Data-efficient Co-Adaptation of Morphology and Behaviour with Deep Reinforcement Learning

Kevin Sebastian Luck
Interactive Robotics Lab
Arizona State University
United States
ksluck@asu.edu

Heni Ben Amor
Interactive Robotics Lab
Arizona State University
United States
hbenamor@asu.edu

Roberto Calandra
Facebook AI Research
United States
rcalandra@fb.com

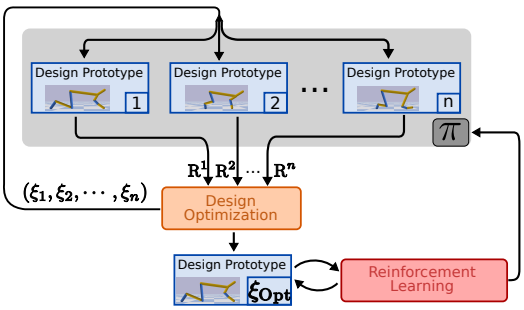
Abstract: Humans and animals are capable of quickly learning new behaviours to solve new tasks. Yet, we often forget that they also rely on a highly specialized morphology that co-adapted with motor control throughout thousands of years. Although compelling, the idea of co-adapting morphology and behaviours in robots is often unfeasible because of the long manufacturing times, and the need to re-design an appropriate controller for each morphology. In this paper, we propose a novel approach to automatically and efficiently co-adapt a robot morphology and its controller. Our approach is based on recent advances in deep reinforcement learning, and specifically the soft actor critic algorithm. Key to our approach is the possibility of leveraging previously tested morphologies and behaviors to estimate the performance of new candidate morphologies. As such, we can make full use of the information available for making more informed decisions, with the ultimate goal of achieving a more data-efficient co-adaptation (i.e., reducing the number of morphologies and behaviors tested). Simulated experiments show that our approach requires drastically less design prototypes to find good morphology-behaviour combinations, making this method particularly suitable for future co-adaptation of robot designs in the real world.

Keywords: Co-adaptation, Morphology, Deep Reinforcement Learning

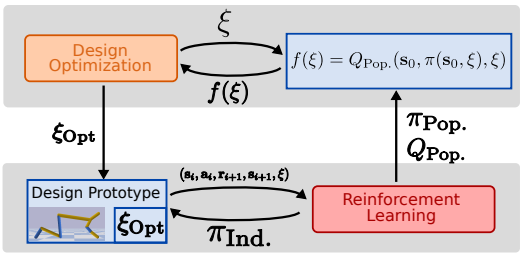
1 Introduction

In nature, both morphology and behaviour of a species crucially shape its physical interactions with the environment [1]. For example, the diversity in animal locomotion styles is an immediate result of the interplay between different body structures, e.g., different numbers, compositions and shapes of limbs, as well as as different neuromuscular controls, e.g., different sensory-motor loops and neural periodic patterns. Adaptation of a species to new ecological opportunities often comes with changes to both body shape and control signals – *morphology and behaviour are co-adapted*. Building upon this insight, we investigate in this paper a methodology for co-adaptation of the morphology and behaviour for computational agents using deep reinforcement learning. Without loss of generality, we focus in particular on legged locomotion. The goal of legged robots in such locomotion tasks is to transform as much electric energy as possible into directional movement [2, 3, 4, 5]. To this end, two approaches exist: 1) optimization of the behavioural policy, and 2) optimization of the robot design, which affects the achievable locomotion efficiency [2, 6, 7, 8]. Policy optimization is, especially in novel or changing environments, often performed using reinforcement learning [8, 9]. Design optimization is frequently based on evolutionary algorithms or evolution-inspired and use a population of design prototypes for this process (Fig. 1a) [2, 6, 10]. However, manufacturing and evaluating a large quantity of design candidates is often infeasible in the real world due to cost and time constraints, especially for larger robots. Therefore, the evaluation of designs is often restricted to simulation, which is feasible but suffers from the simulation-to-reality-gap [11, 12]. Designs and control policies optimized in simulation are often not the best possible choice for the real world, especially if the robotics system is complex and the environmental parameters hard to model. For example, in the work of Lipson and Pollack [13] designs were first optimized in simulation in an evolutionary manner and then manufactured in the real world. However, the performances of the

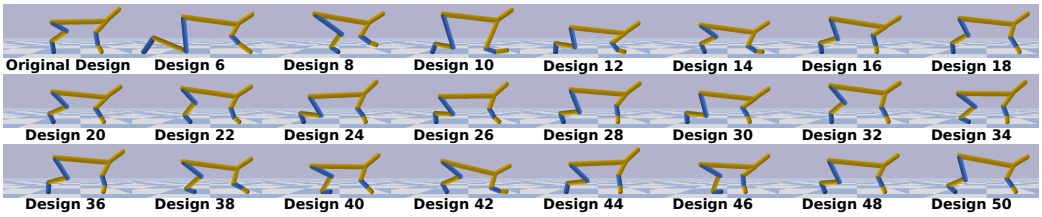
3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.



(a) Traditional approach in which designs are evaluated in simulation during the design optimization process.



(b) Proposed method which evaluates designs with the learned Q- and policy network and thus reduces the amount of simulations or physical prototypes required during the design optimization process.



(c) Designs ξ_{Opt} selected by the proposed method for the Half-Cheetah task.

Figure 1: We propose to (b) use an actor and critic for design exploration instead of (a) creating design prototypes and evaluating their performance in simulation or the real world. Our goal is to reduce the amount of time needed to (c) evolve a robotic prototype in the real world.

manufactured designs in the real world were significant lower than in simulation in all but one case (see Table 1 in [13]), even though efforts were undertaken to close the simulation-to-reality gap for the described robot.

The method proposed in this work caters towards the need of roboticists for data-efficiency in respect to the number of prototypes required to achieve an optimal design. We are combining design optimization and reinforcement learning in such a way that the reinforcement learning process provides us with an objective function for the design optimization process (Fig. 1b). Thus, eliminating the need for a population of prototypes and requiring only one functioning prototype at a time.

2 Related Work

The work of Schaff et al. [7] is a relatively recent approach to combine reinforcement learning and design optimization into one framework. The common idea is to consider the design parameter ξ as an additional input to the policy $\pi(\mathrm{s}, \xi)$ and to optimize the expected reward $\mathbb{E}[R]$ given the policy and design. The policy is trained such that it is able to generalize over many designs and is iteratively updated with experience collected from a population of n prototypes. The algorithm maintains a distribution over designs, whose parameters are optimized to maximize the expected reward. However, this approach [7] requires the maintenance of a population of designs, which is updated every t timesteps and relies on the simulator to compute the fitness of designs. Similarly, the work of David Ha [14] uses the design parameters ξ as input to the policy $\pi(\mathrm{s}, \xi)$ but uses REINFORCE [15] to update the design parameters. Again, this approach requires a population of design prototypes to compute the introduced population-based policy gradient for the design as well as rewards collected from the simulator. The recent method introduced by Liao et al. [16] employs Batch Bayesian Optimization to improve morphology and policies. The expected performance of designs is here learned and inferred by Gaussian Processes (GP), a second GP is also used to optimize the parameters of central pattern generators representing movement policies. The paper demonstrates the design optimization of a simulated micro-robot with three parameters defining the morphology. While the presented results are using a prototype population of 5 designs, the authors mention that the proposed method can handle a single prototype as well. One drawback of [16] is, however, that the GP predicting the fitness of designs is trained only with a single value per design: the single highest

