Symmetry-Aware Robot Design with Structured Subgroups

Heng Dong [1] **Junyu Zhang** [2] **Tonghan Wang** [3] **Chongjie Zhang** [1]

Abstract

Robot design aims at learning to create robots that can be easily controlled and perform tasks efficiently. Previous works on robot design have proven its ability to generate robots for various tasks. However, these works searched the robots directly from the vast design space and ignored common structures, resulting in abnormal robots and poor performance. To tackle this problem, we propose a Symmetry-Aware Robot Design (SARD) framework that exploits the structure of the design space by incorporating symmetry searching into the robot design process. Specifically, we represent symmetries with the subgroups of dihedral group and search for the optimal symmetry in structured subgroups. Then robots are designed under the searched symmetry. In this way, SARD can design efficient symmetric robots while covering the original design space, which is theoretically analyzed. We further empirically evaluate SARD on various tasks, and the results show its superior efficiency and generalizability.

1. Introduction

Humans have been dreaming of creating creatures with morphological intelligence for decades (Sims, 1994a;b; Yuan et al., 2021; Gupta et al., 2021b). A promising solution for this challenging problem is to generate robots with various functionalities in simulated environments (Wang et al., 2019; Yuan et al., 2021), in which robots' functionalities are largely determined by their designs and control policies. Learning control policies for handcrafted robots with fixed designs has been extensively studied in previous works (Schulman et al., 2017; Fujimoto et al., 2018; Huang et al., 2020; Dong et al., 2022). However, as the other critical component, the design of robots has attracted scant attention and achieved limited success in the literature. The field of automatic robot design aims at searching for optimal robot morphologies that can be easily controlled and perform various tasks efficiently. This problem has been a long-lasting challenge, mainly for two reasons: 1) the design space, including skeletal structures and attributes of joints and limbs, is large and combinatorial, and 2) the evaluation of each design requires training and testing an optimal control policy which is often computationally expensive.

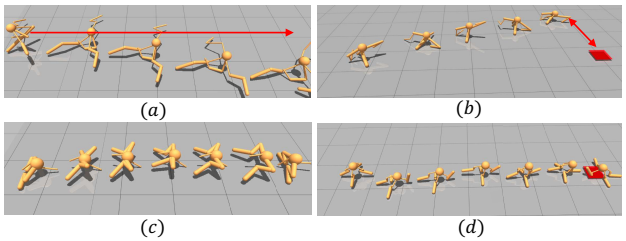

Figure 1. Time-lapse images of robots with different symmetries performing various tasks: (a, c) running forward; (b, d) reaching random goals. Robots designed by prior work (Yuan et al., 2021) do not satisfy any non-trivial symmetry and might be hard to control: (a) the robot deviated from the right direction; (b) the robot missed the goal. Different tasks may require different symmetries: (c) bilateral symmetry is suitable for tasks involving only running forward; (d) radial symmetry for reaching random goals.

For automatic robot design, prior works (Gupta et al., 2021b; Wang et al., 2019) typically adopt evolutionary search (ES) algorithms, where robots are sampled from a large population and learn to perform tasks independently during an iteration. At the end of each iteration, robots with the worst performance are eliminated, and the surviving robots will produce child robots by random mutations to maintain the population. Recently, Yuan et al. (2021) discussed the low sample efficiency problem in ES-based methods, *e.g.*, robots in the population do not share their training experiences and zeroth-order optimization methods such as ES are sample-inefficient for high-dimensional search space (Vemula et al., 2019). They used reinforcement learning (RL) to sample and optimize robot designs by incorporating the design process into the control process and sharing the design and control policies across all robots.

Despite the progress, robots designed by these approaches are intuitively abnormal, empirically hard to control, and

[1]Institute for Interdisciplinary Information Sciences, Tsinghua University [2]Huazhong University of Science and Technology [3]Harvard University. Correspondence to: Heng Dong <drdhxi@gmail.com>, Chongjie Zhang <zhangchongjie@gmail.com>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

ultimately result in poor performance. Examples are provided in the time-lapse images in Figure 1 (a-b), in which robots designed by prior work (Yuan et al., 2021) perform poorly on different tasks. We hypothesize that the underperformance can be attributable to the fact that most prior works directly search for robots in the whole vast design space without exploiting useful structures that can largely reduce the search space.

