Evolution-Based Shape and Behavior Co-Design of Virtual Agents

Zhiquan Wang, Bedrich Benes, Ahmed H. Qureshi, and Christos Mousas

***Abstract*—We introduce a novel co-design method for autonomous moving agents' shape attributes and locomotion by combining deep reinforcement learning and evolution with user control. Our main inspiration comes from evolution, which has led to wide variability and adaptation in Nature and has significantly improved design and behavior simultaneously. Our method takes an input agent with optional user-defined constraints, such as leg parts that should not evolve or are only within the allowed ranges of changes. It uses physics-based simulation to determine its locomotion and finds a behavior policy for the input design that is used as a baseline for comparison. The agent is randomly modified within the allowed ranges, creating a new generation of several hundred agents. The generation is trained by transferring the previous policy, which significantly speeds up the training. The best-performing agents are selected, and a new generation is formed using their crossover and mutations. The next generations are then trained until satisfactory results are reached. We show a wide variety of evolved agents, and our results show that even with only 10% of allowed changes, the overall performance of the evolved agents improves by 50%. If more significant changes to the initial design are allowed, our experiments' performance will improve even more to 150%. Our method significantly improved motion tasks without changing body structures, and it does not require considerable computation resources as it works on a single GPU and provides results by training thousands of agents within 30 minutes.**

***Index Terms*—Evolutionary algorithms, physics-based simulation, reinforcement learning.**

I. Introduction

AUTHORING autonomous moving agents is a significant open problem with applications ranging from robotics to animation [50]. Their manual creation and motion design offer a high level of control but do not scale and are error-prone. Automatic generation does not always lead to the desired morphology and topology. Moreover, having the agents react to the environment requires the design of behavioral policies. Recent approaches focused on the automatic design of behavior policies, and advances have been achieved with the help of deep reinforcement learning (DeepRL) combined with motion simulation and fine-designed reward/objective function in physics-based environments [21], [44], [45].

Manuscript received 21 November 2023; revised 8 January 2024; accepted 15 January 2024. Date of publication 18 January 2024; date of current version 29 October 2024. Recommended for acceptance by L. Wang. *(Corresponding author: Zhiquan Wang.)*

The authors are with Purdue University, West Lafayette, IN 47907 USA (e-mail: wang4490@purdue.edu; bbenes@purdue.edu; qureshi7@purdue.edu; cmousas@purdue.edu).

Digital Object Identifier 10.1109/TVCG.2024.3355745

While a large body of related work has addressed virtual agent behavior and control policy design, the *co-design* of a virtual agent shape and its corresponding control policy is an open research problem. While structural and behavioral co-design is the natural way for living forms, it is a challenging computational problem because the search space is ample. The changes in the agent's configuration may cause the original control method to diverge from the expected motion. Existing algorithms optimizing the agent and its controller either use simple configurations (e.g., 2D space, voxels) [3] or often lead to structures that deviate considerably from the initial design. It is also essential to balance the optimized and the initial structure, as uncontrolled optimization may lead to a significantly different shape from the user's expectations. At the same time, a good way would be to allow body parts to be added or removed via evolution. Our work shows that even subtle changes to the initial design can significantly increase performance: "a slow agent with better legs will run faster".

Our first key observation comes from evolutionary algorithms that address the wide variability of forms and their adaptation [40]. Moreover, recent progress in DeepRL has introduced ways to learn a single, universal behavior policy for a wide range of physical structures resulting in a smaller memory footprint and efficient behavior learning in large-scale settings [15]. Therefore, using universal DeepRL frameworks has the potential to provide an efficient way to explore the ample solution space and design evolution-based methods simultaneously. Our second key observation comes from the high variation the evolutionary design often causes. This is undesirable, and user constraints over how the agents evolve can significantly guide the agent's shape and prune the search space. Our third observation supports discrete morphological changes potentially indicating the evolution preference during the structure optimization. Suppose a body part becomes significantly shorter during evolution. The algorithm will explore removing the part entirely, as it may not be needed to aid the overall goal.

We introduce a novel evolution-based algorithm that co-designs the 3D physical parameters of an agent and its corresponding behavior within a user-defined boundary. Our work aims to co-design various agents with similar physics attributes within the range of user inputs and a universal controller to walk in the given environment. The user input defines the range of the body part's length, radius, and range of joints' angles affecting the agents' kinematic and physics attributes. The evolution-based method creates new agents based on the user-given template agent and optimizes their performance. For

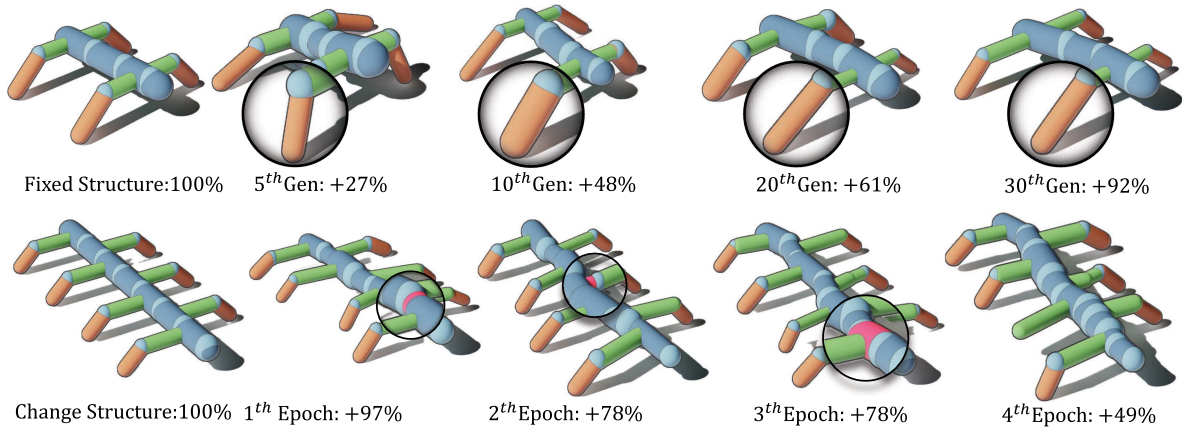

Fig. 1. User-created agent is optimized via evolution (top). The structure is fixed, but its attributes (body part length, width, mass, initial pose, joint range, stiffness, and damping) can change. Even small changes in the attributes lead to significant improvement in performance, as shown in the zoomed-in parts. The performance (see Section V-C) of the baseline agent is 100%, and it improves through the evolution to 27, 48, 61, and 92%. Agent's performance can be improved without significantly altering the original design, even with user constraints. The bottom row shows the optimization when structural changes are allowed, and the evolution will attempt to add or delete body parts. The performance improves through the evolution to 97%, 78%, 78%, and 49% through four epochs of optimization. Each epoch of change structure evolution contains 30 generations of fixed structure evolutions.

each generation, we first train a policy net with Proximal Policy Optimization (PPO) to control agents' motion. Our method builds on the recent work of Gupta et al. [15] that allows for learning a universal controller over a modular robot design space with different structures. We designed a universal policy based on Multiple Layer Perceptron (MLP) that controls all the agents with the same topological structure with a single deep neural network that trains faster on multiple agents in one generation and has a strong generalization ability over generations. After the training phase, we create a new generation by selecting high-performing agents and merging their attributes represented as genes. Through this evolution, we produced agents with higher performance in several generations. The user controls what and how much can be modified through evolution, leading to agents that vary slightly from the original design but achieve significantly better performance (tens to hundreds of percent). An example in Fig. 1 shows the original design (a). When the body modifications are not allowed, our algorithm evolves a new, better-performing agent (b). Enabling the body modifications improves the performance even more (c), and allowing mutations causes more significant alterations to the original design, increasing the performance to 228% (d). The same agent evolved while its body shape was fixed, as shown in (g-k).

Contributions: 1) An evolution-based optimization that produces agent attributes that hold the design requirement and fit the given task. 2) Inspired by Metamorph [15], we train various agents with a single universal policy and expand it with the behavior inherited from the pre-trained model. 3) User control over the allowable agent's modifications in terms of parameter ranges of allowed values. 4) A training pipeline to allow the evolution of agents with different structures.

II. Related Work

We related our work to procedural modeling, physics-based animation, (deep) reinforcement learning for agent motion synthesis, and co-designing structure and behavior.