reward achieved for a design. Since the maximum reward is potentially affected by the initial state a robot is in, this approach has a reduced applicability to tasks with noisy or random start states. In [2], the leg lengths and controller of a quadruped robot were optimized in the real world. The controller was here based on the inverse kinematics of the robot and defined by tuning eight parameters. All leg segment lengths were described by a two-dimensional design vector. Two different evolutionary algorithms were used to optimize these parameters over eight generations with a population size of eight and based on the reward received. While this experiment is an impressive demonstration of the potential of adapting behaviour and morphology in the real world, the task was simplified through the use of a re-configurable robot which is able to adapt its leg-lengths automatically. This decreases the setup-time required between experiments because manufacturing of leg-segments or other body parts are not necessary. All four of these approaches rely on a population of design prototypes whose performance must be evaluated in simulation or the real world, or rely on a single reward.

3 Problem Statement

We formalize the problem of co-adapting morphology and behavior as the optimization

$$\theta^* = \arg\max_\theta R|_\theta \,, \tag{1}$$

of the reward R w.r.t. the variables $\theta = [\xi, \pi]$ were ξ are the morphological properties of the agent, and π the behavior. There are multiple ways to tackle this problem. One commonly used way is to decompose it as bi-level optimization, where we iteratively optimize the morphology first ξ, and after fixing it, we optimize the behavior π. One advantage of this formulation is that by decoupling the two optimization, we can take into consideration the fact that evaluating different morphologies has an associated cost (e.g., manufacturing a physical robot) which can be substantially higher than evaluating different behaviors (e.g., running multiple controllers). In this paper, we frame the learning of the behaviors as an extension of the standard Markov decision process (MDP) [17] given the additional design variable ξ (i.e., the context). In this model, the transition probability to reach a state s_{t+1} after performing action a_t is given by $p(\mathrm{s}_{t+1}|\mathrm{s}_t, \mathrm{a}_t, \xi)$ and depends on design properties ξ of the agent. The reward function $\mathrm{r}(\mathrm{s}, \mathrm{a}, \xi)$ can be dependent on the design as well. For notational clarity, we will generally use $r(\mathrm{s})$ in the remainder of the paper. The actions are generated from the policy $\pi(\mathrm{s}, \xi)$ and the goal is to maximize the expected future reward given by

$$\mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i \mathrm{r}(\mathrm{s}_{t+i+1}, \mathrm{a}_{t+i+1}, \xi) \middle| \mathrm{s}_t = \mathrm{s}, \mathrm{a}_i = \pi(\mathrm{s}_i), \xi \right], \tag{2}$$

with $\gamma \in [0, 1]$ being a discount factor and future states s_{t+i+1} produced by the transition function. Our goal is hence to maximize this objective function for both the policy π and the design ξ using deep reinforcement learning.

4 Optimization of Morphology and Behaviour

We now introduce our proposed framework for sample-efficient optimization of behaviour and design for robotic prototypes. We first describe our novel objective function based on an actor and critic to remove the dependency on prototypes and simulations during design optimization. Thereafter, a method is described for fast behaviour adaptation by training a copy of actor and critic primarily on experience collected with the current design prototype. We continue with an explanation of two different design exploration mechanisms, random selection and novelty search. The chapter closes with a description of the reinforcement learning algorithms and optimization routines used.

4.1 Using the Q-Function for Design Optimization

Optimizing the behaviour of an agent usually requires learning a value or Q-value function and a policy π by the means of reinforcement learning. The rationale of our approach is to extend this methodology to the evaluation of the space of designs, thereby reducing the need for large numbers of simulations or manufactured robot prototypes.

The goal of *design optimization* is to increase the efficiency of the agent given an optimal policy for each design. The objective function for this case can be the sum of rewards collected by evaluating

the behaviour of the agent with this design, given by

$$\max_{\xi} \mathbb{E}_{\pi} \left[\sum_{i=0}^{T} \mathrm{r}_t \right], \tag{3}$$

where the rewards are collected through the execution of a policy π on the agent with design ξ in the real world or in simulation.

To alleviate the aforementioned problems with the evaluation through executions in simulation or real world, we instead propose to reuse the Q-function learned by a deep reinforcement learning algorithm and re-formulate our objective as

$$\max_{\xi} \mathbb{E}_{\pi} \left[Q(\mathrm{s}, \mathrm{a}, \xi) | \mathrm{a} = \pi(\mathrm{s}, \xi) \right], \tag{4}$$

where the action a is given by the policy $\pi(\mathrm{s}, \xi)$. This creates a strong coupling between the design optimization and reinforcement learning loop: We effectively reduce the problem of finding optimal designs to the problem of training a critic which is able to generate an estimated performance of a design given state and action. This means, while optimizing a policy for a design, we also train the objective function given above at the same time. We hypothesize that, during the training process, the critic learns to distinguish and interpolate between designs due to the influence of the design on the reward of transitions. We further reformulate Eq. 4 to optimize over the distribution of start states encountered in trajectories $(\mathrm{s}_0, \mathrm{a}_0, \mathrm{s}_1, \cdots, \mathrm{s}_T)$. The objective function becomes then the expected future reward given a design choice ξ. This could be, for example, the case if the leg lengths of a robot are optimized and the initial position is a standing one. Here, the initial height of the robot would vary with the design choice. Thus, we reformulate the objective function in Eq. 4 such that we optimize over the distribution of start states with

$$\max_{\xi} \mathbb{E}_{\mathrm{s}_0 \sim p(\mathrm{s}_0|\xi)} \left[\mathbb{E}_{\pi} \left[Q(\mathrm{s}_0, \mathrm{a}_0, \xi) | \mathrm{a}_0 = \pi(\mathrm{s}_0, \xi) \right] \right]. \tag{5}$$

The motivation to optimize this function over the distribution of start states is to take potential randomness in the initial positions, or even inaccuracies when resetting the initial position of a robot, into account. Since the distribution of start states might be unknown or even depend on the design, we approximate the expectation by drawing a random batch of start states s_0 from a replay buffer, which contains exclusively all start states seen so far. If we use a deterministic deep neural network for policy π, Eq. 5 reduces to

$$\max_{\xi} \frac{1}{n} \sum_{\mathrm{s} \in \mathrm{s}_{\text{batch}}} Q(\mathrm{s}, \pi(\mathrm{s}, \xi), \xi), \tag{6}$$

with $\mathrm{s}_{\text{batch}} = (\mathrm{s}_0^1, \mathrm{s}_0^2, \cdots, \mathrm{s}_0^n)$ containing n randomly chosen start states. This objective function can be optimized with classical global optimization methods such as Particle Swarm Optimization (PSO) [18, 19] or Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [20].