To verify this hypothesis, in this paper, we explore utilizing *symmetry* as the key characteristic to unveil the structure of the design space and hereby reduce learning complexity. Symmetry is one structure commonly observed in biological organisms (Savriama & Klingenberg, 2011), *e.g.*, bilateral symmetry in flies (Evans & Bashaw, 2010), radial symmetry in jellyfish (Abrams et al., 2015), and spherical symmetry in bacteria (Shao et al., 2017). From the learning perspective, symmetry-aware robot design has two advantages. First, it requires searching for much fewer robot designs. If one design turns out to be unsuitable for the current task, other designs from the same symmetry can be searched less frequently as they are likely to be morphologically and functionally similar. Second, symmetric designs can reduce the degree of control required to learn balancing (Raibert, 1986c;a) as in Figure 1 (a)(c). Prior works noticed the benefits of symmetry (Gupta et al., 2021b; Wang et al., 2019), but only bilateral symmetry was considered. Other tasks may require different symmetries. For example, tasks that involve running in different directions require radial symmetry (Figure 1 (d)). However, none of the previous works explored learning suitable robot symmetries for different tasks.

In this paper, we introduce a novel Symmetry-Aware Robot Design (SARD) framework. Realizing this framework involves two major challenges. The first challenge is how to represent symmetries and how to find optimal symmetry. To consider a wide range of symmetries while also avoiding extra learning complexity, we propose to use the subgroups of dihedral group (Gallian, 2021) to represent symmetries. Each subgroup represents a kind of symmetry and a symmetric space. The trivial subgroup containing only the identity group element is exactly the original design space. To find the optimal symmetry efficiently, we utilize the group structures and adopt a simple local search algorithm applied in the structured subgroups for smoothly changing symmetry types to alleviate gradient conflict problem (Liu et al., 2021). The second challenge is how to design robots that satisfy a given symmetry. We propose a novel plug-and-play symmetry transformation module to map any robot design into a given symmetric space. We also provide theoretical analysis to verify that the transformed robot designs are in the given symmetric space and this module can cover the whole space.

We evaluate our SARD framework on six MuJoCo (Todorov et al., 2012) tasks adapted from Gupta et al. (2021b). SARD significantly outperforms previous state-of-the-art algorithms in terms of both sample efficiency and final performance. Performance comparison and the visualization of the symmetry learning process strongly support the effectiveness of our symmetry searching and transformation approaches. Our experimental results highlight the importance of considering various symmetries in robot design.

2. Related Works

Modular RL. Robot design problem typically requires controlling robots with changeable morphologies, where the state and action spaces are incompatible across robots. This new issue cannot be tackled by traditional monolithic policies used in single-agent RL, but fortunately, modular RL that uses a shared policy to control each actuator separately holds the promise to solve it. Most prior works in this field represent the robot's morphology as a graph and use GNNs or message-passing networks as policies (Wang et al., 2018; Pathak et al., 2019; Huang et al., 2020). All of these GNN-like works show the benefits of modular policies over a monolithic policy in tasks tackling different morphologies. Recent works also proposed to use Transformer (Vaswani et al., 2017) to represent policies to overcome the difficulty of message-passing in complex morphologies to further improve performance (Kurin et al., 2020; Gupta et al., 2021a; Dong et al., 2022). In this paper, we build our method based on GNN-like policies for a fair comparison with the previous state-of-the-art baseline Transform2Act (Yuan et al., 2021) but note that our method is a plug-and-play module that can be used to any modular RL policies.