Early Physics-Based and Procedural Models: Procedural approaches generate a model by executing a code, and the procedural rules and their parameters define a class of generated objects. Procedural animation generates animation sequences that provide a diverse series of actions that could otherwise be created using predefined motion clips. A seminal example is the work of Reynolds [42], who introduced a reactive control [22] of procedural agents that faithfully recreated complex motion patterns of flocks of birds and schools of fish. Physics-based animation represents the agents as interconnected rigid bodies with mass and moment of inertia controlled by joint torques or muscle models [55]. As the control mechanism of an agent significantly affects the motion quality, the choice of control method is important depending on the task. Peng and van de Panne [39] compared the difference across torque control, PD controller, and muscle base control. Many methods work on the control policy to synthesize realistic locomotion. One approach utilizes motion equations or implicit constraints to optimize the locomotion-generated physics-based gaits by numerically integrating equations of motion [41]. Van de Panne et al. [51] developed a periodic control method with cyclic control graph [32] that applies a contact-invariant optimization to produce symmetry and periodicity fully automatically. Bi-pedal creatures were optimized by controlling their muscles in [11]. The design of a physics-based controller remains challenging as it relies on the appropriate design of the agent and the task-specific objective functions assigned to it.

Annother approaches learn to synthesize motions from a **motion dataset** or reference motion clips [6], [28], [29], [54], [55], [58]. For example, the real-time interactive controller based on human motion data that predicts the forces in a short window has been used in [8] and the simulation of a 3D full-body biped locomotion by modulating continuously and seamlessly trajectory in [27]. Wampler et al. [52] applied joint inverse optimization to learn the motion style from the database.

Machine Learning: The seminal works of Sims [47], [48], [49] uses genetic algorithms [23], [25] to evolve 3D creatures

by using physics-based simulation, neural networks, genetic algorithms, and competition. These works were one of the key inspirations for our approach.

Probably the first works to apply machine learning to control locomotion were by Grzeszczuk et al. [13], [14] who used neural networks to learn the motion of fish. DeepRL provides an agent's control policy automatically, and it has been proven effective in diverse and challenging tasks, such as using a finite state machine (FSM) to guide the learning target of RL and drive a 2D biped walking on different terrains [35]. Yu and Turk et al. [59] encouraged low-energy and symmetric motions in loss functions, and Abdolhosseini and Ling et al. [1] address the symmetry from the structure of policy network, data duplication, and loss function and they also handle different types of robots or terrains. One of the drawbacks is the loss of direct control of the target motion because the reward function does not provide explicit motion features.

Combining DeepRL with motion data has the potential to address this issue by giving an imitation target. With the assistance of motion reference, the learning process can discard massive, meaningless motion and dramatically reduce the exploration of action space. Peng and Abbeel el at. [34] enabled learning challenging motion tasks by imitating motion data or video frames directly [37]. Won and Lee [56] handle shape variations of a virtual character. However, learning from the unstructured motion database or motion reference with inaccuracies can make learning the policy difficult. A fully automated approach based on adversarial imitation learning to address this problem by generating new motion clips was introduced in [38]. Peng et al. [36] combined adversarial imitation learning and unsupervised RL techniques to develop skill embeddings that produce life-like behaviors for virtual characters. The characters learn versatile and reusable physically simulated skills.

Co-optimizing design and behavior attempts to find behavior policy and shape simultaneously. Evolution has been used to design the shape of robots [4], [17], [18], [30], and neural graph evolution has been applied to their design [53]. Our work is inspired by the recent work (RoboGrammar) [60] that uses graph search to optimize procedural robots for locomotion on various terrains. RoboGrammar uses a set of well-tuned fixed body attributes (length, density, control parameters), while our method evolves the body attributes of the virtual agents. Lee et al. [26] combined body structures from different candidate agents to keep the motion style. Others focused on motion style transfer from different morphologies [2]. Ha et al. [19] focused on the co-design of the shape and the function of robotics limbs, and the same authors used implicit function theorem to co-design the shape and function of robots [20]. Digumarti et al. [9] designed legged robots optimized for locomotion, and others focus on hand grasping [33], search on terrains [57], or agent construction [43]. Close to our work is [3], which uses co-design via evolution to co-optimize the design and control of 2D grid-based soft robots. This method works in 2D on a fixed set of agent parts and trains each agent individually, while our approach uses group training that significantly shortens training. This is inspired by the works [15], [24], which controls different agents with one universal controller. We designed our universal controller with an MLP network instead of the self-attention layer as it is faster than a Transformer or GNN to train and provides results in minutes on a single GPU but the same performance. Our controller handles agents with the same topology but different body attributes. Gupta et al. [16] evolve the agent's structure by mutations and sampling without merging the parents' genes to reproduce the children. It does not provide control over the agent's design during evolution.

III. Overview

The input to our method (see Fig. 2(a)) is an agent that was either provided by the user or generated randomly. The agent has its body parts with mass and connections with defined motion. The user can also define constraints that guide the changes in the agent form. Examples of the constraints (marked schematically as yellow arrows) are the ranges of the allowed changes in the length of the body, the width of the legs, etc. Our method improves the performance of the physically simulated agent within the constraints via evolution and ensures the result does not deviate from users' expectations. The constraints do not need to be tuned carefully, and their ranges could be small to ensure the visual similarity between the original and optimized designs.

The input agent is trained (Fig. 2(b)) by the PPO in a physics-based environment as a simulated robot with a rigid body, collision detection, shape, and motors to perform a task. The output of this training is used as a baseline for evaluating the performance of the following stages of the algorithm. The learned policy is transferred into the agent's generation (Fig. 2(d)) as a start policy that accelerates the following generations' training with encoded motion prior.

The algorithm then enters into the co-design phase of evolution (Fig. 2(c)-(f)). It creates several hundreds of variants by randomly sampling the allowed ranges of the parameters of the input agent (Fig. 2(c)). This new generation of agents is trained with the universal PPO, which significantly accelerates the training time and allows training on a single GPU. The trained agents are sorted according to their fitness, and the top-performing agents are selected (Fig. 2(e)). The selected agents undergo crossover and mutation to generate a new generation (Fig. 2(f)), and the new generation is trained by bootstrapping with the policy from the parent generation. During the evolution, the agent keeps improving its attributes, and the algorithm stops when the improvement is insignificant, or the user decides the output is satisfactory.

IV. Agent Description

Our agent description can be used in DeepRL frameworks, supports physics-based simulation, and allows for a fast definition or user constraints.

A. Shape

The agent (see Fig. 3) is a directed acyclic graph $\mathcal{G} = \{V, E\}$ with vertices $v_i \in V$ and edges $e_{i,j} : v_i \rightarrow v_j$. Each v_j corresponds to a node that connects different parts of the agents and

Fig. 2. **Overview:** The input agent is either generated randomly or by the user, and the user can also define constraints (yellow arrows) (a). The initial Proximal Policy Optimization (PPO) trains the input agent to provide baseline agent policy (b). The algorithm then creates variants of the initial model (c) and trains them with the universal PPO (d). Selection (e), crossover, and mutation (f) create a new generation trained again. The system outputs the best(s) co-designed agents and their policies (g).

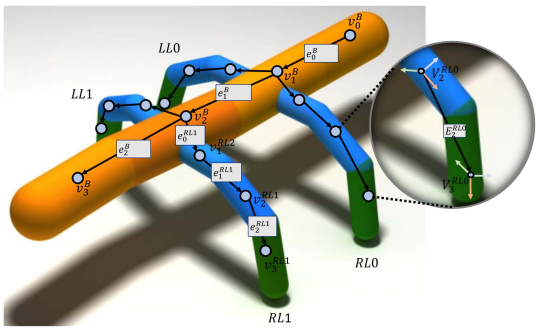

Fig. 3. Example of an agent, its corresponding topological graph, and the coordinate systems of the joints (inset).

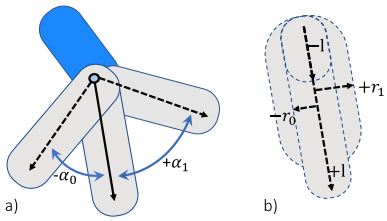

Fig. 4. User-defined constraints.

e_{ij} is a joint that connects two parts (nodes v_i and v_j) of the agent's body.