4.2 Design Generalization and Specialization of Actor and Critic

A naive solution to input the design variable into the actor and critic network would be to append the design vector to the state and train a single set of networks using the experience of all designs. A more promising approach is to have two sets of networks: One *population* (pop.) actor and critic network which is trained on the training experience from all designs, and *individual* (ind.) networks which are initialized with the *population* network but use primarily training experience from the current design (the individual). In practice, we found it helpful to allocate 10% of the training batch for samples from the *population* replay buffer when training the *individual* networks. Essentially, this approach allows the *individual* networks $Q_{\text{Ind.}}$ and $\pi_{\text{Ind.}}$ to specialize in a fast manner to the current design and its nuances to quickly achieve maximum performance. In parallel, we are training the *population* networks $Q_{\text{Pop.}}$ and $\pi_{\text{Pop.}}$ with experience from all designs seen so far by selecting samples from the *population* replay buffer $\text{Replay}_{\text{Pop.}}$. These *population* networks are then able to better generalize across different designs and provide initial weights for the *individual* networks. Hence, policies do not have to be learned from scratch for each new prototype. Instead, previously collected training data is used so that different designs can inform each other and make efficient use of all the experiences collected thus far.

Algorithm 1 Fast Evolution through Actor-Critic Reinforcement Learning

Initialize replay buffers: $\text{Replay}_{\text{Pop.}}$, $\text{Replay}_{\text{Ind.}}$ and Replay_{s_0}
Initialize first design ξ
for $i \in (1, 2, \cdots, M)$ **do**
  $\pi_{\text{Ind.}} = \pi_{\text{Pop.}}$
  $Q_{\text{Ind.}} = Q_{\text{Pop.}}$
  Initialize and empty $\text{Replay}_{\text{Ind.}}$
  while not finished optimizing local policy **do**
    Collect training experience $(s_0, a_0, r_1, s_1, \cdots, s_T, r_T)$ for current design ξ with policy network $\pi_{\text{Ind.}}$
    Add quadruples $(s_i, a_i, r_{i+1}, s_{i+1})$ to $\text{Replay}_{\text{Ind.}}$
    Add quintuples $(s_i, a_i, r_{i+1}, s_{i+1}, \xi)$ to $\text{Replay}_{\text{Pop.}}$
    Add start state s_0 to Replay_{s_0}
    Train networks $\pi_{\text{Ind.}}$ and $Q_{\text{Ind.}}$ with random batches from $\text{Replay}_{\text{Ind.}}$
    Train networks $\pi_{\text{Pop.}}$ and $Q_{\text{Pop.}}$ with random batches from $\text{Replay}_{\text{Pop.}}$
  end while
  if i is even **then**
    Sample batch of start states $s_{\text{batch}} = (s_0^1, s_0^2, \cdots, s_0^n)$ from Replay_{s_0}
    Exploitation: Compute optimal design ξ with objective function $\max_\xi \frac{1}{n} \sum_{s \in s_{\text{batch}}} Q_{\text{Pop.}}(s, \pi_{\text{Pop.}}(s, \xi), \xi)$
  else
    Exploration: Sample design ξ with exploration strategy
  end if
end for

4.3 Exploration and Exploitation of Designs

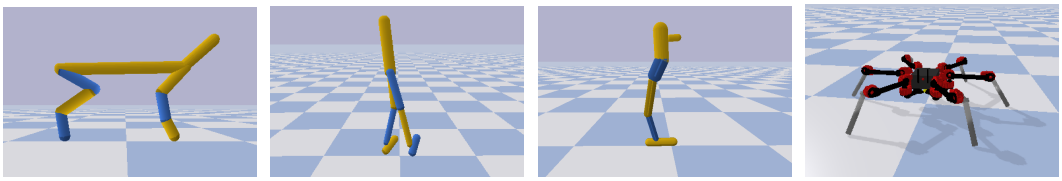
We alternate between design exploration and exploitation to increase the diversity of explored designs, improve generalization capabilities of the critic and avoid an early convergence to regions of the design space. Therefore, every time we find an optimal design during the design optimization process with the objective function (Eq. 6) and conclude the subsequent reinforcement learning process, we next choose one design using the exploration strategy. To this end, we implemented two different approaches: sampling new designs 1) randomly, and 2) using Novelty search [21]. We found that using random sampling as exploration strategy outperformed novelty search (see appendix).

4.4 Fast Evolution through Actor-Critic Reinforcement Learning

The proposed algorithm, Fast Evolution through Actor-Critic Reinforcement Learning, is presented in Algorithm 1. We will now discuss the specifics of the used reinforcement learning algorithm and global optimization method. However, it is worth noting that our methodology is agnostic to the specific algorithms used for design and behaviour optimization.

Reinforcement Learning Algorithm While in principal every reinforcement learning method can be employed to train the Q and policy functions necessary to optimize the designs, we use a deep reinforcement learning method due to the continuous state and action domains of our tasks. Specifically, we employed the Soft-Actor-Critic (SAC) algorithm [22], a state-of-the-art deep reinforcement learning method based on the actor-critic architecture. All neural networks had three hidden layers with a layer size of 200. Per episode we train the *individual* networks $\pi_{\text{Ind.}}$ and $Q_{\text{Ind.}}$ 1000 times while the *population* networks $\pi_{\text{Pop.}}$ and $Q_{\text{Pop.}}$ are trained 250 times. The motivation was to assign more processing power to the *individual* networks to adapt quickly to a design and specialize. A batch size of 256 was used for each training updated.

Optimization Algorithm To optimize the objective function given in Eq. (6), we used the global optimization method Particle Swarm Optimization (PSO) [18, 19]. We chose PSO primarily because of its ability to search the design space exhaustively using a large number of particles. The objective function (Eq. (6)) was optimized using about 700 particles, each representing a candidate design, and updated over 250 iterations. Accordingly, PSO used a total contingent of 175,000 objective function evaluations to find an optimal design. To optimize the design using rollouts in simulation, we had to reduce this number to about 1,050 design candidates, i.e. 35 particles updated over 30 iterations. Although this contingent is only about 0.6% of the size of the Q-function contingent, it takes about two times longer to evaluate this number of designs in simulation. For example, on a system with an Intel Xeon CPU E5-2630 v4 CPU equipped with an NVIDIA Quadro P6000, the design optimization via simulation takes approximately 30 minutes while the optimization routine using the critic requires only 15 minutes. To put this into perspective, the reinforcement learning process on a single design requires approximately 60 minutes for 100 episodes.



(a) Half Cheetah (b) Walker (c) Hopper (d) Daisy Hexapod

Figure 2: The four simulated robots used in our experiments.

5 Experimental Evaluation

We now experimentally evaluate our proposed approach, with the aim of answering the following questions: 1) Can we obtain with our algorithm comparable task performance as optimizing the design by performing extensive trials, by instead relying on the learned model? 2) If so, how much can our approach reduce the number of trials? 3) Can our approach help us to get insight into the design space that we are trying to optimize for a specific task?

Code for reproducing the experiments, videos, and additional material is available online at `https://sites.google.com/view/drl-coadaptation`.

