ECoDe: A Sample-Efficient Method for Co-Design of Robotic Agents

Kishan Reddy Nagiredla, Buddhika Laknath Semage, Arun Kumar AV, Thommen George Karimpanal and Santu Rana
Applied Artificial Intelligence Institute (A^2I^2)
Deakin University
Melbourne, Australia
Email: knagiredla@deakin.edu.au

***Abstract*—Co-designing autonomous robotic agents involves simultaneously optimizing the controller and the agent's physical design. Its inherent bi-level optimization formulation necessitates an outer loop design optimization driven by an inner loop control optimization. This can be challenging when the design space is large and each design evaluation involves a data-intensive reinforcement learning process for control optimization. To improve the sample efficiency of co-design, we propose a multi-fidelity-based exploration strategy in which we tie the controllers learned across the design spaces through a universal policy learner for warm-starting subsequent controller learning problems. Experiments performed on a wide range of agent design problems demonstrate the superiority of our method compared to baselines. Additionally, analysis of the optimized designs shows interesting design alterations including design simplifications and non-intuitive alterations.**

I. Introduction

Reinforcement Learning (RL) has been a prominent approach for training agents to learn complex behaviors, relying solely on reward maximization. Whilst most robotics research is centered around a few well-known, fixed skeleton designs e.g., robotic arms or bipedal humanoids, there is an abundance of skeleton designs in nature that equip animals with unique and powerful capabilities. For e.g., the split hoof design of Alpine Ibex makes them excellent climbers, or the strong hind legs make the Kangaroo rats the best jumpers, etc. Exploring such exotic design spaces can lead us to exceedingly more capable designs. Unfortunately, design optimization is a hard problem because the design space can be large, and evaluating designs can be computationally expensive, especially when the control is learned through inherently sample-intensive RL algorithms.

A subset of the robot design problem that we consider in this work deals with fixed skeletal structures with variable parameters. Such problems are often formulated as bi-level optimization problems [2]. This includes (a) searching over the design space in the outer loop and (b) evaluating each design's task-solving capability by learning the control policy in the inner loop. The inner loop typically involves training the agent with Deep-RL methods [13], making it an extremely sample-intensive process. Ha [5] used a Genetic Algorithm (GA) for optimization of the outer loop, and other notable works such as [21], learned a parameter-attribution policy using RL in the outer loop. Unfortunately, both GAs and RL are sample-intensive, and their combinations as such becomes practically infeasible. A naïve solution may include running sample-efficient optimization algorithms such as Bayesian optimization [10] in the outer loop. However, such a solution would still be limited, as firstly, it ignores the fact that control policies corresponding to adjacent regions in the parameter space may be similar to each other. As a result, the policy corresponding to each parameter would necessarily have to be learned from scratch, which can be highly inefficient. Secondly, such solutions ignore the stochastic monotonicity of RL; i.e., the fact that on average, the performance of RL agents tends to monotonically increase with time, which could form a basis prematurely terminating unpromising designs.

To this end, we propose a novel approach in which the morphology and control policy required to perform a task are learned in conjunction. Our framework (a) employs transfer learning to exploit the closeness of the control policies corresponding to adjacent sets of parameters and (b) uses a type of multi-fidelity approach to exploit the stochastic monotonicity in RL. Specifically, we adopt the multi-armed bandit-based hyperparameter optimization method HyperBand [8] that uses a set of multi-level filters, with each filter offering a specific mix of exploration (how many different parameters are examined) and exploitation (how well they are examined). In HyperBand, the widest filter starts with a large set of random parameters, where at the first level a fixed but small sampling budget is provided for evaluating each parameter. Larger sampling budgets are provided in subsequent levels, for evaluating a smaller set of more promising designs.

Since RL generally exhibits stochastic monotonicity, it is more likely that low-fidelity (the policy is trained with smaller number of samples) evaluations of policies are somewhat reflective of their high-fidelity (the policy trained with larger number of samples) evaluations. However, in HyperBand, since the ordering based on just the low-fidelity evaluation can be noisy, to improve the chances of retaining the best designs, a set of top-k parameters are collected and trained with more samples. The whole process repeats until only one best-performing parameter remains. Subsequent filters start with lesser number of initial parameters, but this initial set is provided a larger budget in the first level than the preceding filter.

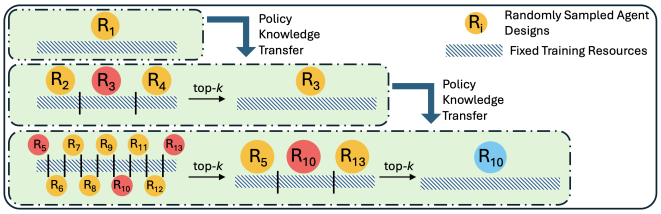


Fig. 1: ECoDe Architecture for multi-fidelity based knowledge propagation mechanism to identify the best co-design (light blue blob). The blobs indicate different robot design samples and hatched boxes indicate the training time. Inside each horizontal box (light green), top-performing samples (red blobs) are made to progress from lower fidelities to higher fidelities. The thick blue arrows indicate the evaluation order to aid effective knowledge transfer through UPN.

