\beta + \tau_3 \cdot r_u + \tau_4 \cdot r_g \tag{6}$$

We denote $\Delta\ D$ as the change of distance from the target in two adjacent time steps. β represents the angle between the straight line connecting the robot and goal, and the robot speed forward. When the robot reaches the undesirable terminal states given a penalty value $r_u = -10$. These states include collision with obstacles, the robot overturning and leaving the area while climbing the stairs. Our goal-reaching reward $r_g = 100$ is related to the time consumed when the multi-task is completed, we define τ_4 as (1 + $\alpha_g(t_{th}$ / $t_{sum})$. The two end conditions will break the training episode. In the simulation, we set τ_1 = 100, τ_2 = 0.05, τ_3 = 1.0, t_{sum} = 3500. And t_{th} represents the time step used by the robot in a training episode.

In the configuration model of the upper layer, the reward function is designed as follows:

$$R_2 = \sigma_1 \cdot R_1 + \sigma_2 \cdot r_s + \sigma_3 \cdot r_c \tag{7}$$

The first component is the cumulative reward of the lower level motion. The second component is to impel the robot climb up higher stairs, the robot obtains different rewards

through stairs of different heights, which we define as $r_s = 60$. The third component is the energy E_c consumed by the robot when passing through the stair, $r_c = E_c$. In the simulation, when the robot is on stair terrain we set σ_1 = 1.0, σ_2 = (1 + s_{ht} / s_{base}), σ_3 = 0.003, s_{base} = 0.04. When the robot passes through the stair terrain, σ_1 = 1.0, σ_2 = 0, $\sigma 3$ = 0. And s_{ht} represents the height of the stairs in the current terrain.

Update method: The state-action value, also known as Q-value, is used as estimator of the expected future return

$$R_{1_t} = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i), \tag{8}$$

where $\gamma \in [0, 1]$ is a discount factor to adjust the priority of short-term rewards, and T is the total number of time-steps taken. When performing action a_t in state s_t, we have:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_{1_t} | s_t, a_t]. \tag{9}$$

The above Bellman equation describes the relationship between the two state-action pairs, (s_m, a_m) and the subsequent $(s_m^{'}, a_m^{'})$, as below:

$$Q^{\pi}(s_m, a_m) = r + \gamma \mathbb{E}_{s_m^{'}, a_m^{'}}[Q^{\pi}(s_m^{'}, a_m^{'})], a_m^{'} \sim \pi(s_m^{'}). \tag{10}$$

In actor-critic-based method, the loss of the critic network can be designed as:

$$L(\theta_i) = \min_{i=1,2} N_m^{-1} \sum_{i=1}^{N_m} (Q_{\theta_i}(s_m, a_m) - y)^2, \tag{11}$$

where

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s_m^{'}, \pi_{\phi'}(s_m^{'})). \tag{12}$$

and the critic network can be updated by:

$$\theta_i \leftarrow \theta_i - \alpha_2 \nabla_{\theta_i} L(\theta_i). \tag{13}$$

After the critic network has been updated, the policy, known as the actor, can be updated through the deterministic policy gradient algorithm [32]:

$$\phi \leftarrow \phi - \alpha_1 \mathbb{E}_{s_m \sim p_{\pi_\phi}}[\nabla_{a_m} Q_{\theta_1}(s_m, \pi_\phi(s_m)) \nabla_\phi \pi_\phi(s_m)]. \tag{14}$$

III. Simulation and Experiment

A. Simulation framework and environment

Our models are implemented in Python using Pytorch and trained on a computer with a GeForce RTX 3090Ti GPU. We use Webots platform to implement the simulation experiment, and introduce the simulation environment used to validate our method below.

Simulation framework: In this experiment, multiple modules of the robot can transmit the environmental information perceived by their sensors to a central brain, and then the central brain distributes the control command output from the motion model to each module. In Webots, we used a supervisor as the central brain and an emitter-receiver mechanism to implement the simulation framework (see Fig.6). The supervisor can process sense information sent by multiple modules simultaneously. After sorting and normalizing all the state information of one step, the supervisor sends it all to the network models to get their output. At the beginning of an episode, the supervisor sends the depth image faced by the modular robot to the configuration model to obtain and implement the configuration. After that, the supervisor sends information to the motion model to get the action, and then distributes the action to each module. Each module processes the action into a motor command for the current specified motion mode.