5.1 Experimental Setting

To evaluate our algorithm, we considered the four control tasks simulated using PyBullet [23] shown in Fig. 2. The design of agents for each task is described as a continuous design vector $\xi \in \mathbb{R}^d$. The initial five designs for each task were pre-selected with the original design and four randomly chosen designs which were consistent over all experiments. All experiments were repeated five times. For the standard PyBullet tasks (Figures 2a to 2c) we executed 300 episodes for the initial five designs and 100 episodes thereafter. The latter was increased to 200 episodes for the more complex Daisy Hexapod task (Fig. 2d) [24]. We will give a short description of the simulated locomotion tasks and state for each task the number of states, actions and design parameters as a vector $(\mathrm{s}, \mathrm{a}, \xi)$. A detailed descriptions of the tasks can be found in the appendix. **Half-Cheetah (17, 6, 6)** and **Walker (17, 6, 6)** are agents with two legs tasked to learn to run forward. Each agent has six leg segments to be optimized independently for their length. The **Hopper (13, 4, 5)** agent has a single leg with four leg segments as well as a nose-like feature and has to learn to move forward as well. All three agents are restricted to movements in a 2D plane. The **Daisy Hexapod (43, 18, 9)** simulates an hexapod and is able to move in all three dimensions. Its goal is to learn to move forward without changing its orientation. The lengths of the leg-segments are mirrored between the left and right side of the robot, with three leg-segments per leg.

5.2 Co-adaptation Performance

We compared the proposed framework, using actor-critic networks for design evaluation, and the classical approach, optimizing the design through candidate evaluations in simulation, on all four locomotion tasks (Fig. 3). We can see that, especially in the Half-Cheetah task, using actor-critic networks might perform worse over the first few designs but quickly reaches a comparable performance and even surpasses the baseline. It is hypothesized that the better performance in later episodes is due to the ability of the critic to interpolate between designs while the evaluations of designs in simulation suffers from noise during execution. Interestingly, using simulations to optimize the design does not seem to lead to much improvement in the case of the Walker task. This could be due to the randomized start state, which often leads to the agent being in an initial state of falling backwards or forwards, which would have an immediate effect on the episodic reward. Additionally, we compared the proposed method using the introduced objective function for evaluating design candidates against the method used for design optimization in [14]. Fig. 5 shows that the evolution strategy OpenAI-ES [25], using the simulator to evaluate design candidates with a population size of 256, is outperformed by our proposed method. Moreover, we verified that for all experiments, designs selected randomly, with a uniform distribution, performed worse than designs selected through optimization (see Fig. 5).

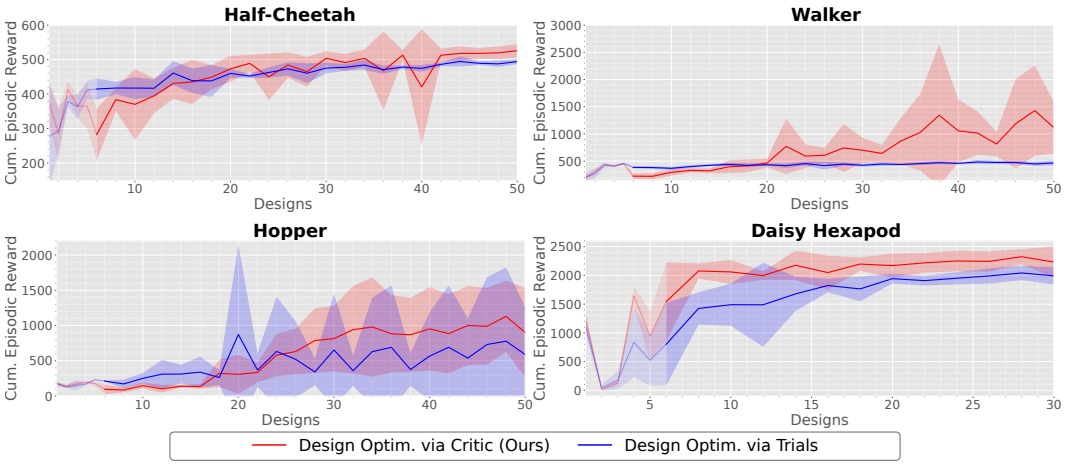


Figure 3: Comparison between our proposed approach (red) and using trials to evaluate the optimality of candidate designs (blue). The plots each show the mean and standard deviation of the highest reward achieved over five experiments for optimal designs ξ_{Opt}. We can see that the proposed method (Fig. 1b) has a comparable or even better performance than optimizing designs via executions in simulation (Fig. 1a).

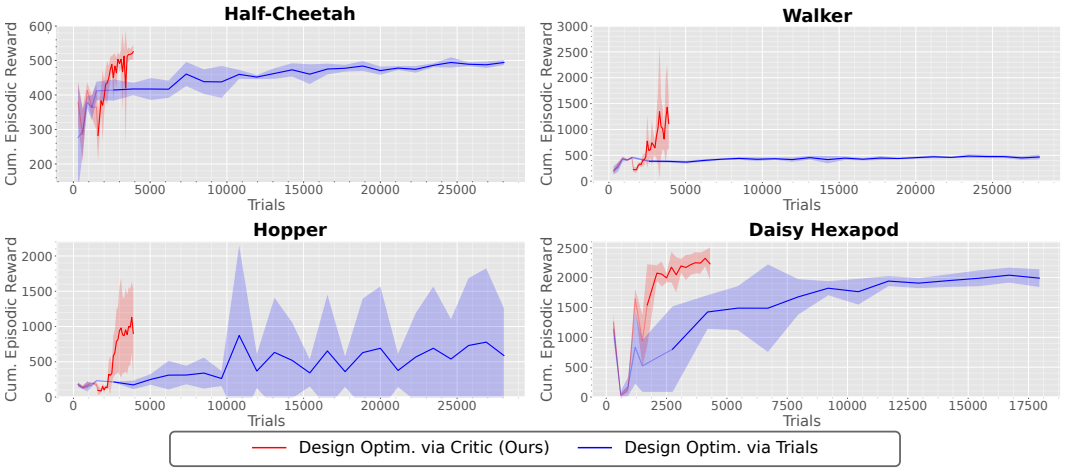


Figure 4: Comparison between the our proposed approach of using the actor and critic to optimize the design parameters (red) and using trials/simulations to evaluate the optimality of candidate designs (blue). The plots show the mean and standard deviation of the highest reward achieved over five experiments. The x-axis shows the number of episodes executed in simulation. We can see that removing the need to simulate design candidates during the design optimization process leads to a comparable performance in a much shorter time frame.

Simulation Efficiency To evaluate the suitability of the proposed method for deployment in the real world, we compared the methods based on the number of simulations required. As we can see in Fig. 4, the actor-critic approach quickly reaches a high performance quickly with a low number of simulations. As explained above, this is due to the design optimization via simulation requiring 1,050 simulations to find an optimal design while the proposed method requires none.