This way HyperBand effectively uses stochastic monotonicity to efficiently navigate the sample space. Additionally, we use Universal Policy Network (UPN) [20] to perform transfer learning across parameters where the policy learning for a new parameter borrows knowledge from the policy learned with the last set of parameters. However, UPN-based transfer can create a biased preference for high-fidelity observations as even a bad design with a high number of samples to train its policy can create a seemingly better policy than a good design with only a small number of samples to train its policy. Thus, through Fig. 1 we show how ECoDe navigates the filters in a specific manner (in the opposite direction to HyperBand - explained later in IV) to reduce such bias.

We implement our method on seven OpenAI Gym [4] environments to demonstrate its effectiveness in identifying good co-designs. We contend that these environments are well-suited for our research, as they provide a standardized interface for training RL algorithms across a wide range of environments and tasks, and closely mimic the type of control required for real-world robotics applications [1]. Our results show that our approach outperforms existing methods in identifying good co-designs across multiple environments.

II. Related Works

The co-design of physical and control structures for robotic agents [21] has for long, been a problem of interest. From Von Neumman's work around the idea of agents utilizing evolutionary mechanisms [18] for morphing, similar mechanisms were later used to generate intelligent virtual agents [14]. While some research continued to focus on the optimization of skeletal structures and design parameters, others focused on using pre-selected skeletons to optimize solely over design parameters. Nevertheless, early evolutionary mechanisms were extremely sample-inefficient [19] as it involved evaluating each design in the population. Recent works by Ha [5] and Yuan et al. [21] have addressed this with sample efficient approaches. Parts of the environment are parameterized, facilitating joint learning of policy and physical structure to uncover task-assistive design principles in both these works. By using the implicit function theorem [7], Ha expresses both motion and design parameters as functions describing robot dynamics. His work optimizes over a linear approximation of these functions resulting in agents that learn to change the policy and design parameters depending on the task.

Transfer learning has been used to improve sample efficiency in several works. Schaff et al. [12] demonstrated the transfer of knowledge between previous and new designs by considering design parameters as an additional policy input. Similarly, Luck et al. [9] used the Q-function from RL as an objective function for design adaptation from randomly chosen initial designs. Model-based methods like Villarreal et al. [17] rely on modeling environment dynamics to learn both morphological and control policies sample-efficiently. However, such methods lack robustness, and are sensitive to changes in the dynamics or design parameters, emphasizing the importance of generalization and knowledge transfer for co-design. In many domains, control policies modeled via neural networks and trained using the Deep-RL framework have outperformed prior methods and have delivered state-of-the-art results [22], [20]. Although their ability to learn complex policies without supervision is desirable, deep RL deals solely with policy optimization, and is as such, sample inefficient by nature.

Recently Bayesian Optimization (BO) [10] and HyperBand [8] have successfully been used for sample efficient optimization, specifically in the context of hyperparameter optimization of deep neural networks. While BO provides a general framework for black-box optimization, HyperBand provides a more specific solution that exploits stochastic monotonicity to achieve further sample efficiency through multi-fidelity search. Since RL also demonstrates stochastic monotonicity with respect to the number of steps, HyperBand provides a more natural choice to work in conjunction with RL techniques. In this work, we focus on improving the sample efficiency of co-design by leveraging multi-fidelity methods and the knowledge-sharing aspect of existing policy transfer mechanisms.

III. Background

A. Reinforcement Learning

Reinforcement Learning (RL) problems are modeled as a Markov Decision Process (MDP), represented as $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where $\mathcal{S}$ is the state-space, $\mathcal{A}$ is the action-space, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function governing the next state reached by taking an action $a \in \mathcal{A}$ in a state $s \in \mathcal{S}$, and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the scalar-valued reward producing function for taking action a in state s.

The learning problem is to find the optimal policy that maximizes the returns $\mathbb{E}_{\tau \sim \pi} R(\tau)$, where π is the policy, τ is a trajectory sampled from π, and $R(\tau) = \sum_{(s,a) \in \tau} R(s, a)$ is shorthand for sum of the rewards over the trajectory τ.

B. Universal Policy Network (UPN)

UPN [20] is an RL approach to simultaneously learn a library of policies corresponding to different design parameters.

	$\mathbf{F}=3$		$\mathbf{F}=2$		$\mathbf{F}=1$		$\mathbf{F}=0$	
i	n_i	p_i	n_i	p_i	n_i	p_i	n_i	p_i
0	27	1	9	3	3	9	1	27
1	9	3	3	9	1	27		
2	3	9	1	27				
3	1	27						

TABLE I: HyperBand process showcasing the 4 filters denoted by F and all stage values for number of configurations (n_i) and number of resource units (p_i) when M=27 and η=3.

UPN state $s_{\text{UPN}} \in \mathcal{S}_{\text{UPN}}$ is obtained by augmenting the agent's state s with design parameters θ as $s_{\text{UPN}} = [s, \theta]^\intercal$.

UPN policy π^{UPN} is learned by solving the MDP $\langle \mathcal{S}_{UPN}, \mathcal{A}, T_{UPN}, R \rangle$, where $T_{UPN} : S_{UPN} \times \mathcal{A} \rightarrow S_{UPN}$. Assuming that the control problems between two close sets of design parameters are not very different, π^{UPN} can also be differentiable and thus learnable. UPN has been shown to exploit this property [20] by learning across different design parameters together via large neural networks.