Each agent consists of two building blocks: body parts are denoted by the upper index B, and legs with the foot are denoted by LL and RL for the left and right leg, respectively. The acyclic graph is a tree with the root always being the node v_0^B. An example in Fig. 3 shows an agent with two pairs of legs and a body with four body parts. An additional index distinguishes each leg, e.g., the third vertex on the second left leg from the torso has index v_1^{LL2} (indexed from zero).

While the topology of the agent is described by the graph $\mathcal{G}$, the geometry is captured by additional data stored in each graph vertex v that is called agent's *attributes*. Each body part is represented as a generalized cylinder (a capsule), and we store its local coordinate system, orientation, radius, and length. The edges also store the rotation axis and rotation range. The *user constraints* (see Fig. 4(b)) are defined as the ranges of motion, length, radius, etc. Note that the ranges may be asymmetrical (see Fig. 4(a)). A global constraint defines how much evolution can change the attributes as a whole.

B. Physics Simulation and Movement

The physics of the motion of each agent is simulated with rigid body dynamics. In addition to the geometric attributes, each edge e also stores physics attributes: stiffness, damping, friction, and mass density. Each body part also stores its mass, derived from the density and volume. The movement simulation uses the Isaac Gym [31], which runs parallel physics simulation with multiple environments on GPU. The agent's topology, geometry, and attributes are stored as an MJCF file interpreted by the Isaac Gym. The simulation engine has various parameters, of which the most notable is the agent's collision with the environment and self-collision that were enabled in our experiments. Enabling self-collision detection slows the simulation significantly. We perform a self-collision check for all initial designs and discard self-colliding agents. The range of changes our simulation allows is relatively small (shown in Section VII-B). Although minor collisions can occur, we do not check for them to keep the simulation fast.

The agent's movement is given by the torque τ applied to each joint over time. There are two methods to control the joint of an agent. The first option (*direct control*) applies the torque directly to each joint, and the actual torque value is provided by the policy network described in Section V. The torque control is fast but can be noisy and unstable as the torque is sampled from a policy-given distribution. The second option uses Proportional Derivative (PD) controller that works as an intermediate between the control policy and the torque. The control policy specifies the target position for the joint, and the PD provides the torque. This control method provides stable motion as the PD controller can reduce the motion jittering. We use both options in our method and refer to them as *direct torque control* and *PD*.

C. Generation

We generate the agents either manually or randomly. The manual description is performed by writing the agent description into a text file that is then visualized, and the motion is displayed. This is a tedious trial-and-error process. The random generation creates the description automatically in a two-step process that starts by generating body parts and then attaching legs. The random generation may lead to non-realistic configurations, such as legs inside the body. We test each agent by detecting these configurations before training and visually verifying them for consistency.

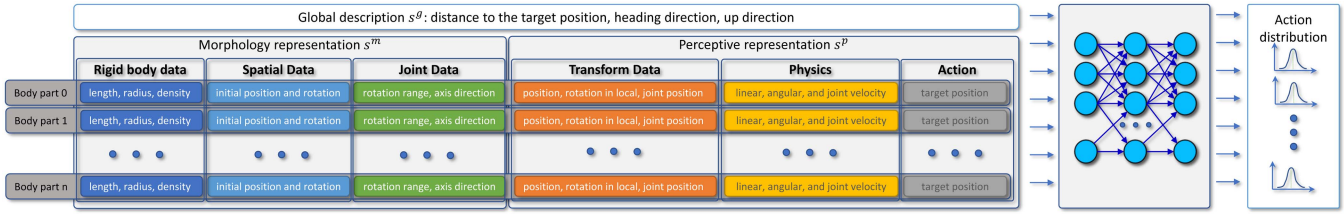

Fig. 5. Control Policy Network (Actor in PPO) of a single agent consisting of multiple body parts. The body part properties and the global description are processed by a deep neural network that generates the corresponding action.

V. DeepRL Model Representation

The DeepRL generates a control policy that produces the agent's locomotion. The learned control policy should be robust across the entire generation. Moreover, we need to train a large number of agents, so the control policy should be able to train rapidly.

The agent's description and attribute values become the DeepRL framework *states* optimized towards the desired behavior. We use Proximal Policy Optimization (PPO), an Actor-Critic-based RL algorithm [46]. The Critic estimates the value function and acts as a baseline for the computation of advantages, while the Actor updates the policy distribution in the suggested direction. The controller is trained with PPO with advantages computed with Generalized Advantage Estimation (λ) [45]. The controller receives the state of an agent $s(t)$ at the time t, and it outputs an action $a(t)$ for each joint that leads to the state $s(t+\Delta t)$. The action $a(t)$ is either the torque τ applied directly to each joint or a position of a PD controller that then computes the required torque (Section IV-B).

A. States and Actions

The state of the agent s_t at time t is (see also Fig. 5):

$$s_t = (s_t^m, s_t^p, s_t^g), \tag{1}$$

where s_t^m is the agent's morphology, s_t^p denotes the perceptive representation, and the global representation is denoted by s_t^g. We will not specify the time t unless needed in the following text.

The *morphology representation* s^m consists of

$$s_t^m = (s_{rigidbody}, s_{spatial}, s_{joint}),$$

where $s_{rigidbody}$ includes the physics attributes of a body: length, radius, and density. The spatial data $s_{spatial}$ includes the initial heading direction of the body computed from $fromto$ attributes in an MJCF file and the initial local position. The values of s_{joint} contain the attributes of the joints attached to the body, such as the rotation axis and the rotation range of the joint. The morphology representation s^m does not change during the simulation and training, and it changes only after evolution when the new generation is generated (Section VI). The network can decide on different agents based on their morphology attributes because this part is a constant input to the policy network.

The *perceptive representation* s_t^p stores the dynamics information that changes at each time step t

$$s_t^p = (s_{transform}, s_{physics}, s_{act}),$$

where the transform attributes $s_{transform}$ include the local position, local rotation represented as a quaternion, and the joint position. The physics attributes $s_{physics}$ include linear, angular, and joint velocity. Actions from the previous time step of each joint are also used. The last parameter is the action s_{act} that specifies the target position of the PD controller or direct torques for each joint. The actual value of actions is sampled from Gaussian distributions given by a control policy. We use hinge joints for each agent, specified as the 1D rotation angle q, normalized based on their joint rotation ranges.

Finally, the *global description* s^g contains information that indicates the overall behavior of the agent, i.e., distance to the target point, heading direction, and the up vector of the torso.

B. Network Architecture

The Actor and the Critic in the PPO algorithm are modeled with a deep neural network (see Fig 5). The Actor-network is a control policy π that maps the given state s to the Gaussian distributions over actions $\pi(a|s) = \mathcal{N}(\mu(s), \Sigma)$, which takes a mean $\mu(s)$ from the output of the deep neural network and a diagonal covariance matrix Σ specified by a vector of learnable parameters σ. The mean is specified by a fully connected network with three hidden layers with sizes [256,128,64] and the Exponential Linear Unit (ELU) [7] as the activation function, followed by a linear layer as the output. The covariance matrix values $\Sigma = diag(\sigma_0, \sigma_1, \ldots, \sigma_n)$ are learnable parameters and are updated as part of the deep neural network with gradient descent. The Critic network $V(s(t))$ is modeled as a separate network with the same architecture as the Actor-network, except that its output size is one providing the given state value. A fully connected network is beneficial as it provides faster learning and easier transfer learning than transformer-based networks. We implemented both solutions, and the transformer takes around 120 minutes to converge to the optimal policy. In comparison, the fully connected actor policy takes 20 minutes to converge at the initial stage and 2 minutes from a pre-trained model from the ancestors.

C. Rewards

The reward function is designed to produce natural motion that reflects the motion of real animals, e.g., for the agent in Fig. 10, we attempted to set the parameters to resemble a caterpillar's motion. We use the same values of reward terms from [5], [10], [60] to create a fair and standard reward signal for experiments while considering motion aesthetics. The reward r evaluates an agent's performance, e.g., encouraging the agent to walk forward over flat terrain. It attempts to maintain a constant moving speed towards a target distance, and the agent should be able to keep stable locomotion without flipping or deviating from the target direction. It also minimizes energy consumption. The rewards function is a sum of multiple task objectives

$$r = r^p + r^v + r^e + r^a, \tag{2}$$

where r^p is the pose reward that encourages the agent to maintain a stable initial pose during the movement, r^v is the velocity reward, r^e denotes the efficiency reward, and r^a is the alive reward.