Robot Design. Automatic robot design problem aims at searching for robots that can be easily controlled and can perform various tasks efficiently. A line of works in this field focused only on designing the attributes of robots while ignoring skeletal structures (Ha et al., 2017; Yu et al., 2018; Ha, 2019), which is limited in designing robot morphology. Another line of work considers both attribute design and skeleton design, which is known as combinatorial design optimization. Previous works mainly utilize evolutionary search (ES) algorithms for combinatorial design optimization (Sims, 1994a; Nolfi & Floreano, 2000; Auerbach & Bongard, 2014; Cheney et al., 2018; Jelisavcic et al., 2019; Zhao et al., 2020), which require robots with different structures to perform certain task independently and do not share experiences between robots. This will result in severe sample inefficiency and may require thousands of CPUs to finish one experiment (Gupta et al., 2021b). Recently, Yuan et al. (2021) discussed this issue and used RL to optimize robot designs by incorporating design procedure into the decision-making process and formulating design optimization as learning a conditional policy. This method has shown

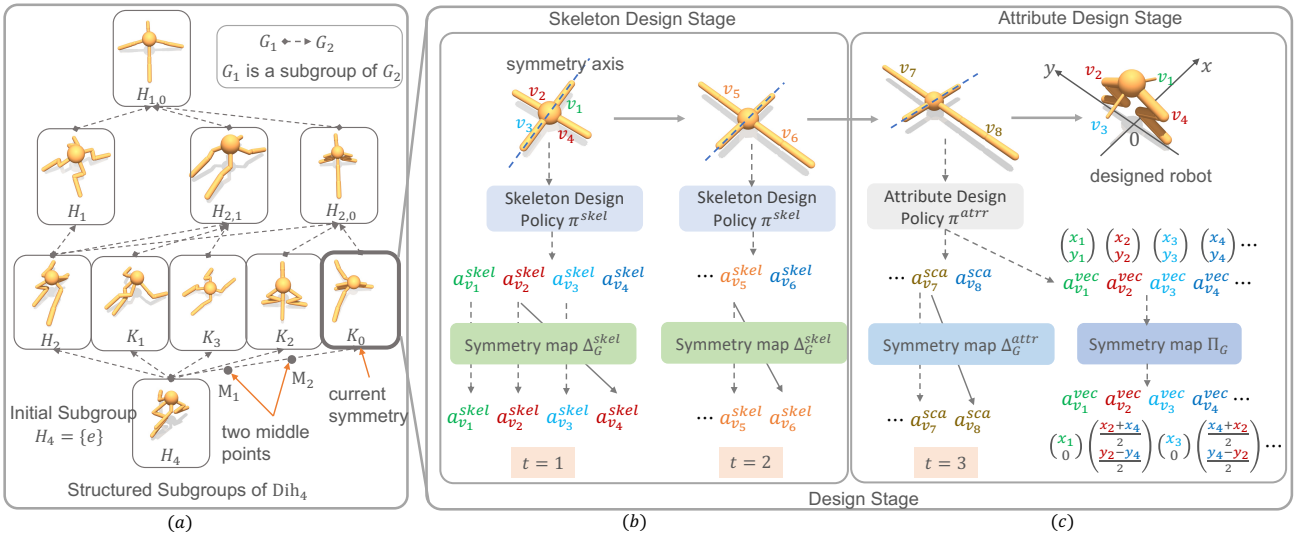

Figure 2. Symmetry-aware robot design framework. (a) Search for the optimal symmetry in structured subgroups; (b) design the skeletal structures of the robot; (c) design the attributes of joints and limbs of the robot. Joints in the same color are in the same orbit in (b-c).

great sample efficiency. However, none of the above studies have explored the structure of the design space and searched for robots directly from the vast design space, which may end up with abnormal robots with poor performance. As for works that considered symmetry in robot design, NGE (Wang et al., 2019) only considered bilateral symmetry, while our work considers a wide range of symmetries for various tasks.

Symmetry in Real-World Creatures. Our method that utilizes symmetry as the classification characteristic shares a similar and intriguing principle in real-world creatures, which might have been developed millions of years ago according to fossil evidence (Evans et al., 2020). Certain symmetries are maintained by natural selection pressures and deviation from perfect symmetry is negatively correlated with species fitness (Enquist & Arak, 1994). Raibert (1986b) showed that during a series of bouncing and ballistic motions, symmetry contributes to achieving more complicated running behaviors, *e.g.*, reciprocating leg symmetry is essential to make a quadruped gallop, symmetry of wings can help reduce energy expenditure in flight (Polak & Trivers, 1994). Our experiments in Section 5 also showed similar conclusions that symmetry can help reduce control costs.