C. HyperBand

HyperBand [8] is a powerful bandit-based multi-fidelity technique that extends the capability and generalizability of a much simpler but effective technique - Successive Halving [6], designed to stop poorly performing configurations early. HyperBand shows a remarkable performance boost in solving the problem of Hyperparameter Optimization in deep neural networks and outperforms random search and Bayesian Optimization [8].

In HyperBand, a given budget B is partitioned into a combination of a number of configurations (M). A budget of $(F_{max}+1)M$ is allocated per configuration in each filter, where F_{max} is the maximum number of filters (obtained from $\lfloor log_\eta(M) \rfloor$). These filters (arranged from highest to lowest exploration as presented in Table I) are independent and resemble arms in a multi-armed bandit technique. Successive Halving is then called as a subroutine on the randomly sampled configurations n_i and a resource budget of p_i is allocated, which then outputs the top-k performers.

IV. Methodology

We aim to determine our optimal design parameters θ^*, given by:

$$\theta^* = \arg\max_{\theta \in \Theta} f(\theta) \tag{1}$$

where θ represents the design parameters to be learned, $f()$ is a performance measure for a particular design choice, and Θ is the space of choices for the design parameters, which we assume to be a bounded set.

In many cases $f()$ can be time-consuming to measure, as learning a policy for complex control problems will require a large number of observations ($\{s, a, R(s,a)\}$). Instead, we consider obtaining a low-fidelity (noisy) evaluation of the policy by providing it with a limited budget of resources for evaluation. The key idea is that although low-fidelity evaluations may be noisy, they may still be reflective of the quality of the designs. For instance, it may be possible to use low-fidelity measurements to discard low-performing designs and reserve the high-fidelity evaluations for more promising design configurations. This manner of multi-fidelity evaluations boosts sample efficiency by ensuring that a large number of samples is not spent for accurately evaluating clearly inferior design configurations. Such problems of optimization of expensive functions, exploiting the multi-fidelity problem is common in neural network HyperParameter Optimization problems (HPO) where θ would be the set of hyperparameters to be tuned, $f()$ being the validation performance. Even in such cases, multi-fidelity measurement of $f()$ can be achieved by limiting the number of training epochs. One prominent HPO algorithm in that context is HyperBand, discussed in III-C.

Though HyperBand has been proven to show good performance, adaptability as well as scalability in high-dimensional spaces, it does not attempt to learn across configurations. For instance, once a design configuration is evaluated using the multi-fidelity approach described above, one would need to redo the evaluation for another configuration, even if it is highly similar to the former. We address this by using UPN, which trains the required configuration while simultaneously updating the policy parameters of neighboring configurations. As a result, when applying the multi-fidelity approach to neighboring configurations, we obtain a much more reliable estimate of its performance.

In the original HyperBand (Table I), filtering is done from left to right ($F = 3$ to $F = 0$), starting with low-fidelity filters, and subsequently moving to high-fidelity ones. Since the nature of low-fidelity filters is to obtain noisy evaluations of a number of configurations, and that of high-fidelity filters is to obtain more reliable evaluations of specific configurations, using UPN in this same direction may not be the best choice. Through ECoDe, we propose that UPN if instead was used in reverse (right to left), it would start with high-fidelity evaluations on a smaller number of configurations, using which policies corresponding to multiple configurations would be updated (as UPN tends to train policies for multiple configurations simultaneously). As a result, the evaluations of more number of configurations in subsequent lower fidelity filters would be more reliable, thereby further improving the sample efficiency of the co-design process.

To summarize, ECoDe evaluates and filters out ineffective co-designs and continually learns from successful co-designs to achieve good task performance in a sample-efficient manner. Our proposed method is provided in Alg. 1.

V. Experiments and Results

A. Environments

Our simulation environments consist of 2 modified Classic control and 5 modified Mujoco [16] environments from OpenAI Gym [4] (shown in Fig. 2).

CartPole (Classic): We use the standard CartPole-v1 environment which has a cart balancing a pole of a given length, and allow for the pole length to be changed in the range $[0.1, 3]$

Algorithm 1 ECoDe (Efficient Co-Design)

Input: M (maximum configurations), η (early-stopping aggressiveness)
Initialize $F_{max} = \lfloor log_{\eta}(M) \rfloor, B = (F_{max} + 1)M$
//$F_{max} + 1$: maximum number of filters, B: budget per filter
for $F \in \{0, 1, ..., F_{max} - 1, F_{max}\}$ **do**
// Begin Successive_Halving with calculated n and p values.
$n = \left\lceil \frac{B}{M} \frac{\eta^F}{(F+1)} \right\rceil$ //n : configurations per filter
$p = M\eta^{-F}$ //p : resource units per filter
$T = get\_hyperparameter\_configuration(n)$
for $i \in \{0, ..., F\}$ **do**
$n_i = \lfloor n\eta^{-i} \rfloor$ //n_i : configurations per stage
$p_i = p\eta^i$ //p_i : resource units per stage
$L = \{run\_UPN(t, p_i) : t \in T\}$
$T =$ top-$k(T, L, \lfloor n_i/\eta \rfloor)$ // best configurations
end for
end for
return $t \in T$ with the largest average reward