The pose reward r^p maintains the heading direction of the agent's body aligned with the target direction (0,1,0) as the agent walks along the y-axis. The up direction of the head should point to the up-axis (0,0,1) to prevent the agent from swinging its body or flipping:

$$r^p = w^{heading} \cdot r^{heading} + w^{up} \cdot r^{up}, \tag{3}$$

and the weights $w^{heading} = 0.5$ and $w^{up} = 0.1$. The heading reward $r^{heading}$ is computed as

$$\begin{aligned} p^{heading} &= \mathbf{heading} \cdot (\mathbf{0}, \mathbf{1}, \mathbf{0}) \\ r^{heading} &= min\left(\frac{p^{heading}}{t^{heading}}, 1\right) \end{aligned} \tag{4}$$

where $p^{heading}$ is the projection of the heading vector of the head to the target direction, $t_{heading} = 0.8$ is the threshold of getting the maximum heading reward. We apply the same equation to the up stable reward r^{up}, except that the aligning vector points up, and we use a different threshold of 0.9 that has been established experimentally.

The velocity reward r^v encourages the agent to move forward along the y-axis

$$r^v = \left(P^y(t) - P^y(t-1)\right)/d_t, \tag{5}$$

where $P^y(t)$ is the walking distance along y-axis at the time step t and $d_t = 1/60\,s$.

The efficient reward r^e encourages the agent to perform energy-efficient actions each time by penalizing high torques of joints close to extreme positions to have smoother locomotion.

$$r^e = w^{act} \cdot r^{act} + w^{power} \cdot r^{power} + w^{jointlimit} \cdot r^{jointlimit}, \tag{6}$$

where the weights are $w^{act} = w^{power} = -0.05$ and $w^{jointlimit} = -0.1$. The action cost

$$r^{act} = \sum_{\forall joint} a^2$$

penalizes high torque action given by the control policy or joint position closer to the range limitation in the PD control. The energy cost

$$r^{power} = \sum_{\forall joints} |a \cdot v|$$

prevents the agent from taking high-energy consumption actions by avoiding high joint velocity v.

The joint-at-limit reward $r^{jointlimit}$ prevents the agent from not utilizing all joints by penalizing the joint stuck at the limit position

$$r^{jointlimit} = w^{jointlimit} \sum_{\forall joint} \begin{cases} 1, & \text{if } p^{joint} > t^{jointlimit} \\ 0, & \text{otherwise} \end{cases}$$

where p_{joint} is the normalized joint position, $t^{jointlimit} = 0.99$ is the threshold to receive the penalty and $w^{jointlimit} = -0.1$ is the weight. The alive reward r^a is set to zero when the agent walks out of the scene's boundaries; otherwise, it is set to one.

We measure the performance of an agent based on the reward function in (2), which is a weighted linear combination of sub-rewards designed for the desired locomotion of the agent. It has the following targets: 1) keeping a stable pose and heading direction without flipping or rolling (3), 2) walking with a target velocity (5), and 3) walking efficiently with fewer energy consumption (6). A higher reward indicates a better performance on these targets. Table A.7, available online, in the Appendix shows the effect of these reward designs.

D. Training

Our control policy is trained with the PPO [46] on GPU-based parallel environment Isaac Gym [31]. The training is performed first for the template input agent (Fig. 2(a)) and then for each generation during the evolution (Fig. 2(d). Both training stages proceed episodically, starting at an initial state s_0 of each agent, which is randomly sampled from a given range to enhance the generalization of the policy network. The experience tuples $(s(t), a(t), r(t), s(t+1))$ are sampled in parallel at each time step t by sampling actions a from control policy π with a given state $s(t)$. The experience tuples are stored in the replay buffer for the training iteration later. Each episode is simulated within a maximum number of steps, or some specific conditions like flipping or walking in the wrong direction can terminate it. After the replay buffer is filled with experience tuples, several training iterations are performed to update the Actor-network (policy network) and the Critic network (value network). The learning rate lr is dynamically adapted to the KL-divergence kl between the new and old policy

$$lr = \begin{cases} \max\left(lr/1.5, lr_{min}\right), & \text{if } kl > \text{desired } 2 \cdot kl \\ \min\left(lr \cdot 1.5, lr_{max}\right), & \text{if } kl > \text{desired } 2 \cdot kl \end{cases} \tag{7}$$

where $lr_{min} = 1e^{-4}$ is the minimum and $lr_{max} = 1e^{-3}$ is the maximum learning rate allowed during the training, and desired kl is a hyper-parameter that controls the update of learning rate based on the distance between old policy and the new policy during policy update iteration.

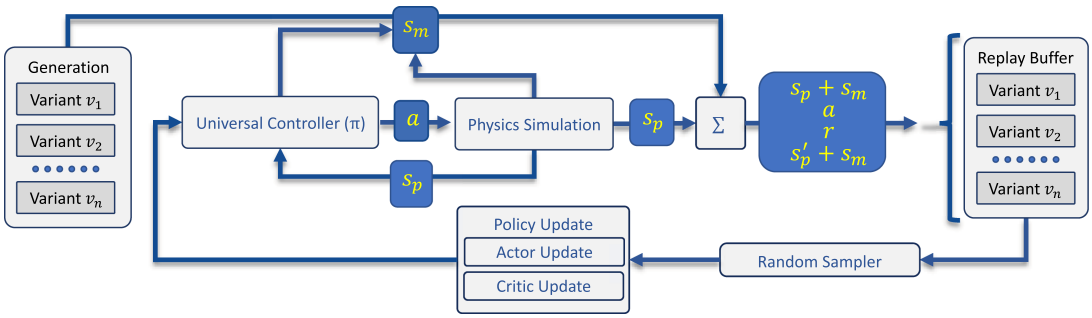

Fig. 6. Grouped agent training pipeline where s_m is the morphology state, s_p the receptive state, a are the actions and r rewards.

Single-agent training: We train the initial (template) agent (Fig. 2(b)) to complete the task until the reward (2) reaches maximum or does not change significantly. The result provides the baseline policy, reward value, and initial locomotion.

Generation Training: The input to the generation training is the template agent policy, which attempts to optimize a whole generation of agents for evolution. Since each generation of agents shares the same structure, the control policy of the template agent is reused via transfer learning. Then, the descendants could quickly inherit the locomotion experience from the previous generation, which, in effect, increases the speed of training (to 20% in our experiments).

The generation includes n variants trained in parallel (Fig. 6) each in its environment. At each time step t, the universal control policy takes the states s of an agent v_i and outputs its actions a. The experiences are sampled and stored in the replay buffer. The experience tuples sampled from different variants are randomly sampled for the policy update phase. This training part is inspired by metamorph [15] that trains a universal controller with a transformer base structure for robots with different morphology. We use a fully connected network in the policy net instead of an attention-base structure because MLP provides the same performance and trains around 8-10$\times$ faster. We tested the transformer [15] that provided similar results, but the training took about two hours instead of 10 minutes by using MLP.

VI. Evolution

A. Fixed Structure Evolution

Each trained group of agents (Fig. 6) produces a set of variants of agents with different body attributes altogether with their reward function. The goal is to choose the best variants of agents and create a new generation while ensuring that their most beneficial traits propagate and possibly improve in the next generation.

Let $V^g = \{v_1^g, v_2^g, \ldots, v_n^g\}$ denote the $g-$th generation with variants of agents v_i. Each agent has a list of attributes att_i that we call its gene. The next generation $g+1$ is produced via selection, crossover, and mutation [12], [25].

Selection: We sort all variants V^g in the actual generation g according to their reward and select the top $p\%$ $(p = 20)$ agent variants. This initial set becomes the seed of the generation V^{g+1}.

Crossover: The seed of the new generation is expanded to the number of variants n by crossover. We take the genes att_i and att_j of two randomly chosen agent variants v_i and v_j, from the seed set. We use a random crossover that takes an attribute $att_i[k]$ and swaps it with $att_j[k]$ with the 50% probability, where k denotes the $k-$th value in a binary gene. This process is repeated until a new generation V^{g+1} with n variants has been created. The attributes that can be evolved during the evolution include radius, length, density, initial position, body rotation, stiffness, damping, max effort, and joint rotation range.

Mutation: Each attribute can be mutated by altering its value by a random value $\pm r$. The overall probability of mutation is set to 1% [12].

The user-defined constraints: The user controls the evolution (Section IV-A) by setting some attributes fixed. These attributes will not be affected by the mutation and crossover. Moreover, the user can also specify the range of values as user-defined constraint limits. Values that would mutate out of these ranges are clipped.