3. Preliminaries

In this section, we introduce some background knowledge and notations necessary to present our method.

Problem Settings. We aim to search for a robot design D from a design space $\mathcal{D}$ to finish different tasks efficiently. A design D includes the robot's skeletal structure, limb-specific attributes, and joint-specific attributions (*e.g.*, limb length, size, and motor strength). Formally, a robot design can be represented by a graph $D = (V, E, Z)$, where each node $v \in V$ represents a joint in robot morphology and each edge $e = (v_i, v_j) \in E$ represents a limb connecting two joints v_i, v_j. Each $z \in Z$ is the attributes of a joint and the limb attached to this joint, including scalar values and vector values, and $|V| = |Z|$.

The designed robot then learns control policies to finish tasks by reinforcement learning (RL) algorithms. RL formulates a control problem as an infinite-horizon discounted Markov Decision Process (MDP), which is defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \rho_0, R, \gamma)$, where these items represent state set, action set, transition dynamics, initial state distribution, reward function and discounted factor, respectively. For a fixed robot design and task, the objective of RL is to learn a control policy π^C that maximizes the expected total discounted reward: $J(\pi^C) = \mathbb{E}_{\pi^C}[\sum_t \gamma^t r_t]$.

Given that the robot design is changeable during learning, we condition the original transitions dynamics and reward function in $\mathcal{M}$ on design D, yielding a more general transition dynamics $\mathcal{T}(s_{t+1}|s_t, a_t, D)$ and reward function $R(s_t, a_t, D)$. Hence the reinforcement learning objective is also conditioned on D, and we optimize $J(\pi^C, D)$. Now the design optimization problem can be naturally formulated as a bi-level optimization problem (Sinha et al., 2017; Colson et al., 2007): $D^* = \arg\max_{D \in \mathcal{D}} J(\pi^{C,*}, D)$ s.t. $\pi^{C,*} = \arg\max_{\pi^C} J(\pi^C, D)$ where the inner optimization problem is typically solved by RL but is especially computationally expensive for changeable robot designs. The outer optimization problem can be solved by evolutionary algorithms (Wang et al., 2019; Gupta et al., 2021b; Sims, 1994a) or RL (Yuan et al., 2021). In this paper, we follow Transform2Act to use RL as the outer problem solver for its

efficiency compared with evolutionary algorithms.

Group Theory. We use group theory to represent different types of symmetries. Here we briefly introduce some notations and please refer to Appendix A.2 for details. A *group* G is a set with a binary operation such that it has four basic properties, *i.e.*, associativity, closure, the existence of an identity ($e \in G$), and the existence of an inverse of each element ($g^{-1} \in G, \forall g \in G$) (Gallian, 2021). If a subset H of G is also a group under the same operation of G, we refer to it as a *subgroup* of G and denote it by $H < G$.

A *group action* is a function $G \times X \rightarrow X$ of group G on some space X, and it satisfies $ex = x$, $(gh)x = g(hx), \forall g, h \in G, x \in X$. For an element $g \in G$, we define a *transformation function* $\alpha_g^X : X \rightarrow X$ given by $x \mapsto gx$, which can be interpreted as a transformation of the point x under group element g. For example, if x is a robot, g could be a rotation transformation of the robot along the z-axis passing through the torso of the robot. The *orbit* of a point $x \in X$ is the set of all its transformation under G, denoted by $\mathcal{O}_G(x) = \{\alpha_g^X(x) | \forall g \in G\}$. An important property is that X can be partitioned by orbits: $X = \bigcup_{x \in X} \mathcal{O}_G(x)$.

Dihedral Group. The dihedral group is a finite discrete group containing rotation and reflection transformations. A dihedral group $\text{Dih}_n (n \geq 3)$ can be generated by rotation transformation ρ (counterclockwise rotation by $360°/n$) and reflection transformation π (reflection along x-axis). Concretely, $\text{Dih}_n = \{\rho_k, \pi_{k-1} | k = 1, 2, \cdots, n\}$, where $\rho_k = \rho^k$, $\rho_0 = \rho_n = e$ and $\pi_k = \rho^k \pi$. Each group element g has multiple representations. We consider permutation representation P_g and matrix representation M_g in this paper. Considering the designed robot in Figure 2(c) and taking $\pi_0 \in \text{Dih}_4$ as an example, P_{π_0} exchanges joint v_2, v_4 and M_{π_0} reflects their coordinates along x-axis.