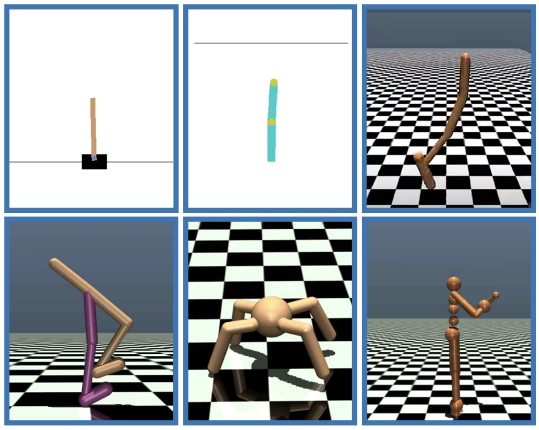


Fig. 2: OpenAI Gym Environments (clockwise from top) - CartPole, Acrobot, Hopper, Humanoid, Ant and Walker2D.

to maintain the pole upright. Additionally, we modified the reward function to impose a ground truth length of 1.425.

Acrobot (Classic): We use the Acrobot-v1 environment, where the agent's goal is to provide enough actuation at the intersection of the two links to get the second link to reach the target height. The configurable design parameters are the link lengths and their masses, which are both varied between 0.1 to 2.

Our simulation environments consist of 2 standard Classic control and 5 standard Mujoco [16] environments from OpenAI Gym [4], discussed below and shown in Fig. 2.

Hopper (Mujoco): We use the Hopper-v3 environment in which the 2D one-limb robot has a torso, thigh, leg, and foot. The Foot and Leg lengths are allowed to vary between 1/4th and 4 times their original lengths.

Walker2D (Mujoco): We use the Walker2D-v3 environment in which the two-limb robot contains 4 main parts: a torso, two thighs, two legs, and two feet. All these parts are allowed to vary within $\pm 20\%$ of their original values.

Ant (Mujoco): We use the standard Ant-v3 environment in which the agent is a 3D ant-like four-limb robot with a spherical torso. The design parameters are the link lengths of all 4 limbs. The range is chosen to be between 0.1 to 0.5.

Humanoid (Mujoco): We use the standard Humanoid-v3 environment in which the agent is a 3D human-like two-legged and two-armed robot. We have considered thigh, shin, and feet lengths of both leg lengths as the design parameters that can vary between 0.5 to 1.5 times the default values.

MaxHumanoid (Mujoco): This environment is similar to the original Humanoid except that we allow for 16 configurable parameters. This presents a highly complex co-design problem where the agent has to choose length and thickness values for all 12 parts in the four limbs with a range similar to that of the original Humanoid environment (0.5 to 1.5 times the default values).

The goal of all the Mujoco agents is to move forward without falling to collect the maximum reward.

B. Baselines

We evaluate our method against the following baselines:

RandomSearch: We randomly sample parameters from a uniform distribution within their ranges. This provides a lower bound on the expected performance of other methods.

Transform2Act [21]: Transform2act uses graph-based representation for agents with limbs represented as edges and joints as nodes in a 3-stage policy optimization - first for skeleton design, second for parameter learning, and then for learning control policy. In our implementation of Transform2Act, we freeze Transform2act's first stage as we assume the skeleton to remain the same.

nLimb [12]: nLimb uses a Gaussian mixture model to parameterize the design distribution and maintains multiple different hypotheses for designs that may be promising. Their method maintains a uniform distribution across components, and half of them with the lowest rewards are eliminated after every N iterations.

HyperBand [8]: In original form, without UPN.

Coadaptation [9]: This approach utilizes the Q-function of a trained policy to evaluate the suitability of given design parameters. Subsequently, it employs particle swarm optimization in conjunction with exploration heuristics to identify the next viable design parameters for evaluation.

We compare these methods with our proposed method (discussed in IV) to make the co-optimization sample efficient. All methods are given a budget of a fixed number of steps that can be taken in an environment.

C. UPN Framework

The UPN agent architecture we use is a dense network with 3 hidden layers, each containing 64 nodes. The input for the network is the concatenated vector of observations (o) for the task with design parameters (θ) for the agent and the environment. The output of this network produces a policy distribution (mean and std. variation) with the size of action space for the environment.

D. Policy Performance

Table II shows the average rewards accumulated by the best parameter-policy combinations for different environments and

by different algorithms. We used a budget of 2.13 million steps for CartPole, Acrobot, Hopper, and Walker2D and an increased 2.84 million steps for Ant and 7.11 million steps for Humanoid, as Ant and Humanoid are more complex control problems.

Meth

The results in Table II represent an average of 10 independent trials along with the standard errors. We implemented Transform2Act and nLimb on the Hopper, Walker2D, and Ant environments and we extended their application to the Humanoid environment. We implemented Coadaptation from scratch and hence were able to compare it with ECoDe on all the environments. Except for CartPole environment, ECoDe significantly outperforms the second-best (i.e. with $p < 0.05$) in all other environments. In CartPole, ECoDe is also better than the second-best but at a slightly reduced significance level (i.e. with $p = 0.06$). This shows the utility of knowledge propagation via UPN and the efficient use of environment interactions through multi-fidelity evaluations.