Some attributes can be linked (for example, a pair of symmetric legs or body parts belonging to the same group (torso body)) and will always be treated as a fixed group. When one of them is swapped, the other will be as well. If one value changes, the others will be changed by the same value.

B. Changing Body Structure Evolution

Here, we explore the option of changing the body structure during evolution. The previous section described optimized agents with a fixed structure, and the evolution modified their body attributes. By measuring the difference between the original and the optimized design, we observed that the changes in the length of the different bodies indicate the preference of the evolution path of different agents. These changes imply the agent's convergence to longer or shorter body parts to accomplish the task.

We perform informed structure evolution by adding or deleting some body parts to reach a better structure. Our approach is similar to [16], but we use the information from the previous generation to guide the changes. We then evolve the modified agent using the fixed structure evolution. However, allowing the evolutionary algorithm to modify the structure arbitrarily

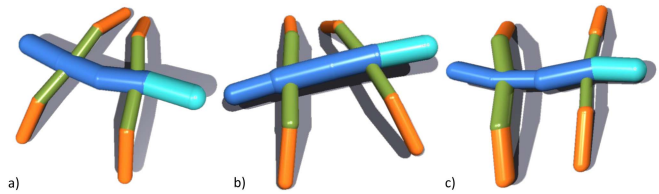

Fig. 7. Baseline agent (a) is evolved by allowing $\pm 10\%$ (b), and $\pm 20\%$ (c) of variance of all its parameters. The reward function value 470 of the baseline agent (a) improves to 132% (b), and 151% (c).

precludes us from using transfer learning between generations, and the optimization must be run from scratch.

Structure Optimization of an agent is modified by either adding (splitting) or removing parts. During the evolution, we evaluate the ratio of length changes of each body part, and we split the part with the greatest positive body change in two. The length of each body will be half of the original length, and the total length will be constant to maintain the appearance of the agent. We delete the body part with the largest negative body change ratio (shrinking). When deleting the part, we maintain the agent topology by correctly reattaching the parts.

Selection: At the first epoch, we perform the fixed structure evolution to produce N optimized agents and select the agent that achieved the highest reward as the candidate for structure optimization. We then split and removed several parts of the winner and trained each from scratch, no matter whether the performance was better or not compared to the previous generation. The best candidate is then used for structure optimization.

Optimization Termination: We stop the optimization if performance decreases or the ratio of changes is lower than a threshold to prevent performance collapse. Since the change of structure will lead to a different performance easily, e.g., missing one leg could lead to an agent's inability to move, it is easy for the agent to fall into a bad scenario for the un-smoothness of the problem.

VII. Implementation and Results

A. Implementation

We use Python to develop the agent generator and all the components in our evolution system. Isaac Gym [31] was used for the physics simulation of the agent, and we implemented the PPO optimization in Python. The neural network is based on Pytorch version 1.8.1. The computation, including deep neural network training, physics simulation, and rendering, runs on a single Nvidia GeForce RTX 3090.

B. Results

1) Fixed Structure Evolution: We test the effect of the evolution on the agent co-design on several manually designed agents (Figs. 7, 10, and 11), randomly generated agents (Fig. 8) and on an optimized design (Fig. 9). All results are summarized in Table I, and details of each body part are in the Appendix. Please note this paper has an accompanying video that shows its results.

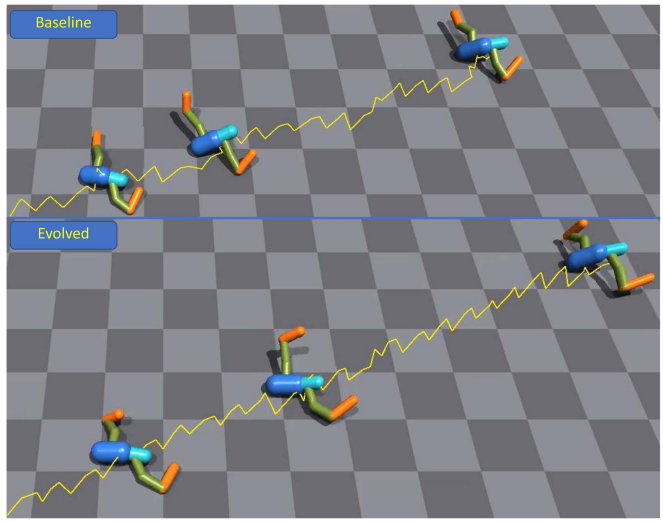

Fig. 8. Simple baseline agent (top) evolved by allowing $\pm 20\%$ of variance of all its parameters. The evolved agents travel a larger distance at the same allotted time, and the evolved reward functions are improved by 489→566 (116%).

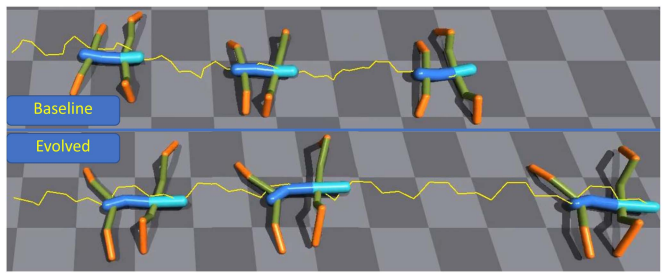

Fig. 9. Medium complex baseline agent (top) evolved by allowing $\pm 20\%$ of variance of its parameters. The evolved reward functions are 572→921 (161%). The original design of this agent is from [60].

TABLE I
Quantitative Results of All Experiments

	Constrained		Reward	New Reward (%)	Change (%)
	Fix	Evo			
Fig. 1 (a)	-	20%	960	100% (Baseline)	N/A
Fig. 1 (b)	No	20%	1,274	132%	0.39%
Fig. 1 (c)	No	20%	1,594	166%	5.82%
Fig. 1 (d)	No	20%	1,775	184%	5.84%
Fig. 1 (e)	No	20%	1,913	199%	5.63%
Fig. 1 (f)	No	20%	2,169	228%	5.72%
Fig. 1 (g)	Body	20%	1,160	121%	0.56%
Fig. 1 (h)	Body	20%	1,842	192%	8.93%
Fig. 1 (i)	Body	20%	2,174	226%	9.40%
Fig. 1 (j)	Body	20%	2,355	245%	9.45%
Fig. 1 (k)	Body	20%	2,428	253%	9.71%
Fig. 7 (a)	-	0%	470	100% (Baseline)	N/A
Fig. 7 (b)	No	10%	621	132%	3.24%
Fig. 7 (c)	No	20%	710	151%	8.75%
Fig. 8 (base)	-	0%	489	100% (Baseline)	N/A
Fig. 8 (evo)	No	20%	566	116%	10.83%
Fig. 9 (base)	-	0%	572	100% (Baseline)	N/A
Fig. 9 (evo)	No	20%	921	161%	8.02%
Fig. 10 (base)	-	0%	683	100% (Baseline)	N/A
Fig. 10 (evo)	No	20%	1,108	155%	2.47%
Fig. 11 (a)	-	0%	683	100% (Baseline)	N/A
Fig. 11 (b)	Torso	40%	1,108	162%	5.24%
Fig. 11 (c)	Leg	40%	870	127	6.44%

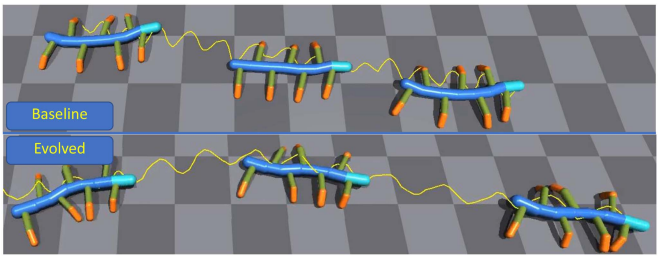

Fig. 10. Complex asymmetric baseline agent (top) evolved by allowing ±20% of variance. The evolved reward functions are 1,118→1,737 (155%).

The first example in Fig. 1 shows the effect of the evolution on the changes and reward function of an agent. The baseline agent is trained to walk with the state-of-the-art PPO training (a), and we then use the evolutionary algorithm to improve its performance while changing its attributes to complete the same task. The reward function value for the baseline agent is 473, and it improves through the evolution after the first generation by 132% (b), the fifth generation 166% (c), 15-th generation 184% (d), 25-th generation 199% (e), and 35-th generation to 228% (f). We then take the same agent and fix its body shape so it cannot change through evolution. The agent is trained from the baseline leading to the new reward after the first generation by 121% (g), the fifth generation 192% (h), 15-th generation 226% (i), 25-th generation 245% (j), and 35-th generation to 253% (k).