Visualization of Reward Landscapes for Designs A major advantage of the proposed method is the possibility to visualize the expected reward for designs. Instead of selecting a number of designs to evaluate, which would take a significant effort in the real world as well as computationally, we are able to query the introduced objective function (Eq. (6)) in a fast manner. This allows us to visually inspect the reward landscape of designs and to gai ninsight at what makes designs perform better or worse. In Fig. 6, the first two principal components were computed based on the designs selected for

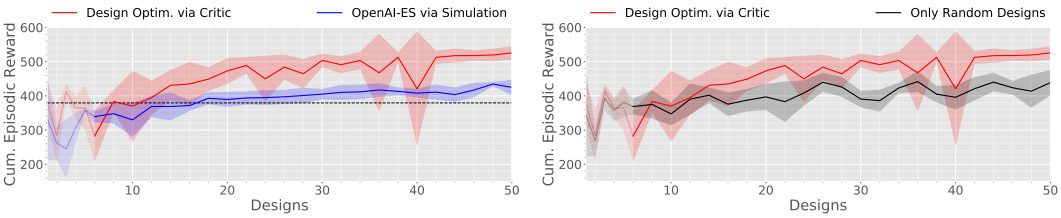


Figure 5: (*Left*) Comparison on the Half-Cheetah against the proposed method and OpenAI-ES [25] with a population size of 256 as used in [14]. While our method uses the proposed objective function, OpenAI-ES uses the simulator for evaluating design candidates. The dotted line shows the average reward achieved on the original design of Half-Cheetah. (*Right*) Comparison of the proposed method and sampling **only** random designs instead of optimizing the objective function. The plots show the mean and standard deviation of the highest reward achieved over five experiments. The proposed approach outperforms the random baseline.

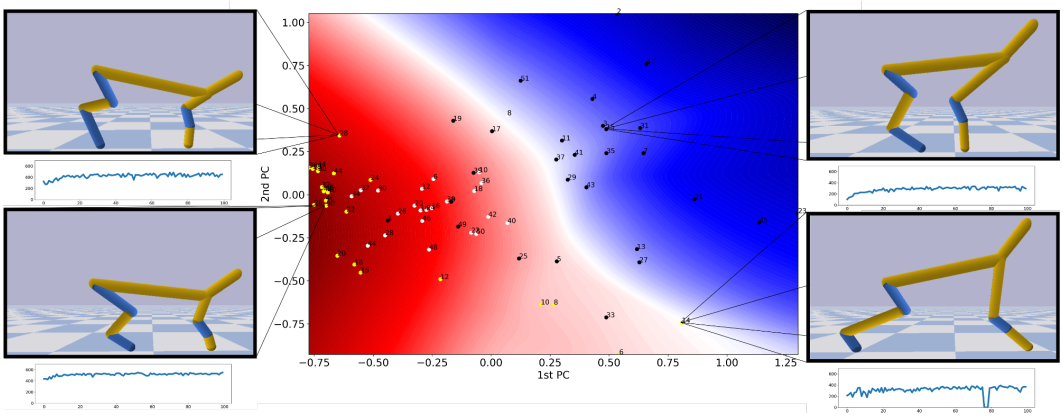


Figure 6: First two principal components of the six dimensional design space of Half-Cheetah as computed with PCA. Colours indicate the Q-value given by the critic on a batch of 256 start states after 50 evaluated designs, with red indicating regions of higher expected reward, and blue the regions of low expected reward. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design. Numbers indicate the order in which the designs were chosen for reinforcement learning.

learning in the Half-Cheetah task. We can see, for example, that a shorter second segment of the back leg and as well as a shorter first segment of the front leg seems to be desirable.

6 Conclusion

In this paper, we study the problem of data-efficiently co-adapting morphologies and behaviors of robots. Our contribution is a novel algorithm, based on recent advances in deep reinforcement learning, which can better exploit previous trials to estimate the performance of morphologies and behaviors before testing them. As a result, our approach can drastically reduce the number of morphology designs tested (and their eventual manufacturing time/cost). Experimental results on 4 simulated robots show strong performance and a drastically reduced number of design prototypes, with one robot requiring merely 50 designs compared to the 24,177 of the baseline – that is about 3 orders of magnitudes less data. The unparalleled data-efficiency of our approach opens exciting venues towards the use in the real world of robots that can co-adapt both their morphologies and their behaviors to more efficiently learning to perform the desired tasks with minimal expert knowledge. In future work, we aim to demonstrate the capabilities of this algorithm on a robot in the real world.

Acknowledgments

We thank Akshara Rai for the valuable discussions during the early stages of this research, as well as for testing the early implementations of the Daisy hexapod simulation thoroughly. Furthermore, we thank Ge Yang for his support to run additional simulations when they were needed. Finally, we thank the anonymous reviewers for their helpful comments.

References

[1] R. C. Bertossa. Morphology and behaviour: functional links in development and evolution introduction. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 366:2056–68, 07 2011. doi:10.1098/rstb.2011.0035.

[2] T. F. Nygaard, C. P. Martin, E. Samuelsen, J. Torresen, and K. Glette. Real-world evolution adapts robot morphology and control to hardware limitations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 125–132. ACM, 2018.

[3] S. Seok, A. Wang, M. Y. M. Chuah, D. J. Hyun, J. Lee, D. M. Otten, J. H. Lang, and S. Kim. Design principles for energy-efficient legged locomotion and implementation on the mit cheetah robot. *Ieee/asme transactions on mechatronics*, 20(3):1117–1129, 2014.

[4] R. M. Alexander. Walking and running: Legs and leg movements are subtly adapted to minimize the energy costs of locomotion. *American Scientist*, 72(4):348–354, 1984.

[5] A. Jansen, K. S. Luck, J. Campbell, H. B. Amor, and D. M. Aukes. Bio-inspired robot design considering load-bearing and kinematic ontogeny of chelonioidea sea turtles. In *Conference on Biomimetic and Biohybrid Systems*, pages 216–229. Springer, 2017.

[6] K. Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994.

[7] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9798–9805. IEEE, 2019.

[8] K. S. Luck, J. Campbell, M. A. Jansen, D. M. Aukes, and H. B. Amor. From the lab to the desert: Fast prototyping and learning of robot locomotion. In *2017 Robotics: Science and Systems, RSS 2017*. MIT Press Journals, 2017.

[9] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[10] F. Corucci, M. Calisti, H. Hauser, and C. Laschi. Novelty-based evolutionary design of morphing underwater robots. In *Proceedings of the 2015 annual conference on Genetic and Evolutionary Computation*, pages 145–152. ACM, 2015.

[11] J. C. Zagal, J. Ruiz-del Solar, and P. Vallejos. Back to reality: Crossing the reality gap in evolutionary robotics. In *IAV 2004 the 5th IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal*, 2004.

[12] S. Koos, J.-B. Mouret, and S. Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2012.

[13] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974, 2000.

[14] D. Ha. Reinforcement learning for improving agent design. *arXiv preprint arXiv:1810.03779*, 2018.

[15] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[16] T. Liao, G. Wang, B. Yang, R. Lee, K. Pister, S. Levine, and R. Calandra. Data-efficient learning of morphology and controller for a microrobot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2488–2494, 2019. doi:10.1109/ICRA.2019.8793802.

[17] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[18] M. R. Bonyadi and Z. Michalewicz. Particle swarm optimization for single objective continuous space problems: a review, 2017.

[19] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Oct 1995. doi:10.1109/MHS.1995.494215.

[20] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pages 312–317. IEEE, 1996.

[21] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. *Artificial Life*, 11:329, 2008.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1856–1865, 2018.

[23] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[24] HEBI Robotics. Hebi robotics x-series hexapod data sheet. `http://docs.hebi.us/resources/kits/assyInstructions/A-2049-01_Data_Sheet.pdf`, 2019. Accessed: 06.07.2019.