Surprisingly, Transform2Act sometimes performed worse than Random Search, possibly due to it trying to solve a much harder problem (i.e. learning a policy to choose good designs, rather than directly learning a good design). In Hopper, Humanoid & MaxHumanoid, the default reward function is such that the agents move forward while balancing themselves with minimal control maneuvers. ECoDe seems to focus on getting the balance first by getting the physical design parameters in the right zone (e.g., increase the foot size) and then choosing the best parameter by finding the one easiest to control. This intuitive breakdown of the design process is akin to what an expert might have done.

E. Design Search Optimality

The results in Table II showcase ECoDe's ability to find co-designs that perform significantly better compared to other algorithms across multiple environments. Table III shows the corresponding co-design pole lengths for the CartPole environment as found by different algorithms when given different amounts of sampling budget As seen, even across different sampling budgets, ECoDe found designs, which are closest to the ground truth optimal length of 1.425.

With abundant samples and a powerful network that can learn intricate control rules, the length selection seems less important for achieving higher rewards. However, ECoDe still provides an optimal length closest to the ground truth.

F. Computational Efficiency

The total number of agent-environment interactions (the most time-consuming aspect in RL) for ECoDe is $O(\omega n log n)$ where n is the number of different configurations that we must try and ω is the minimum number of agent-environment interactions that must be allowed. Thus, we see that it uses the total resources quite efficiently. Table **??** shows the wall-clock time for each environment when run on a Dell R6525 2.25GHz CPU with 64 cores and 1TB RAM machine.

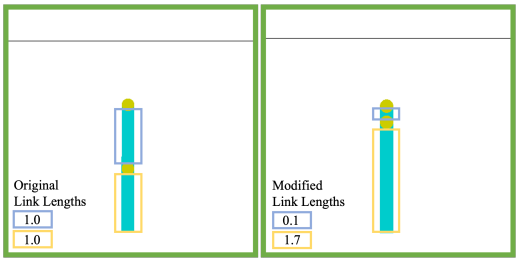


Fig. 3: A visualization of the simplification of design by ECoDe from the original Acrobot (left) to the simplified Acrobot (right), where the control input is applied at the intersection of the two links. The simplified Acrobot (right) resembles a Pendulum and the control problem with this design is much easier to solve as the first link is small compared to the original Acrobot design (left).

Amongst all the baselines, Transform2Act, nLimb, Coadaptation, and ECoDe collected much higher average rewards across multiple environments, possibly owing to the policy-transfer mechanisms associated with them. However, while ECoDe took 1.5 hours to train the bipedal walker in the Walker2D environment, Transform2Act required 6 hours, and the average performance of their co-design is still sub-par (Table II). We observe a similar trend in Hopper and Ant environments as well. In contrast, nLimb and Coadaptation matched our computational efficiency, although ECoDe's multi-fidelity based resource allocation mechanism facilitated collecting high average rewards in similar time.

G. Design Simplification

A rather more interesting result is observed in the Acrobot environment. The best-performing co-design configuration happens to find a design that reduced the more difficult two-link Acrobot control problem to a simpler one-link control problem by choosing the smallest length value for the first link (Fig. 3).

H. Constrained Design

We experimented with breaking the robot's symmetry in the Ant environment by removing the front left leg's ground contact link (Fig. 4). While other methods struggled to identify designs that walk, our method found good co-designs within a 2.8 million steps budget. The resultant design had the other front leg (i.e., right leg) shortened and the hind legs elongated, resembling a Kangaroo rat. This bio-mimetic design exhibits jumping and crawling behaviours. Although the control policy can be further refined, the design appears near-optimal given the limited time steps.

I. Multi-terrain Performance

Furthermore, we applied our method to identify good co-designs in varying terrains. As such, we used the original Humanoid environment and modified it to create an incline and decline surface with a slope of 15°. While the default design barely managed to stay upright in such conditions, our

	CartPole	Acrobot	Hopper	Walker2D	Ant	Humanoid
Random Search	328.4 ± 4.7	-434.6 ± 2.9	587.1 ± 21.2	138.3 ± 3.7	-1183.1 ± 22.1	774.1 ± 28.8
HyperBand [8]	432.3 ± 9.1	-33.2 ± 7.6	823.8 ± 28.4	428.7 ± 4.1	-33.4 ± 2.7	856.8 ± 18.8
Transform2Act [21]	-	-	545.1 ± 2.8	614.1 ± 4.3	533.3 ± 9.3	613.2 ± 8.1
nLimb [12]	-	-	967.3 ± 5.4	1344.5 ± 6.6	691.9 ± 5.7	645.5 ± 3.2
Coadaptation [9]	444.4 ± 5.9	-29.5 ± 2.8	867.5 ± 71.4	1568.9 ± 198.3	1018.3 ± 203.9	819.4 ± 26.9
ECoDe	**464.1 ± 7.9**	**-12.9 ± 2.4**	**1089.4 ± 7.1**	**3297.4 ± 179.6**	**3419.1 ± 128.2**	**4110.3 ± 179.5**

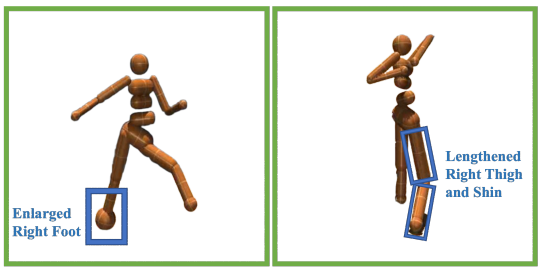
TABLE II: Average Performance across 10 different runs of the best design-policy combination across various environments and algorithms. All algorithms have been given the same sampling budget (2.13 million for all, except for the Ant, which uses 2.84 million steps, and Humanoid, which uses 7.11 million steps).