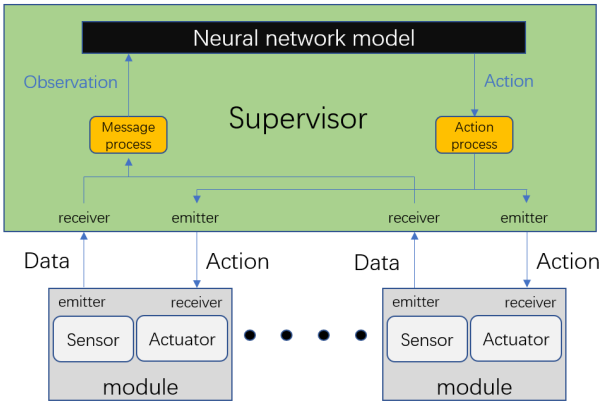

Fig. 6: Simulation framework for our co-optimization method.

System overview: The system design of the modular robot is based on the prototype of YaMoR [33]. Each module has a relatively independent ability to collect environment information and implement actions (see Fig.7, the z-axis refers to the height axis). The length of each module is 0.16 meters, the height of its main part is 0.12 meters, and the wheel radius is 0.06 meters. The sensors used include some touch sensors, some distance sensors and a rangefinder sensor which is unique to the first module (initial module). Each module has two connecting plates that attract or repel other modules by controlling magnets and resembles the EP-Face introduced in [34], in principle. Each module has four motors to realize the three-dimensional motion of the whole modular robots. These motors include two wheel motors for flat motion of the modules, and two Z-axis motors for relative movement along the Z-axis between the modules by swinging the connecting plates up and down. Each module has a battery and its charge is known by the simulator at all times.

Implementation of motion models: In this experiment, we mainly apply two motion modes, stair terrain motion mode and flat terrain motion mode. The control commands given by the motion model are parsed into motor commands that each module can execute according to the current motion mode. We calculate the speed of the left and right wheels using Equation (4), and then each module uses its wheel motors to execute the speed control command. In stair terrain motion mode, we obtain the motor-steering gear position

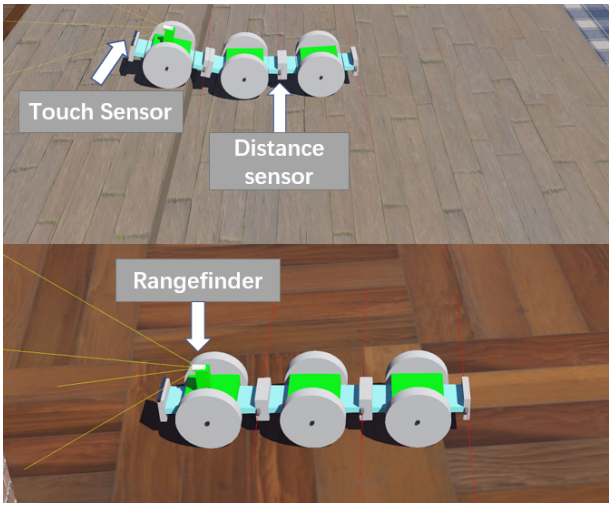

Fig. 7: Modular robotic system in various task scenarios: on stair terrain (upper image), and on flat terrain (lower image), with different motion modes.

control command from Equation (5), which is executed by the z-axis motor at the front of each module (except the first module).

In the simulation experiment, the difference in the configuration is mainly the number of modules, except for Face1 of the first module, both faces of the other modules can be connected in an unconnected state, the exact connection state being determined by the configuration model of the upper layer. A single-module robot is directly excluded as it cannot complete the task of stair climbing, the number of configurations can be selected from the range [2, 10]. When the robot enters the first terrain, its number of modules is recorded as C_n for the decision-making primary configuration. When the terrain classifier classifies the depth image acquired by the robot as the second terrain, the candidate space for the number of configuration modules reduces to [2, C_n]. The first two dimensions of the five-dimensional control command are used to calculate ω_1 and ω_2 in Equation (4), while the third and fifth dimensions are used to calculate ω_3 in Equation (5) (two-module robots only utilize the third dimension), and the fourth dimension is used to calculate ω_4 in Equation (5).