[25] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

7 Appendix

7.1 Simulation Environments

This section states a short description of each task simulated in PyBullet [23]:

Half-Cheetah (17, 6, 6) The half cheetah task has an 17 dimensional state space consisting of joint positions, joint velocities, horizontal speed, angular velocity, vertical speed and relative height. Actions have six dimensions and are accelerations of joints. The original reward function used in PyBullet was adapted to be design independent and is given by $r(\mathrm{s}) = \max(\frac{\Delta x}{10}, 0)$ where Δx is the horizontal speed to encourage forward motion. The continuous design vector is a scaling factor of the original leg lengths of Half-Cheetah: $(\xi_1 \cdot 0.29, \xi_2 \cdot 0.3, \xi_3 \cdot 0.188, \xi_4 \cdot 0.29, \xi_5 \cdot 0.3, \xi_6 \cdot 0.188)$. The dimensions of the design vector are in the interval $\xi_i \in [0.8, 2.0]$.

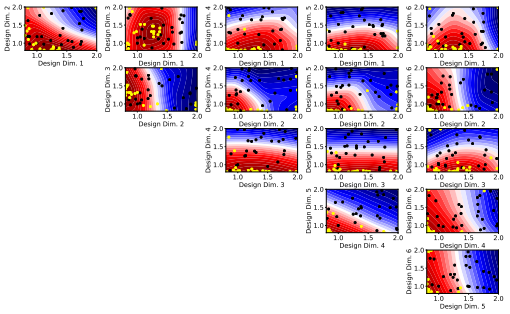
Walker (17, 6, 6) Similar to the Half-Cheetah task, the state space of the Walker task is given by joint positions, joint velocities, horizontal speed, angular velocity, vertical speed and relative height and has 16 dimensions. The two legs of Walker are controlled through acceleration with a six dimensional action. Again, the original reward was adapted to be design agnostic. The term encouraging maximum height of the torso of walker was replaced by two terms favouring vertical orientation y_{rot} of the torso and reaching a minimal height h_{torso} of 0.8. The full reward function is given by $\mathrm{r(s)} = \frac{1}{10}\left((h_{\text{torso}} > 0.8) \cdot (\max(\Delta x, 0) + 1) - \parallel y_{\text{rot}} \parallel_2 0.1\right)$. The design vector is a scaling factor of the leg and foot lengths of the Walker agent: $(\xi_1 \cdot 0.45, \xi_2 \cdot 0.5, \xi_3 \cdot 0.2, \xi_4 \cdot 0.45, \xi_5 \cdot 0.5, \xi_6 \cdot 0.2)$. Each design dimension lies in the interval $\xi_i \in [0.5, 1.5]$.

Hopper (13, 4, 5) In the planar Hopper task a one-legged agent has to learn jumping motions in order to move forward. The state space of this task has thirteen dimensions and four dimensions in the action space. We use the same reward function as for the Walker task with $\mathrm{r(s)} = \frac{1}{10}\left((h_{\text{torso}} > 0.8) \cdot (\max(\Delta x, 0) + 1) - \parallel y_{\text{rot}} \parallel_2 0.1\right)$. In addition to the length of the four movable leg segments, the length of the nose-like feature of walker is an additional design parameter, here ξ_1. The full design vector is given by $\xi = (\xi_1 \cdot 0.7, \xi_2 \cdot 0.15, \xi_3 \cdot 0.33, \xi_4 \cdot 0.32, \xi_5 \cdot 0.25)$ with $\xi_{2:5}$ being the length of each movable segment from pelvis to foot. The design parameters were bounded with $\xi_1 \in [0.5, 4.0]$ for the length of the nose and $\xi_{2:5} \in [0.5, 2.0]$ for all leg lengths.

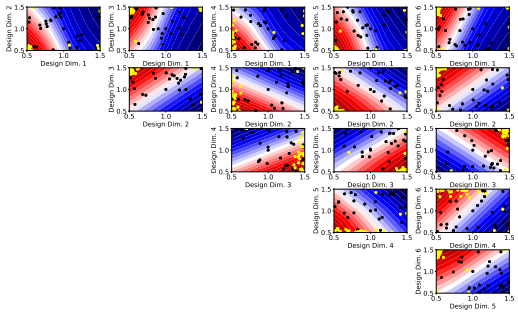
Daisy Hexapod (43, 18, 9) For a preliminary study and to evaluate whether the proposed method is suitable for real world applications, a simulation of the six-legged Daisy robot by HEBI Robotics [24] was created in PyBullet. Each leg of the robot has three motors and hence the action space has 18 dimensions. The state space has 43 dimensions and consists of joint positions, joint velocities, joint accelerations, the velocity of the robot in x/y/z directions and the orientation of the robot in Euler angles. The task of the robot is to learn to walk forward while keeping its orientation and thus the reward function is given by $r(s) = \frac{\max(\Delta y, 0)}{0.066} - 0.25 \cdot \mathrm{diff}(e_{\text{original}}, e_{\text{current}})$, with Δy being the dislocation along the y-axis, the direction the robot faces at initialization, and $\mathrm{diff}(e_{\text{original}}, e_{\text{current}})$ representing the angle between the original and current orientation in quaternions. The design vector consists of two parts: leg lengths, and movement range of the motors at the base of the legs. All parameters are symmetric between the left and right side of the robot. The leg lengths are in $\xi_{1:6} \in [0.12, 0.5]$ for the two leg segments of each leg. Additionally, we allowed the algorithm to optimize the movement range of the first out of three motors on each leg. The base motors are restricted in movement between $(-0.35 + \xi_{7:9}, 0.35 + \xi_{7:9})$ radians with the design parameters $\xi_{7:9} \in [-0.2, 0.2]$.

7.2 Visualization of Design Space

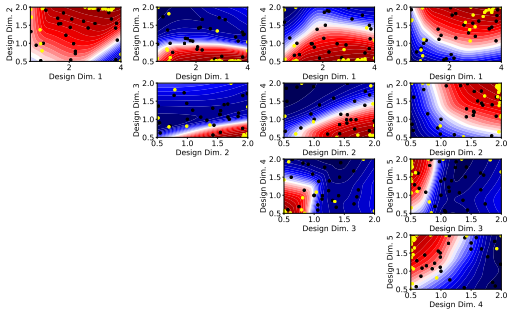
Because we can query the proposed objective function from eq. 6, we are able to visualize the cost landscape of each task. Figure 7 shows the design spaces of the three standard PyBullet tasks Half-Cheetah, Walker and Hopper after 50 designs evaluated in simulation. Each single plot shows the design landscape of two dimensions while the other dimensions were held fix with stated design vectors as well as the location of design chosen by the proposed method (yellow) and designs chosen randomly for exploration (black). The cost landscape of the more complex Daisy Hexapod task is shown in figure 8.



(a) Design Space of Half Cheetah for $\xi = (1.27, 0.87, 1.17, 0.84, 0.80, 0.82)$



(b) Design space of Walker for $\xi = (0.50, 0.52, 1.35, 0.66, 0.57, 1.49)$



(c) Design Space of Hopper for $\xi = (3.41, 1.95, 0.52, 0.52, 1.90)$

Figure 7: The visualized cost landscape of the design spaces. A batch of 256 start states was used.