The experiment in Fig. 7 studies the effect of globally increasing the range of allowed changes. The baseline input agent was trained, leading to a reward function value 470. We then run the evolutionary co-design, allowing the global change attributes by ±10% and ±20%. While the reward is increasing by 132%, and 151% of the baseline design, the structure of the agent has also changed significantly.

Figs. 8, 9, and 10 show three agents with increasing complexity evolved by allowing ±20% of global attributes changes. The motion snapshots are taken after the same time spent, showing the traveled distance for comparison. The simple agent improved to 153%

Another example in Fig. 11 shows the effect of the restricted control of the evolution. We fixed the torso (Fig. 11(a)) during the evolution by not allowing any changes in the agent. While the body remains the same, the legs and their control were allowed to change by 40%, leading to an improvement of 162%. Fig. 11(b)) shows the same agent where only the torso can evolve, and the legs remain fixed. This limits the motion, and the improvement was only 127% of the baseline.

Our experiments show that small changes in the existing structure parameters can substantially improve the agent's performance. The related work [60] shows 50% improvement on the original design (from 4 to 6 on flat terrain) in their reward functions while optimizing the structure to fit the environment. Our work shows that evolution can achieve up to 100% improvements without changing the structure. We also attempted a wider range of changes, as shown in Fig. 12 where 90% of changes were allowed. Using such large modifications does not allow for efficient sampling of the shape space and quickly leads to degenerated configurations. Allowing modifications of 100% and more led to agents that did not move at all.

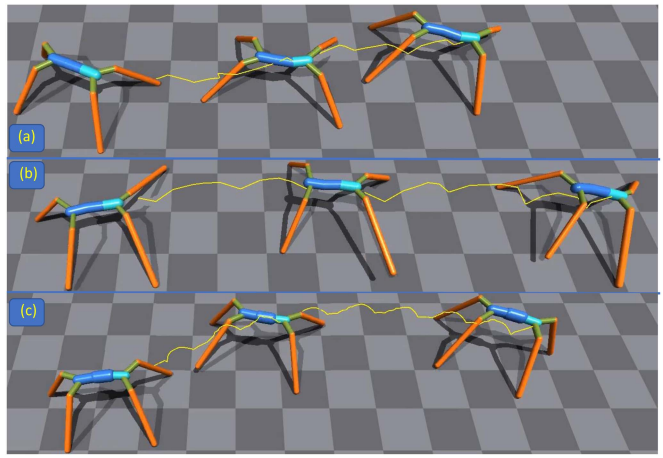

Fig. 11. Agent (a) is evolved with a restricted torso and the allowed changes of 40% to the rest of the body. (b) The legs improved, and the reward function changed 683→1,108 (162%). (c) The last row shows the agent evolved only with allowed modifications to the torso (legs are fixed). The reward function changed 683→870 (127%).

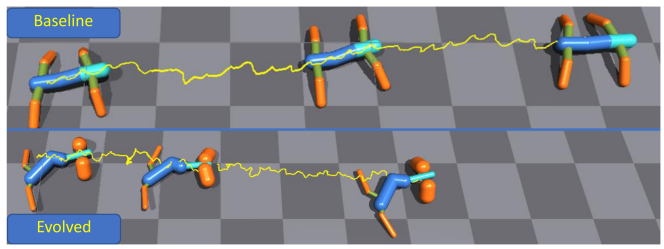

Fig. 12. Allowing ±90% variations produced bad-performing shapes.

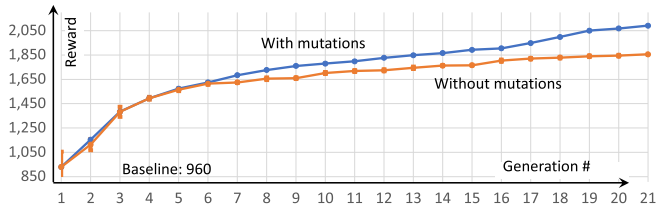

Fig. 13. Agent from Fig. 1 is evolved with and without the mutation, showing that the mutations positively affect the reward function. The line is the mean of multiple tests, and the error bar indicates the standard deviation. Note that the standard deviation is small, and the error quickly becomes negligible.

While the examples mentioned above were generated with the PD control, the accompanying video shows that our evolutionary algorithm handles the direct torque control from the PPO.

We tested the effect of the mutation on the convergence of the reward function. We trained the baseline agent from Fig. 1 with and without the mutation. The progress of both reward functions in Fig. 13 shows that the mutation positively affects the reward function, leading to faster convergence and about 9% higher reward (2,091 vs. 1,856).

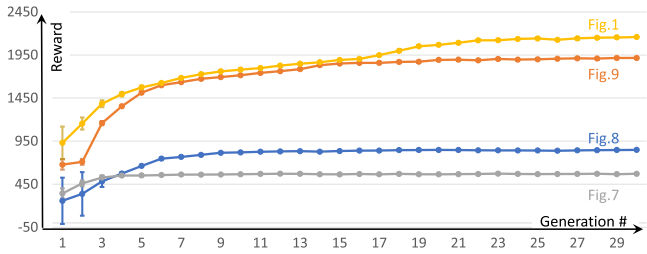

Fig. 14. Reward function evolution for the examples from this paper (mean and standard deviation).

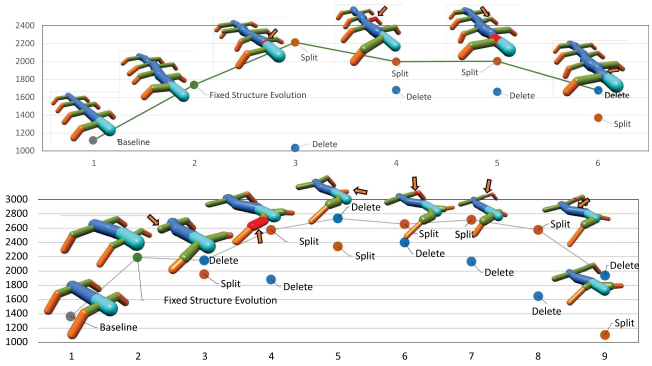

Fig. 15. Changing structure evolution of two agents and the corresponding reward function.

The reward functions of results from this paper through the 30 generations of the evolution are shown in Fig. 14. The reward function increases most if no constraints are imposed on the model or if the model has high complexity, allowing for more changes. The error bar indicates the standard deviation.

We attempted to provide insight into the traits affecting the agents' overall performance. We analyzed the data from the Appendix showing all numbers of the agent changes from Figs. 8, 9, 10, and 11. The tendency to allow the agents to perform better is diminishing their weight. The control parameters are important in the locomotion as its global changes are relatively higher than the others. The statistics show that increasing the body's average length also helps improve performance. This is especially true for the legs, indicating that longer legs are beneficial. Moreover, stiffness and the max effort tend to increase through the evolution as they provide a faster response to the target joint position and increase the maximum torque. An exception is an agent in Fig. 11 that could not evolve its legs, leading to decreased damping and max effort. If the agent is high and unstable, the evolution reduces the torque, which decreases the risk of falling. We also notice that the middle and tail torso often becomes heavier to help maintain stability. If the user-defined constraints fix the torso, the evolution attempts to find different ways to improve efficiency.

The agent generation training with the universal controller is trained for 30 epochs and 150 variants. Each variant runs on six parallel environments. The training for each generation takes around 30 seconds. The overall evolution of the 30 generations takes around 22 minutes, depending on the complexity of the agent and the environment. The main limitation is the size of the GPU memory. An agent takes 5 minutes to be trained, and the total time needed for the optimization without the universal controller and transfer learning used for the optimization would be 375 hours for 30 generations with 150 agents per generation. However, using the universal controller and the transfer learning cuts the time to around 22 minutes (1,125$\times$ faster).