	Pole Lengths per Sampling Budgets		
Methods	710K	2.13M	2.84M
RandomSearch	1.51± 0.1	1.55 ± 0.1	1.86 ± 0.1
HyperBand	1.11 ± 0.2	1.17 ± 0.1	1.11 ± 0.2
ECoDe	**1.45 ± 0.1**	**1.5 ± 0.1**	**1.34 ± 0.1**

TABLE III: Design analysis (as mean ± standard error) of Random Search, HyperBand and ECoDe algorithms in Cart-Pole Environment over 10 trials when trained with a budget of 710K steps, 2.13M steps and 2.84M steps.

Environment	Runtime (in hours)
CartPole	0:35
Acrobot	0:45
Hopper	1:30
Walker2D	1:30
Ant	6:45
Humanoid	8:30
MaxHumanoid	9:00

TABLE IV: Total runtime per environment



(a) *Incline* (b) *Decline*

Fig. 5: ECoDe has identified that to walk along: (a) Incline plane with a 15°slope, it is preferable to have a larger foot on the trailing leg (right leg) to balance the body with only the forward leg (left leg) being used to move the humanoid forward. (b) Decline plane with a 15°slope, it is preferable to have a longer trailing leg (right leg) relative to the forward leg (left leg), and each leg is swung forward in an alternate sequence, as in regular human walking.

co-designs were able to walk up and down the slope with only a budget of 10.2 million steps, and their average performance is reported in Table V. Most notably, these co-designs were identified to be the best from a pool of 547 candidates.

Moreover, on the flat surface, our method was able to identify functional but non-intuitive asymmetrical designs, showing the existence of asymmetrical designs that can walk as well as the symmetrical design. The chosen designs exhibit a characteristic feature on flat and inclined surfaces, wherein one leg is solely used for balancing while the other drags the body forward. Conversely, on the declined surface, we observe a longer trailing leg and human-like walking. (Fig. 5).

Additionally, in Fig. 6 we present a histogram obtained by running the fully trained UPN over 1000 designs which are randomly sampled from a uniform distribution within the 0.5 to 1.5 times range of default design values. It is hence evident that a large portion of the co-designs are adequately good, however, only a small number of the better ones can produce the highest task performance.

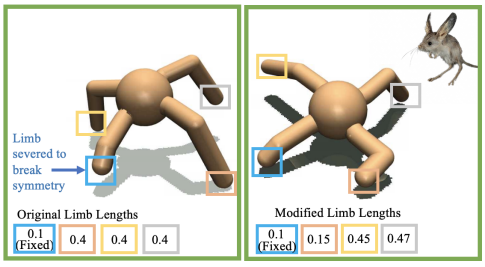


Fig. 4: Illustration of the initial ant robot with a broken limb design (left) when the front left limb is severed (as indicated by the blue bounding box). The simple and non-intuitive co-design (right) suggested by ECoDe, shows shortened front right limb (indicated with the orange bounding box) and lengthened hind limbs (indicated with yellow and grey bounding boxes) resembling a Kangaroo rat.

	Original Design	Co-Design
Flat Terrain	5174.1 ± 86.7	**6133.8 ± 291.3**
Inclined Terrain	4731.3 ± 35.3	**5444.7 ± 80.8**
Declined Terrain	5178.5 ± 44.5	**5852.4 ± 77.5**

TABLE V: Average Rewards (as mean ± standard error) of the original Humanoid design compared with ECoDe's Co-Designed Humanoid across varying terrain conditions computed across 10 independent trials when a 10.2 million steps budget is provided.

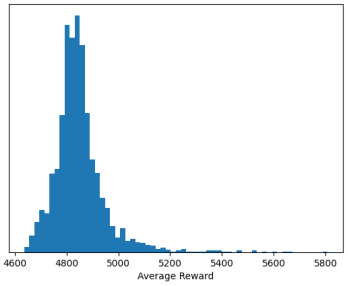


Fig. 6: Histogram of average reward collected by the fully trained ECoDe over 1000 16-dimensional humanoid designs randomly sampled from a uniform distribution.

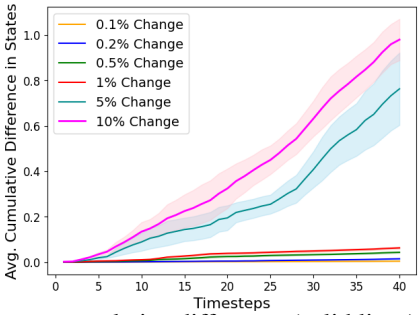


Fig. 7: Average cumulative difference (solid lines) and the std. error (shaded areas) between trajectories (Euclidean distance between states) trained on two nearby sets of parameters $(\theta, \theta's)$ for Walker2D environment.