Multi-task scenario: As Fig.8 shows, our simulation environment contained two main regions for the multi-task implementation. In the stair terrain area, there are three steps, the step height appears randomly at three heights of 0.04, 0.06, and 0.08, and the size of flat area for obstacle avoidance task is 6*6, with each obstacle size 0.3*0.3*0.4. When training the model, the stair heights are randomized in each episode and the number of obstacles is 6, the position of the obstacles is changed randomly at each episode, and the size of the random area is 6*3. In the test phase, we test the models' performances using a different number of random obstacles, and the stair heights are also random.

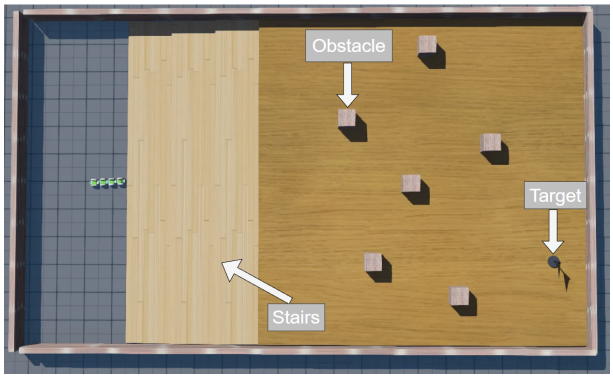

Fig. 8: Simulation of multi-task scenario: stair climbing, traversing and obstacle avoidance.

B. Comparative studies

For evaluation of the proposed method, we compared the performance of different algorithms and models in various environments. We remove the upper layer of configuration model and initialize a configuration without changing it. We choose several representative configurations C_r with module number 2, 4, 6, 8 and 10. to carry out the experiment. In addition, we also compared the performance of the Soft Actor Critic (SAC) [35] algorithm. For convenience, these two models are called TD3-C_r and SAC.

Implementation Details: For TD3, the action output exploration noise is sampled from the normal distribution, and the mean value of this normal distribution is 0, and the variance is attenuated from the initial 0.12 to 0.02 in 4000 generations. The learning rate of the actor and critic networks are chosen as 0.00005, discount γ as 0.99, soft update parameter μ 0.05 and batch size 256. Every $d = 3$ transitions, we update the actor network and target network. An episode is terminated when the modular robot action covers more than 3500 steps. The upper limit of power use of each module is $20000J$. For SAC, all the hyper parameters are basically consistent with TD3. For all models, the first 300 episodes randomly output actions to generate exploratory experiences into the buffer pool, and then they train for another 4000 episodes.

The training curves are shown in Fig.9. Table I compares the performance of using TD3 in motion mode, using the upper configuration model and not using it in the default scene. Table II compares the performance of different algorithms in different training environments. The default test environment is with random stair height of 0.04, 0.06, 0.08 meter and 6 random obstacles. The **Battery** and **Time** column represent the average energy and time consumed by all successful cases during the testing phase. In order to further reflect the performance differences of different models, we counted the success rate of Robot reaching the goal under three different stair heights and the average of the three. The **Scenario** column shows additional changes to the default environment. All models are trained with a random seed of 1.

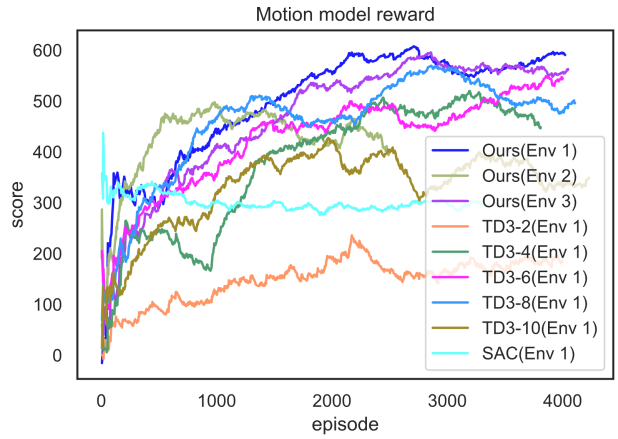

Fig. 9: Performance comparison of various methods. Score represents the sum in reward of each episode. Training environment 1: 6 random obstacles; Training environment 2: 7 random obstacles; Training environment 3: 8 random obstacles, all with random stair height of 0.04, 0.06 and 0.08 m.