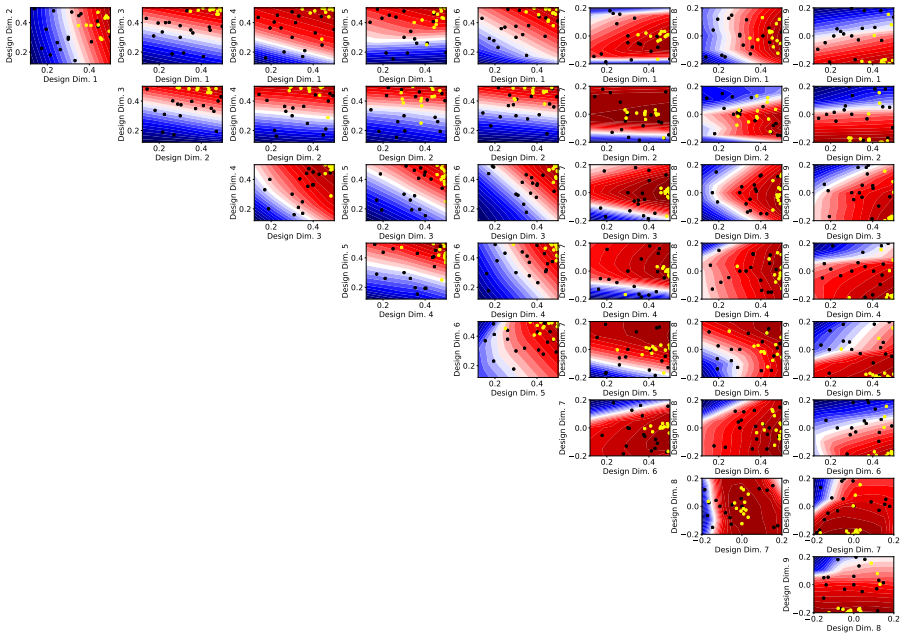
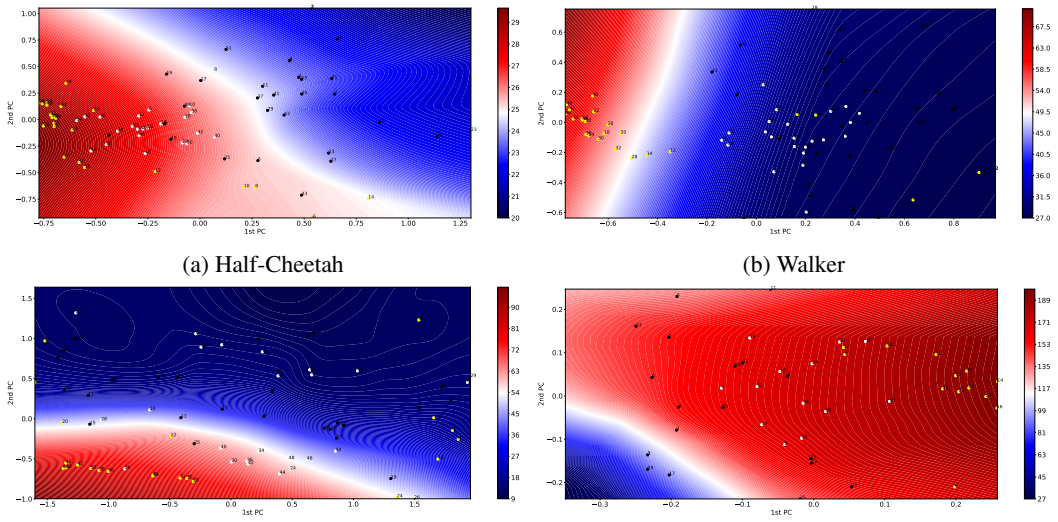


Figure 8: Design space of the Daisy Hexapod for $\xi = (1.27, 0.87, 1.18, 0.84, 0.80, 0.83)$. A batch of 256 start states was used. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design.



(a) Half-Cheetah (b) Walker

(c) Hopper (d) Daisy Hexapod

Figure 9: First two principal components of each design space as computed with PCA. Colours indicate the Q-value given by the critic on a batch of 256 start states after 50 (30 for the Daisy Hexapod) evaluated designs, with red indicating regions of higher expected reward, and blue the regions of low expected reward. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design. Numbers indicate the order in which the designs were chosen for reinforcement learning.

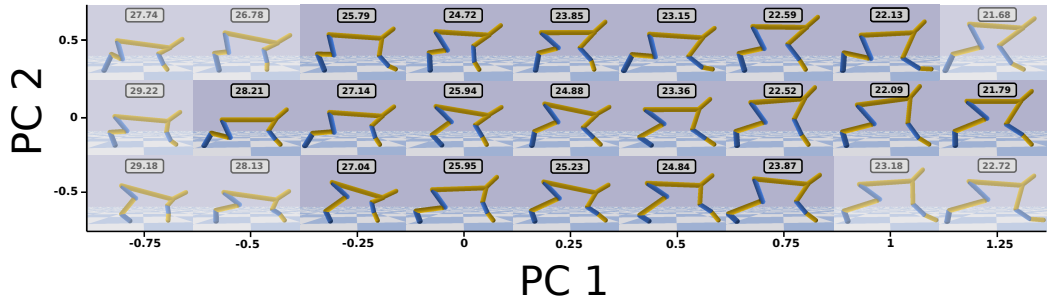


Figure 10: A selection of designs for Half-Cheetah, generated from the principal components in Fig. 9a. Designs which are outside of the bounds set for the design space are reduced in opacity. Each design is evaluated with the objective function stated in Eq. 6.

7.3 Visualization of the Latent Design Space

For a better understanding of the cost landscape a low dimensional design space was computed with principal component analysis. Figure 9 shows the low-dimensional projection of the design space as well as the designs ξ_{Opt} chosen by the proposed method (ywllow) and randomly selected designs for exploration (black) In white designs chosen by the optimization via simulation method are shown. We can see that the convergence rate of *optimization via simulation* appears to be slower than our method. To see what properties of the design lead to a better performance we visualized the design along the two principal components (Fig. 10). We can see that just longer leg do not appear to lead automatically to better performance but shorter front legs and slightly longer back legs do.

7.4 Evolution of Walker

Figure 11 shows the evolution of designs with the proposed objective function. We can see that the start states are random and lead to different poses of Walker, sometimes falling for- or backwards.

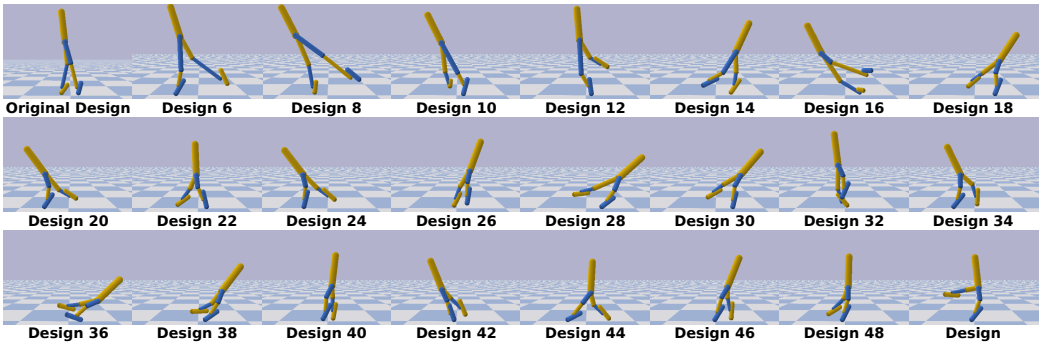


Figure 11: Designs ξ_{Opt} selected by the proposed method for the Half-Cheetah task.