J. Scalability

We use the MaxHumanoid Environment discussed in V-A to create a humanoid with 16 configurable design parameters, which include individual links on both hands and legs. Remarkably, with the same budget of 10.2 million timesteps, our agent was able to identify the best co-design and learn the complex task of humanoid walking.

K. Stability of Optimal Designs

Our method also showcased remarkable stability when a 2% uniform noise is added to all the identified design parameters and evaluated over 10 episodes. The mean and standard deviation for the 6-dimensional and 16-dimensional configurable design Humanoids was recorded as 6203.3 $\pm$ 208.4 and 4885.7 $\pm$ 40.3. Interestingly, we see a high but less stable performance for 6d optimization but a more stable but lower performance for 16d optimization. It may show further room for improvement for the 16d case where more samples would have led us to a better but sharper performance region.

L. Justification for Algorithm Choice

1) UPN Assumption: : Except for a few boundary cases (where setting a parameter to 0 may imply a different configuration or a different dynamics being at play), the nearby policies in continuous parameter space are similar. To test this hypothesis, we ran an experiment using the Walker2D environment. First, we sample a random set of design parameters θ and then engineer new samples θ' such that they are at different levels of nearness ($||\theta - \theta'||_2 = [0.001, 0.1]$) to the initial sample θ. We independently train θ and θ' without UPN. Fig. 7 shows the average cumulative difference between trajectories between the policies trained on θ and those trained on θ' in their respective environments, averaged over 5 random parameter sets. As evident from Fig. 7 when $||\theta - \theta'||_2 < 0.01$, trajectories are almost identical, whereas larger differences between the parameter sets showed higher trajectory divergence, validating the assumption made by UPN that control problems between close sets of design parameters are not very different.

	Walker2D	Ant	Humanoid
UPNHB	2096.27 $\pm$ 244.6	3212.83 $\pm$ 116.6	2587.86 $\pm$ 343.5
ECoDe	**3297.41 $\pm$ 179.6**	**3419.03 $\pm$ 128.2**	**4110.32 $\pm$ 179.5**

TABLE VI: Average performance comparison (as mean $\pm$ standard error) of UPNHB (a naive UPN and HyperBand combination) with ECoDe across the complex Mujoco environments Walker2D, Ant, and Humaniod over 10 independent trials when a 10.2 million steps budget is provided.

2) HyperBand Filter Order: : As hypothesized in IV, we have experimentally observed that ECoDe performs better than UPNHB (a naive combination of HyperBand and UPN) because it shuns the undesirable bias. For example, while UPNHB selects the best configuration from the right-most (less explorative) filter, $F = 0$ (as shown in Table I), ECoDe selects the best configuration from the left-most (most explorative) filter, $F = 3$. Since in the latter case the best configuration is chosen from a large number of initial configurations, it results in lower bias. We report the performance difference between ECoDe and UPNHB in the most complex environments like Ant and Humanoid in Table VI.

M. Sim-to-Real Transferibility

1) Reliance on Simulation Environment: Effectively transferring behaviors learned through simulation into real-world robots has been demonstrated in multiple prominent works including [3] [15] and [11]. While bridging the reality gap is necessary to transfer behaviors learned in simulation to reality, which is a challenge in itself, this work addresses the preceding problem of how to derive reliable co- designs for any given task in a sample efficient manner. Hence, through simulated environments, this work explores diverse co-design scenarios both for training and testing which can be extremely challenging to set up in the real world.

2) Robustness of ECoDe: To study the robustness of our method's learned behavior, we introduced minor simulation environment perturbations in the Ant environment by randomly changing 5 parameters, i.e., sliding friction, torsional friction, rolling friction, joint damping, and joint stiffness in a $\pm 5\%$ range around their default values. We only noticed a small variation in performance (within the 1.2% of the original value). In these different environmental conditions, we

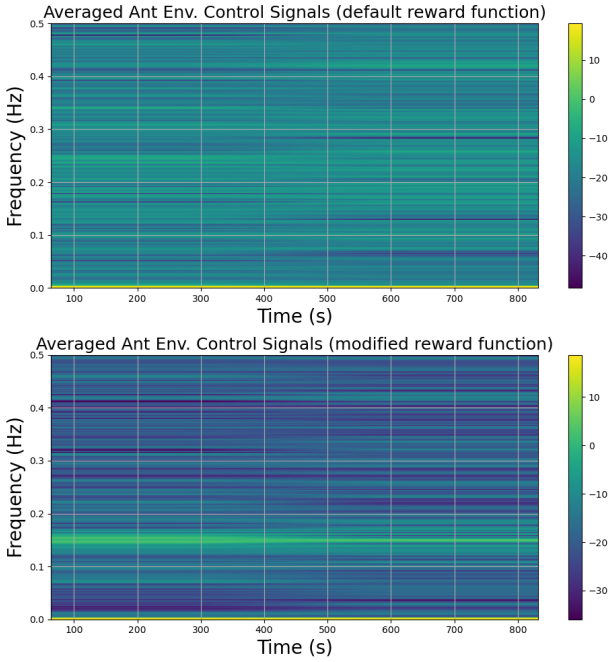


Fig. 8: The spectrogram plots of the averaged control signals show that with the modified reward function (i.e. when no penalty is applied for energy usage), all the limb's movements are in sync and thus would seem more natural. In contrast, with the default reward function, each limb appears to have different base frequencies and thus will lack any naturalistic rhythm.

observed small changes in gait but no change in the agent's ability to walk.