TABLE I: Ablation study of the hierarchical framework.

Scenario	Default					
Framework	Ours	Fixed (w/o hierarchy)				
		2	4	6	8	10
Time(s)	43.1	56.2	42.7	41.3	40.8	49.3
Battery(J)	3090	3006	4572	6633	8380	13200
Success Rate(%)	85.7	72.5	78.5	75.0	67.6	58.8 (0.04)
	90.2	0	71.2	77.8	71.8	63.5 (0.06)
	92.1	0	62.0	68.1	75.9	59.1 (0.08)
	89.3	24.2	70.6	73.6	71.8	60.5 (AVG)
$F(\mathcal{H})$	1.136	-0.284	0.598	0.465	0.258	-0.520

TABLE II: Performance comparison of different algorithms.

Scenario	Default		7 random obstacles	8 random obstacles
Policy	TD3	SAC	TD3	TD3
Time(s)	43.1	57.6	50.2	66.4
Battery(J)	3090	3987	3564	4455
Success Rate(%)	85.7	2.28	65.2	47.3 (0.04)
	90.2	5.60	67.9	44.3 (0.06)
	92.1	1.83	73.1	38.1 (0.08)
	89.3	3.24	68.7	43.2 (AVG)
$F(\mathcal{H})$	1.136	-0.813	0.590	-0.135

For our hierarchical framework, two-module, four-module and six-module configurations are selected in the case of random stair height of 0.04, 0.06 and 0.08 m, respectively. After traversing the stair terrain, the model selects a two-module configuration to complete subsequent tasks. The two-module robot performs well through stairs of 0.04 height, but not through 0.06 and 0.08 height. When the four-module robot was selected to climb the 0.06 stair height using the hierarchical framework, the success rate of the final task was about 90.2%, and the success rate without the hierarchical framework was 71.2%. It can be seen from Table I that our hierarchical framework performs better than other algorithmic models in terms of success rate, energy consumption and time efficiency. And when the robot climbs the stairs, in terms of the success rate alone, the more modules a robot has, the higher possibility it can climb up the stairs, while energy consumption will also increase. It can be seen from Table II that when the SAC algorithm is used in the motion model, the motion model often outputs extreme values to control the boundary motion of the robot, which is relatively rare in the model of the SAC algorithm. In addition, the actions output by the SAC algorithm are relatively jumpy. For example, the first move is 0.298, and the next move might be 0.401. This does not apply to robot control commands that require continuity.

We can observe that the number of modules has an impact on the robot's motion strategy training. Firstly, during the stair climbing task, we observe that the more number of robot modules, the easier the robot's motion strategy can be trained. When the number of robot modules is large, the robot is less prone to roll and turn over during stair climbing and more likely to obtain a larger reward value, i.e., $\tau_1 \cdot \Delta D$ as the first component of R_1 in Equation (6); when the robot has a small number of modules, it is prone to roll over and spend more time steps on the stair climbing task; when the number of robot modules is less, its motion strategy is more difficult to train, such as a two-module robot with a fixed configuration climbing 0.06 step height. Secondly, in the obstacle avoidance task, the more modules the robot has, the more difficult it is for the robot to make turns and have a higher probability of colliding with the obstacle. Although the robot has distance sensors on each side of each module, it is still difficult to train a good motion strategy to complete the obstacle avoidance task.