In this paper, we use the subgroups of the dihedral group to represent various symmetries. The subgroups of dihedral groups have three types: (1) $H_d = \langle \rho_d \rangle$, where $1 \leq d \leq n$ and $d | n$ (n is divisible by d); (2) $K_i = \langle \pi_i \rangle$, where $0 \leq i \leq n-1$; and (3) $H_{k,l} = \langle \rho_k, \pi_l \rangle$, where $0 \leq l < k \leq n-1$ and $k | n$. The group structure of Dih_4 is shown in Figure 2(a). For more details, please refer to Appendix A.3.

4. Method

In this section, we present our Symmetry-Aware Robot Design (SARD) scheme that utilizes symmetry as a characteristic to exploit the structure of the design space and reduce the learning complexity. As in previous works on robot design, our learning framework consists of design searching and control policy learning. The focus of this paper is to incorporate symmetry into design search. To this end, our method consists of two major components: (1) searching for the optimal symmetry G (Section 4.2) and (2) learning robot design under the given symmetry G (Section 4.1).

4.1. Learning Robot Design under a Given Symmetry

Algorithm 1 SARD: Symmetry-Aware Robot Design

input group Dih_n; number of intervals between symmetries K; symmetry sampling exploration rate ϵ;
output symmetry G; design policy π^D, control policy π^C;
Initialize π^D and π^C;
(Sec.4.2): Initialize symmetry $G \leftarrow \{e\}$, value dict for symmetries $V \leftarrow \mathbf{0}$;
while not reaching max iterations **do**
  Memory $\mathcal{M} \leftarrow \emptyset$;
  while $\mathcal{M}$ not reaching batch size **do**
    $D \leftarrow$ initial robot design;
    while in Design Stage **do**
      Sample design actions from π^D;
      (Sec. 4.1): Transform design actions with symmetry maps $\Delta_G^{skel}, \Delta_G^{atrr}, \Pi_G$ (Eq. (1) to (3));
      Apply design actions to modify design D and store them to $\mathcal{M}$;
    end while
    Use π^C to control current robot design D and store trajectories to $\mathcal{M}$.
  end while
  Update π^C, π^D with PPO using samples in $\mathcal{M}$;
  (Sec.4.2): Update $V(G) \leftarrow$ mean episode rewards in $\mathcal{M}$ and sample a new symmetry from neighbors $G \leftarrow$ *Neighbor*(G) with ϵ-greedy using V;
end while

We now describe how to design robots that satisfy a given symmetry G. Symmetry refers to an object that is invariant under some transformations, and every subgroup of the dihedral group Dih_n represents a type of symmetry, *e.g.*, the designed robot in Figure 2(c) is invariant under the symmetry represented by K_0 which contains a reflection transformation along the x-axis and the identity transformation. As the trivial subgroup that contains only identity transformation is considered, our method also covers the original design space. To formally represent symmetries, we define a G-symmetric property as follows:

Definition 4.1. A robot design $D = (V, E, Z)$ is G-symmetric if the robot is invariant under the transformation of group G. Specifically, $\forall g \in G$, we have $D_g = D$, where $D_g \triangleq (V_g, E_g, Z_g)$, $V_g \triangleq \{\alpha_g^V(v) | v \in V\}, E_g \triangleq \{\alpha_g^E(e) | e \in E\}, Z_g \triangleq \{\alpha_g^Z(z) | z \in Z\}$.

where $\alpha_g^V, \alpha_g^E, \alpha_g^Z$ transform the design with group element g and formal definitions are given in Appendix A.4. All G-symmetric robots constitute the G-symmetric space. Using the transformation function $\alpha_g^V(v)$, we can define the orbit of v: $\mathcal{O}_G(v) = \{\alpha_g^V(v) | \forall g \in G\}$. For the designed

robot in Figure 2(c), joints v_2, v_4 are in the same orbit, *i.e.*, $\mathcal{O}_{K_0}(v_2) = \mathcal{O}_{K_0}(v_4) = \{v_2, v_4\}$.