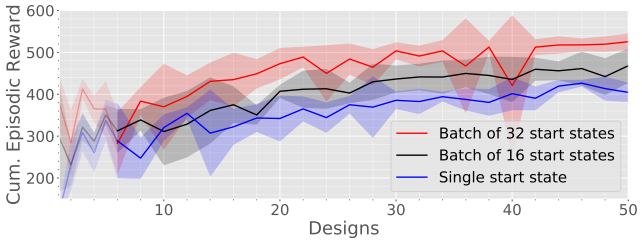


Figure 12: Evaluation of different batch sizes used in Eq. (6) on the Half-Cheetah task.

It can be seen that while shorter legs seem desirable, the larger the foot length the better the performance.

7.5 Using CMA-ES for Evolutionary Design Optimization

As proposed in the work of David Ha [14] we evaluated our approach against two approaches using CMA-ES (Fig. 15) and OpenAI-ES (see main text) in an evolutionary manner for the optimization of the designs. For this experiment, we let CMA-ES create a population of design candidates and evaluated them in the simulator. We then executed exactly one update iteration of CMA-ES and used the best design found in the reinforcement learning loop. Figure 15 shows that this method is outperformed by the approach proposed in this paper. The proposed method uses the Q-function for design evaluations during the design optimization phase and executes a number of update iterations before selecting the best design for the reinforcement learning loop.

7.6 Design Exploration Strategies

We alternate between design exploration and exploitation to increase the diversity of explored designs, improve generalization capabilities of the critic and avoid an early convergence to regions of the design space. Therefore, every time we find an optimal design during the design optimization process with the objective function (Eq. 6) and conclude the subsequent reinforcement learning process, we

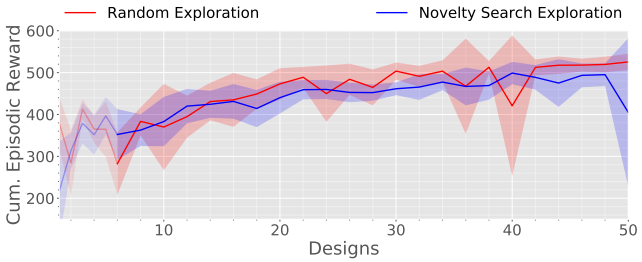


Figure 13: Evaluation of using novelty search or random selection as exploration mechanism during design optimization.

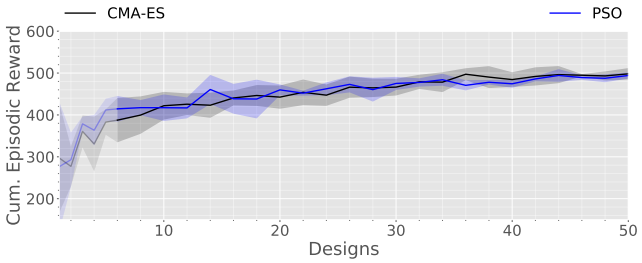


Figure 14: Comparison of CMA-ES and PSO when using rollouts from the simulator to optimize designs.

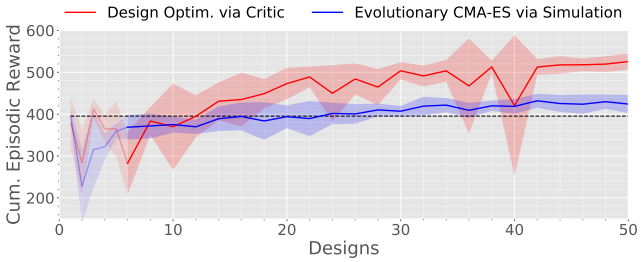


Figure 15: The plot shows, in blue, the use of CMA-ES used in the same manner as in [14]. In each iteration lambda (here nine) design candidates are generated and evaluated in simulation. Then one iteration of CMA-ES is executed and the RL loop is executed on the best design found. Finally, a new population is generated and the next iteration of CMA-ES is executed. The dotted line shows the average best performance on the initial design.

next choose one design using the exploration strategy. To this end, we implemented two different approaches: sampling new designs 1) randomly, and 2) using Novelty search [21]. Novelty search is an exploration strategy in which the objective maximizes distance to the closest neighbours. The objective function is given by

$$\max_{\xi} \frac{1}{m} \sum_{\tilde{\xi} \in \mathrm{NN}(\xi, \bar{\Xi})} \| \xi - \tilde{\xi} \|_2, \tag{7}$$

where the function $\mathrm{NN}(\xi, \bar{\Xi})$ returns the m nearest neighbors of a design ξ from the set $\bar{\Xi}$ of chosen designs so far. This set includes only designs which were selected for evaluation in the real world or simulation, i.e., were handed over to the reinforcement learning algorithm as ξ_{Opt} (Fig. 1b). Experiments showed that using novelty search for exploration did not yield an advantage over random selection of designs (Fig. 13).

7.7 Performance of Optimization Algorithms for Design Optimization

Since we had to reduce the number of simulations considerably during the design optimization stage, we also evaluated the performance between Particle Swarm Optimization (PSO) and Covariance Matrix Adaptation-Evolution Strategy (CMA-ES). However, we could not find a significant difference in performance (Fig. 14).

7.8 About the Use of Batches of Start States for the Evaluation of Design Candidates

We evaluated the importance of evaluating the objective function (Eq. 6) over a batch of start states. Figure 12 shows the use of a single start state s_0, using a batch of 16 and 32 start states in the objective function presented in Eq. 6. The evaluation shows that averaging the objective function over a number of randomly drawn start states increases the performance of the proposed approach considerably.

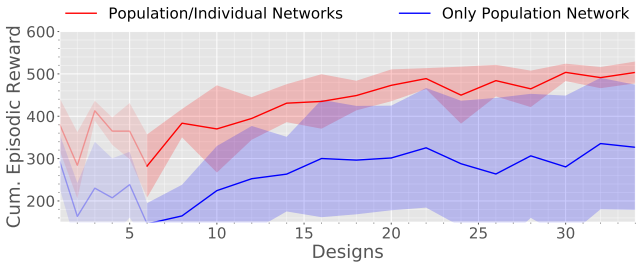


Figure 16: Using only a single *population* network shows a worse performance than using the proposed combination of *population* and *individual* networks. This evaluation was performed on the Half-Cheetah task.

7.9 Evaluating the use of Population and Individual Networks

In a preliminary evaluation we were able to confirm that the use of a single set of *population* networks, instead of using a combination of *population* and *individual* networks, shows a decreased performance (Fig. 16). This shows that the ability of the *individual* networks, to adapt quickly to the current design, is important for the overall performance of the proposed approach.