To show the upper bound of our method, we performed two experiments on the agent from Fig. 1 to explore the performance of the universal policy compared to a single agent policy training from scratch. We selected the best design from every five generations (generation 1-35). We performed two experiments for a single controller with equivalent training epochs in our evolution method 1) training without a pre-trained policy with the same iteration 50, where the agent achieved rewards 481,289, 273, 361, 347, 314, 300, 284 and 2) training with the pre-trained policy, and get rewards 1,310, 1,576, 1,732, 1,733, 1,841, 1,883, 1,973, and 2,141. These designs trained with shared controller and pre-trained models achieved 1,274, 1,595, 1,722, 1,776, 1,866, 1,913, 1,992, and 2,190 rewards which are -2.75%, 1.21%, -0.58%,2.48%,1.36%,1.59%,0.96%,2.49% than the single controller showing that our method achieves similar results with smaller performance loss.

2) Changing Structure Evolution: We experiment with agents that changed their structure during evolution. It is not feasible to allow changes for all agents as this requires training them individually from scratch, thus losing the main advantage of the universal controller that allows training hundreds of agents of the same structure with different parameters. We experimented with the changing structure evolution on two agents: 1) a complex asymmetric agent from Fig. 10 and 2) a simple symmetric agent in Fig. 12. After the parameter training, we split the most quickly growing part in two or deleted the most quickly shrinking part. We then optimized the agent from scratch.

The first result in Fig. 15 top shows the evolution preference of an agent with a long torso and multiple legs whose motion is driven by the body's swinging and the movement of the legs. The results show that evolution tends to grow more torso parts to extend flexibility and move faster. The selection decision of the five epochs are split (torso), split (torso), split (torso), split (torso), delete (leg), and the rewards of the selected agents are 1,118 (baseline), 1,737 (fixed evolution), 2,211, 1,996, 2,001, and 1,676. We stopped the experiment when the performance decreased. The performance of the agent increased 27% after the first epoch. It kept a similar performance in the following epochs, which indicates the fixed structure evolution already provides a good design without changing the structure.

The second example in Fig. 15 bottom shows an agent with three body parts and two legs. The results show that the evolution method reduces the total weight by deleting legs to achieve faster movement speed. The agent learns to jump with high swing frequency with an unbalanced pose. The selection decisions of the seven epochs were delete (leg), split (leg), delete (leg), split (leg), split (leg), split (leg), and delete(leg). The rewards of the selected agents are 1,364 (baseline), 2,191 (fixed evolution), 2,140, 2,575, 2,737, 2,717, 2,576, and 1,937. We also stopped the

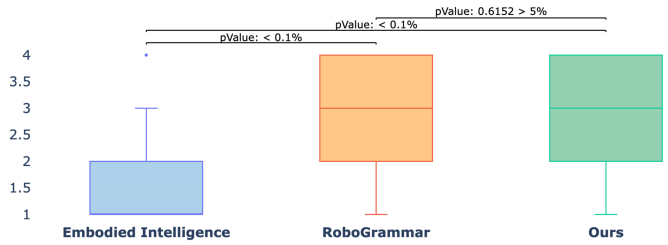

Fig. 16. Statistical comparison of pairs of detection thresholds analyzed with Wilcoxon test.

evolution when the performance decreased. The performance of the agent increased 25% at the fourth epoch with shorter front legs, high leg swinging frequency, and an unbalanced body.

The runtime of the changing structure evolution depends on the epochs it runs,s; the first experiment took 30 minutes per epoch and five epochs (150 minutes total), and the second experiment took around 20 minutes and seven epochs (140 minutes total).

3) User Study on Perceived Realism: Study Design: We performed a user study to evaluate the perceived realism of the generated motion. We have shown a video of a moving agent and asked, "How realistic is the motion of the 3D structure compared to living creatures? ". The options were "Not realistic at all (1)", "Not realistic (2)", "Realistic (3)", and "Very Realistic (4)". The participants selected the answer after the video ended.

Ethical approval: The study was approved by Purdue University's Institutional Review Board IRB-2023-1439.

Setting All videos were represented online, and the participants used their own computers and web browsers.

Participants: We have recruited 21 participants ages 18-55 from undergraduate and graduate students, as well as faculty members population. The user study included 15 males, three females, three self-described, and three prefer not to say. Three participants self-reported having no experience with character animation, three had a low, eight had medium, and four had high experience.

Procedure: The participants were shown nine videos: three from the Embodied Intelligence [16], three from RoboGrammar [60], and three from our results (Figs. 7, 9, and 10). The play time of each video was around 10 seconds, and the sequence of the videos was randomized.

Data collection: All collected were stored in a coma-separated-value file, and the participants were anonymized.

Data Analysis & Result: We ran Wilcoxon tests for the data that violated the normality assumption. Normality was investigated with the Shapiro-Wilk test. The p-values of the Wilcoxon tests between embodied intelligence and our work are $5.217 - 13 < 0.001$, Embodied Intelligence and Robogrammar is $3.2935e - 8 < 0.001$, our work and Robogrammar is $0.6152 > 0.05$. The results show significant differences ($p < 0.05$) between Embodied Intelligence and our work, as well as Embodied Intelligence and RoboGrammar. It shows no significant differences between our work and Robogrammar. The study suggests that the motion generated by our algorithm is perceived as realistic compared to living creatures, it is on par with RoboGrammar and outperforms Embodied Intelligence (Fig. 16).

VIII. Conclusion

We have introduced a novel approach that improves state-of-the-art DeepRL training by adding evolutionary changes to the agent's parameters. While the agent's topology remains the same, the genetic algorithm explores the space of the agent's attributes and attempts to improve its performance to complete the given task. Our approach has two main advantages. First, it allows for user control of the evolving parts. Second, it uses a universal policy and transfer learning that enables us to train a whole generation of agents on a single GPU. This significantly shortens the training time of the evolutionary algorithm to less than one minute per generation during the evolution. We have shown various examples of agents trained with varying shapes and parameters, showing that the performance improved by tens of percent even after just a few generations.

Our approach has several **limitations**. First, we used Isaac Gym and PPO as our simulation and RL training baseline. While this is a suitable choice for comparison, the RL algorithms and physics engine include parameters that need to be tuned, and they may have a negative effect on the training. We have carefully used the same parameters when comparing the results, but we noted, for example, that self-collision detection for complex agents changes the results significantly. The second limitation comes from the choice of the initial agent. If the template agent fails the task, the descendants will not benefit from the pre-trained policy. We use a universal policy for controlling all agents instead of training separate policies per agent. While this speeds up the computation significantly, it likely leads to a sub-optimal policy at the first several generations.

There are many possible avenues for **future work**. First, studying how many and what parameters suit the user would be interesting. We showed several ways of controlling the shape and its evolution, but the actual user intent and feedback would be a worthy research project. Second, the space that needs to be explored during the evolution is vast, and it is evident that our approach is leading only to a limited set of solutions. Future work could use several solutions and see what makes them different. Another important problem is to answer the question of what makes the design perform better. It could be achieved by tracking the values of attributes and seeing how they relate to the performance. However, the relation is very unlikely straightforward, and the parameters may affect each other. Also, when adding a new part, the actual location of the new part is fixed (based on its previous location). It would be interesting to evaluate different positions and their effect on the overall performance. Meanwhile, studying how to transfer the pre-trained model from the ancestors is challenging. As the input and output dimensions change, the policy cannot be fine-tuned based on the pre-trained model. Exploring the transfer learning on different structures can speed up the optimization. An evident future work is studying more complex tasks and environments and allowing topology changes to the body. It would also be interesting to compare it to previous work. However, our approach does not require

significant computing resources, while most of the related work would require significant computing power to generate results for comparison. We could also experience multi-objective scenarios by applying the multi-objective evolutionary algorithm. Our work focuses on the chained multi-legged agents. It would be interesting to show how the same approach works for different configurations of agents.

REFERENCES

[1] F. Abdolhosseini, H. Y. Ling, Z. Xie, X. B. Peng, and M. V. de Panne, "On learning symmetric locomotion," in *Proc. Conf. Motion Interaction Games*, 2019, pp. 1–10.

[2] M. Abdul-Massih, I. Yoo, and B. Benes, "Motion style retargeting to characters with different morphologies," *Comput. Graph. Forum*, vol. 36, no. 6, pp. 86–99, 2017, doi: 10.1111/cgf.12860.

[3] J. Bhatia, H. Jackson, Y. Tian, J. Xu, and W. Matusik, "Evolution GYM: A large-scale benchmark for evolving soft robots," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 2201–2214.

[4] J. C. Bongard, "Evolutionary robotics," *Commun. ACM*, vol. 56, no. 8, pp. 74–83, 2013.