We further conducted a set of mixed experiments that differed from Env 1 in that we placed some obstacles on the stairs. The robot's configuration choice was almost identical to that of Env 1. However, the final success rate was low, with an average success rate of about 25%, and we believe that it is difficult for the modular robot to climb stairs while avoiding obstacles. In the future, when coping with mixed scenarios, we can consider splitting the modular robot into multiple agents (with shorter length) to be more flexible and trade-off between different tasks, and they can re-construct into fewer number of robots to complete the subsequent tasks.

Based on the above discussion, the experimental results show that compared with other methods, the proposed framework can obtain high-quality optimized configuration and motion strategies for modular robots in multi-task scenarios.

IV. Conclusions

In this work we proposed a co-optimization framework based on hierarchical deep reinforcement learning, to con-

currently generate optimized morphology and behavior of modular robots to enhance their adaptability in multi-task scenarios. The framework consists of a configuration model and a motion model based on the TD3 algorithm, which update asynchronously with a shared reward. To reduce the dimension and complexity of the motion space, we predefined some motion modes and optimize different groups of parameters in the motion model to adapt to various task scenarios. The proposed framework was demonstrated on the Webots platform, and preliminary results show that it outperforms state-of-the-art methods in terms of success rate, time and energy consumption. In the future, we consider incorporating more complexity to configuration and motion spaces, as well as more diversity in the multi-task scenarios, to facilitate the application of modular robots to real world tasks.

ACKNOWLEDGMENTS

This work was sponsored by the National Natural Science Foundation of China (NSFC) through grants No. 62103163 and No. 62003055, Natural Science Foundation of Jilin Province through grant No. YDZJ202101ZYTS033, and Natural Science Foundation of Shanghai through grant No. 22ZR1479600. We thank the above mentioned funds for their financial support.

REFERENCES

[1] J. Seo, J. Paik, and M. Yim, "Modular reconfigurable robotics," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 63–88, 2019.

[2] K. Gilpin and D. Rus, "Modular robot systems," *IEEE robotics & automation magazine*, vol. 17, no. 3, pp. 38–55, 2010.

[3] M. Yao, C. H. Belke, H. Cui, and J. Paik, "A reconfiguration strategy for modular robots using origami folding," *The International Journal of Robotics Research*, vol. 38, no. 1, pp. 73–89, 2019.

[4] M. Yao, X. Xiao, Y. Tian, H. Cui, and J. Paik, "An actuation fault tolerance approach to reconfiguration planning of modular self-folding robots," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8274–8280, IEEE, 2020.

[5] M. Yao, X. Xiao, C. H. Belke, H. Cui, and J. Paik, "Optimal Distribution of Active Modules in Reconfiguration Planning of Modular Robots," *Journal of Mechanisms and Robotics*, vol. 11, no. 1, 2018. 011017.

[6] M. Yao, H. Cui, X. Xiao, C. H. Belke, and J. Paik, "Towards peak torque minimization for modular self-folding robots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7975–7982, 2018.

[7] J. E. Auerbach, A. Concordel, P. M. Kornatowski, and D. Floreano, "Inquiry-based learning with robogen: An open-source software and hardware platform for robotics and artificial intelligence," *IEEE Transactions on Learning Technologies*, vol. 12, no. 3, pp. 356–369, 2018.

[8] J. Whitman, R. Bhirangi, M. Travers, and H. Choset, "Modular robot design synthesis with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 10418–10425, 2020.

[9] H. Wei, Y. Chen, J. Tan, and T. Wang, "Sambot: A self-assembly modular robot system," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 4, pp. 745–757, 2010.

[10] W. Tan, H. Wei, and B. Yang, "Sambotii: a new self-assembly modular robot platform based on sambot," *Applied Sciences*, vol. 8, no. 10, p. 1719, 2018.

[11] M. Yim, "New locomotion gaits," in *Proceedings of the 1994 IEEE International conference on Robotics and Automation*, pp. 2508–2514, IEEE, 1994.

[12] M. Yim, D. G. Duff, and K. D. Roufas, "Polybot: a modular reconfigurable robot," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 1, pp. 514–520, IEEE, 2000.

[13] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans, "Modular reconfigurable robots in space applications," *Autonomous Robots*, vol. 14, no. 2, pp. 225–237, 2003.