3) Lack of Naturality in Agent's Gait: We acknowledge that we observed some seemingly unnatural movements of our optimized agents. While that can partially be attributed to the simplistic nature of the simulator, our analysis suggests that the primary cause was the nature of the reward function used (as presented in Fig. 8). The default reward function comprises two elements: a reward for movement and a penalty for energy usage. The latter minimizes joint movements, affecting the naturalness of the agent's movements. By removing this penalty, we observed a noticeable improvement in the naturalness of the movement. Hence, by adjusting rewards, we can effectively tailor design outcomes for real-world applications where the goals are almost always multi-objective. However, discussions on designing reward functions for real-world application contexts extend beyond the scope of this paper.

N. Discussion

A natural extension of our work is to make it applicable for free-form skeletal designing, which is an extremely challenging problem and will require innovations in the design iterator that can handle a variable number of parameters as well as in a transfer learning mechanism that can transfer control policies across different skeleton structures. Other extensions of our work can be towards making the co-design more robust using sim2real strategies such as domain randomization, optimization based on multi-objective criteria such as improving naturalness in the movement, integrating the differential cost of design changes for different parts, etc.

VI. Conclusion

We presented ECoDe, a multi-fidelity-based co-design method that uses a transfer learning mechanism to efficiently discover optimal design-control policy combinations for robotic agent design. Specifically, we perform a multi-fidelity search whilst warm-starting policies so that inferior designs can be identified and discarded using fewer samples, resulting in a sample-efficient co-design method. We evaluated our method on 7 different robot design problems using realistic physics simulators and the results show that on all occasions ECoDe performed the best. An interesting future direction may include going beyond just design parameters and adapting ECoDe to co-design the skeletal structure as a whole.

References

[1] Andrychowicz, O.M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al.: Learning dexterous in-hand manipulation. International Journal of Robotics Research (2020)

[2] Bhatia, J., Jackson, H., Tian, Y., Xu, J., Matusik, W.: Evolution gym: A large-scale benchmark for evolving soft robots. In Neural Information Processing Systems **34**, 2201–2214 (2021)

[3] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., Vanhoucke, V.: Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 4243–4250 (2018). https://doi.org/10.1109/ICRA.2018.8460875

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv e-prints (2016)

[5] Ha, D.: Reinforcement learning for improving agent design. Artificial life **25**(4), 352–365 (2019)

[6] Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Artificial intelligence and statistics (2016)

[7] Jittorntrum, K.: An implicit function theorem. Journal of Optimization Theory and Applications **25**(4) (1978)

[8] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. The Journal of Machine Learning Research **18**(1) (2017)

[9] Luck, K.S., Amor, H.B., Calandra, R.: Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning. In: Conference on Robot Learning. PMLR (2020)

[10] Pelikan, M., Goldberg, D.E., Cantú-Paz, E., et al.: Boa: The bayesian optimization algorithm. In: Proceedings

of the genetic and evolutionary computation conference GECCO-99. vol. 1 (1999)
[11] Peng, X.B., Coumans, E., Zhang, T., Lee, T.W., Tan, J., Levine, S.: Learning agile robotic locomotion skills by imitating animals. arXiv preprint arXiv:2004.00784 (2020)
[12] Schaff, C., Yunis, D., Chakrabarti, A., Walter, M.R.: Jointly learning to construct and control agents using deep reinforcement learning. In: 2019 International Conference on Robotics and Automation (ICRA). pp. 9798–9805. IEEE (2019)
[13] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
[14] Sims, K.: Evolving virtual creatures. In: Proceedings of the 21st annual conference on Computer graphics and interactive techniques (1994)
[15] Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., Vanhoucke, V.: Sim-to-real: Learning agile locomotion for quadruped robots. arXiv preprint arXiv:1804.10332 (2018)
[16] Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: IEEE/RSJ international conference on intelligent robots and systems (2012)
[17] Villarreal-Cervantes, M.G., Cruz-Villar, C.A., Alvarez-Gallegos, J., Portilla-Flores, E.A.: Robust structure-control design approach for mechatronic systems. IEEE/ASME Transactions on Mechatronics **18**(5), 1592–1601 (2012)
[18] Von Neumann, J., Burks, A.W.: Theory of self-reproducing automata. IEEE Transactions on Neural Networks **5**(1) (1996)
[19] Yu, T., Zhu, H.: Hyper-parameter Optimization: A Review of Algorithms and Applications. arXiv preprint arXiv:2003.05689 (2020)
[20] Yu, W., Tan, J., Bai, Y., Coumans, E., Ha, S.: Learning fast adaptation with meta strategy optimization. IEEE Robotics and Automation Letters **5**(2) (2020)
[21] Yuan, Y., Song, Y., Luo, Z., Sun, W., Kitani, K.M.: Transform2act: Learning a transform-and-control policy for efficient agent design. In: International Conference on Learning Representations (2021)
[22] Zhao, W., Queralta, J.P., Westerlund, T.: Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: 2020 IEEE symposium series on computational intelligence (SSCI). pp. 737–744 (2020)