[5] G. Brockman et al., "Openai GYM," 2016, *arXiv:1606.01540*.

[6] N. Chentanez, M. Müller, M. Macklin, V. Makoviychuk, and S. Jeschke, "Physics-based motion capture imitation with deep reinforcement learning," in *Proc. 11th Annu. Int. Conf. Motion Interact. Games*, 2018, pp. 1–10.

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

[8] M. Da Silva, Y. Abe, and J. Popović, "Simulation of human motion data using short-horizon model-predictive control," in *Computer Graphics Forum*. Hoboken, NJ, USA: Wiley, 2008, pp. 371–380.

[9] K. M. Digumarti, C. Gehring, S. Coros, J. Hwangbo, and R. Siegwart, "Concurrent optimization of mechanical design and locomotion control of a legged robot," in *Mobile Service Robotics*. Singapore: World Scientific, 2014, pp. 315–323.

[10] B. Ellenberger, "Pybullet gymperium," 2018–2019. [Online]. Available: https://github.com/benelot/pybullet-gym

[11] T. Geijtenbeek, M. Van De Panne, and A. F. Van Der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–11, 2013.

[12] D. E. Goldberg, *Genetic Algorithms*. Noida, India: Pearson Education India, 2006.

[13] R. Grzeszczuk and D. Terzopoulos, "Automated learning of muscle-actuated locomotion through control abstraction," in *Proc. 22nd Annu. Conf. Comput. Graph. interactive Techn.*, 1995, pp. 63–70.

[14] R. Grzeszczuk, D. Terzopoulos, and G. Hinton, "NeuroAnimator: Fast neural network emulation and control of physics-based models," in *Proc. 25th Annu. Conf. Comput. Graph. Interactive Techn.*, 1998, pp. 9–20.

[15] A. Gupta, L. Fan, S. Ganguli, and L. Fei-Fei, "MetaMorph: Learning universal controllers with transformers," 2022, *arXiv:2203.11931*.

[16] A. Gupta, S. Savarese, S. Ganguli, and L. Fei-Fei, "Embodied intelligence via learning and evolution," *Nature Commun.*, vol. 12, no. 1, pp. 1–12, 2021.

[17] D. Ha, "Reinforcement learning for improving agent design," *Artif. Life*, vol. 25, no. 4, pp. 352–365, 2019.

[18] d. Ha, "Reinforcement learning for improving agent design," *Artif. Life*, vol. 25, no. 4, pp. 352–365, 2019, doi: 10.1162/artl_a_00301.

[19] S. Ha, S. Coros, A. Alspach, J. Kim, and K. Yamane, "Task-based limb optimization for legged robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 2062–2068.

[20] S. Ha, S. Coros, A. Alspach, J. Kim, and K. Yamane, *Robotics*, S. Srinivasa, N. Ayanian, N. Amato, and S. Kuindersma, Eds., USA: MIT Press Journals, 2017, doi: 10.15607/rss.2017.xiii.003.

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[22] C. Hartman and B. Benes, "Autonomous boids," *Comput. Animation Virtual Worlds*, vol. 17, no. 3, pp. 199–206 2006, doi: 10.1002/cav.123.

[23] J. H. Holland, "Genetic algorithms," *Sci. Amer.*, vol. 267, no. 1, pp. 66–73, 1992.

[24] W. Huang, I. Mordatch, and D. Pathak, "One policy to control them all: Shared modular policies for agent-agnostic control," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 4455–4464.

[25] J. R. Koza, "Survey of genetic algorithms and genetic programming," in *Proc. Wescon Conf. Rec.*, 1995, pp. 589–594.

[26] S. Lee, J. Lee, and J. Lee, "Learning virtual chimeras by dynamic motion reassembly," *ACM Trans. Graph.*, vol. 41, no. 6, pp. 1–13, 2022.

[27] Y. Lee, S. Kim, and J. Lee, "Data-driven biped control," in *Proc. ACM SIGGRAPH Conf. 2010 Papers*, 2010, pp. 1–8.

[28] C. K. Liu, A. Hertzmann, and Z. Popović, "Learning physics-based motion style with nonlinear inverse optimization," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1071–1081, 2005.

[29] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep Q-learning," *ACM Trans. Graph.*, vol. 36, no. 3, pp. 1–14, 2017.

[30] K. S. Luck, H. B. Amor, and R. Calandra, "Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning," in *Proc. Conf. Robot Learn.*, 2020, pp. 854–869.

[31] V. Makoviychuk et al., "Isaac gym: High performance GPU-based physics simulation for robot learning," 2021, *arXiv:2108.10470*.

[32] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 1–8, 2012.

[33] X. Pan, A. Garg, A. Anandkumar, and Y. Zhu, "Emergent hand morphology and control from optimizing robust grasps of diverse objects," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 7540–7547.

[34] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–14, 2018.

[35] X. B. Peng, G. Berseth, and M. Van de Panne, "Dynamic terrain traversal skills using reinforcement learning," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 1–11, 2015.

[36] X. B. Peng, Y. Guo, L. Halper, S. Levine, and S. Fidler, "ASE: Large-scale reusable adversarial skill embeddings for physically simulated characters," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 1–18, 2022.

[37] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, and S. Levine, "SFV: Reinforcement learning of physical skills from videos," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 1–14, 2018.

[38] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, "Amp: Adversarial motion priors for stylized physics-based character control," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–20, 2021.

[39] X. B. Peng and M. van de Panne, "Learning locomotion skills using deepRL: Does the choice of action space matter?," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2017, pp. 1–13.

[40] R. Pfeifer and J. Bongard, *How the Body Shapes the Way We Think: A New View of Intelligence*. Cambridge, MA, USA: MIT Press, 2006.

[41] M. H. Raibert and J. K. Hodgins, "Animation of dynamic legged locomotion," in *Proc. 18th Annu. Conf. Comput. Graph. Interactive Techn.*, 1991, pp. 349–358.

[42] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proc. 14th Annu. Conf. Comput. Graph. Interactive Techn.*, 1987, pp. 25–34.

[43] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter, "Jointly learning to construct and control agents using deep reinforcement learning," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 9798–9805.

[44] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[45] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv: 1707.06347*.

[47] K. Sims, "Artificial evolution for computer graphics," in *Proc. 18th Annu. Conf. Comput. Graph. Interactive Techn.*, 1991, pp. 319–328.

[48] K. Sims, "Evolving 3 D morphology and behavior by competition," *Artif. Life*, vol. 1, no. 4, pp. 353–372, 1994.

[49] K. Sims, "Evolving virtual creatures," in *Proc. 21st Annu. Conf. Comput. Graph. Interactive Techn.*, 1994, pp. 15–22.

[50] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. robots Syst.*, 2012, pp. 5026–5033.

[51] M. Van de Panne, R. Kim, and E. Fiume, "Virtual wind-up toys for animation," in *Graphics Interface*. Princeton, NJ, USA: Citeseer, 1994, pp. 208–208.

[52] K. Wampler, Z. Popović, and J. Popović, "Generalizing locomotion style to new animals with inverse optimal regression," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 1–11, 2014.
[53] T. Wang, Y. Zhou, S. Fidler, and J. Ba, "Neural graph evolution: Towards efficient automatic robot design," 2019, *arXiv:1906.05370*.
[54] J. Won, D. Gopinath, and J. Hodgins, "A scalable approach to control diverse behaviors for physically simulated characters," *ACM Trans. Graph.*, vol. 39, no. 4, pp. 33–1, 2020.
[55] J. Won, D. Gopinath, and J. Hodgins, "Control strategies for physically simulated characters performing two-player competitive sports," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–11, 2021.
[56] J. Won and J. Lee, "Learning body shape variation in physics-based characters," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 1–12, 2019.
[57] J. Xu, A. Spielberg, A. Zhao, D. Rus, and W. Matusik, "Multi-objective graph heuristic search for terrestrial robot design," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 9863–9869.
[58] K. Yin, K. Loken, and M. Van de Panne, "SIMBICON: Simple biped locomotion control," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 105–es, 2007.
[59] W. Yu, G. Turk, and C. K. Liu, "Learning symmetric and low-energy locomotion," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–12, 2018.
[60] A. Zhao et al., "RoboGrammar: Graph grammar for terrain-optimized robot design," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–16, 2020.