[14] W.-M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein, and J. Venkatesh, "Multimode locomotion via superbot reconfigurable robots," *Autonomous Robots*, vol. 20, no. 2, pp. 165–177, 2006.

[15] W.-M. Shen, H. C. Chiu, M. Rubenstein, and B. Salemi, "Rolling and climbing by the multifunctional superbot reconfigurable robotic system," in *AIP Conference Proceedings*, vol. 969, pp. 839–848, American Institute of Physics, 2008.

[16] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *The International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 403–421, 2008.

[17] S. Castro, S. Koehler, and H. Kress-Gazit, "High-level control of modular robots," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3120–3125, IEEE, 2011.

[18] R. J. Alattas, S. Patel, and T. M. Sobh, "Evolutionary modular robotics: Survey and analysis," *Journal of Intelligent & Robotic Systems*, vol. 95, no. 3, pp. 815–828, 2019.

[19] S. G. R Prabhu, R. Seals, P. Kyberd, and J. Wetherall, "A survey on evolutionary-aided design in robotics," *Robotica*, vol. 36, no. 12, pp. 1804–1821, 2018.

[20] A. Kamimura, H. Kurokawa, E. Toshida, K. Tomita, S. Murata, and S. Kokaji, "Automatic locomotion pattern generation for modular robots," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 1, pp. 714–720, IEEE, 2003.

[21] D. Ko and H. Cheng, "Reconfigurable software for reconfigurable modular robots," in *Proceedings of ICRA 2010 Workshop Modular Robots: State of the Art*, p. 100, 2010.

[22] Y. Zhu, J. Zhao, X. Cui, X. Wang, S. Tang, X. Zhang, and J. Yin, "Design and implementation of ubot: A modular self-reconfigurable robot," in *2013 IEEE International Conference on Mechatronics and Automation*, pp. 1217–1222, IEEE, 2013.

[23] A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata, "Distributed adaptive locomotion by a modular robotic system, m-tran ii," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2370–2377, IEEE, 2004.

[24] S. Pouya, J. Van Den Kieboom, A. Spröwitz, and A. J. Ijspeert, "Automatic gait generation in modular robots:"to oscillate or to rotate; that is the question"," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 514–520, IEEE, 2010.

[25] T. Mori, Y. Nakamura, M.-a. Sato, and S. Ishii, "Reinforcement learning for cpg-driven biped robot," in *AAAI*, vol. 4, pp. 623–630, 2004.

[26] M. Pitchai, X. Xiong, M. Thor, P. Billeschou, P. L. Mailänder, B. Leung, T. Kulvicius, and P. Manoonpong, "Cpg driven rbf network control with reinforcement learning for gait optimization of a dung beetle-like robot," in *International Conference on Artificial Neural Networks*, pp. 698–710, Springer, 2019.

[27] S. Ha, J. Kim, and K. Yamane, "Automated deep reinforcement learning environment for hardware of a modular legged robot," in *2018 15th International Conference on Ubiquitous Robots (UR)*, pp. 348–354, IEEE, 2018.

[28] E. Zardini, D. Zappetti, D. Zambrano, G. Iacca, and D. Floreano, "Seeking quality diversity in evolutionary co-design of morphology and control of soft tensegrity modular robots," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 189–197, 2021.

[29] K. P. Cheng, R. E. Mohan, N. H. K. Nhan, and A. V. Le, "Multi-objective genetic algorithm-based autonomous path planning for hinged-tetro reconfigurable tiling robot," *IEEE Access*, vol. 8, pp. 121267–121284, 2020.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[31] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approxi-

mation error in actor-critic methods," in *International Conference on Machine Learning*, pp. 1587–1596, PMLR, 2018.

[32] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*, pp. 387–395, PMLR, 2014.

[33] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. J. Ijspeert, "Exploring adaptive locomotion with yamor, a novel autonomous modular robot with bluetooth interface," *Industrial Robot: An International Journal*, 2006.

[34] T. Tosun, J. Davey, C. Liu, and M. Yim, "Design and characterization of the ep-face connector," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 45–51, IEEE, 2016.

[35] 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*, pp. 1861–1870, PMLR, 2018.