Most RL- or ES-based methods in robot design can be roughly divided into two stages: (1) **Design Stage**, where a new robot with design D is generated from an initial design D_0 by a design policy π^D or mutated by a random design policy π^D; (2) **Control Stage**, where the generated robot interacts with the environment using control policy π^C.

The **Design Stage** can be further divided into two sub-stages: ***Skeleton Design Stage*** and ***Attribute Design Stage***, which generate the skeletal graph (V, E) and attributes Z of the design D, respectively. Thus the design policy π^D will also be separated into two sub-policies: skeleton design policy π^{skel} and attribute design policy π^{attr}, *i.e.*, π^D=(π^{skel}, π^{attr}).

Every episode starts with the ***Skeleton Design Stage*** by initializing a G-symmetric (where G<Dih_n) initial design: D_0=(V=$\{v_1, v_2, \cdots, v_n\}$, E=$\emptyset$, Z=$\mathbf{0}$), *i.e.*, the first robot in Figure 2(b). In each time step, each joint $v \in V$ selects a discrete skeleton design action $a_v^{skel} \in \mathcal{A}^{skel}$ based on the skeleton design policy $\pi^{skel}(a_v^{skel}|D, G)$. Here D is the current design, G is the given symmetry type and $\mathcal{A}^{skel}$ is the skeleton action set including three actions: 1) AddJoint: joint v will add a child joint u to the skeletal graph; 2) DelJoint: joint v will remove itself from the skeletal graph if it has no child joints; 3) NoChange: no changes will be made to joint v. All joints share the same action space. Then the robot design will transit to D' and this sub-stage will last for N^{skel} steps. To ensure that the designed robot is G-symmetric, we propose to keep the robot symmetric in each time step by making joints in the same orbit choose the same skeleton action, *e.g.*, they could all choose the action selected by the joint with the smallest index. Therefore, joints v_2, v_4 in Figure 2(b) will both use the action selected by v_2. This can be realized by a symmetry map Δ_G^{skel} : $\mathcal{A}^{skel} \to \mathcal{A}^{skel}$, defined as follows:

$$\Delta_G^{skel}(a_v^{skel}) = a_{\mu(\mathcal{O}_G(v))}^{skel}, \quad \forall v \in V, \tag{1}$$

where $\mathcal{O}_G(v)$ is v's orbit under subgroup G, and μ : $\mathcal{P}(V) \to V$ returns the element with the smallest index in $\mathcal{O}_G(v)$. $\mathcal{P}(V)$ is the power set of V. We use $\{\Delta_G^{skel}(a_v^{skel})\}_{v \in V}$ to generate a robot. In this way, the new design D' is also G-symmetric.

At the ***Attribute Design Stage***, each joint $v \in V$ chooses a continuous attribute action $a_v^{attr} \in \mathcal{A}^{attr}$ for v and the limb attached to v based on attribute design policy $\pi^{attr}(a_v^{attr}|D, G)$. Here $\mathcal{A}^{attr}$=$\mathcal{A}^{sca} \times \mathcal{A}^{vec}$ is the attribute action set containing scalar values $a_v^{sca} \in \mathcal{A}^{sca}$ (motor strength, limb size, *etc.*) and vector values $a_v^{vec} \in \mathcal{A}^{vec}$ (limb offset, *etc.*). Then the robot design will transit to D' and this sub-stage will last for N^{attr} steps. Like the first stage, we keep the robot symmetric in each time step to ensure that D' is G-symmetric. For the scalar values, we define a similar symmetry map like Equation (1), $\Delta_G^{attr} : \mathcal{A}^{sca} \to \mathcal{A}^{sca}$:

$$\Delta_G^{attr}(a_v^{sca}) = a_{\mu(\mathcal{O}_G(v))}^{sca}, \quad \forall v \in V. \tag{2}$$

Therefore, joints v_7, v_8 in Figure 2(c) will both adopt the action selected by v_7.

As for the vector values, we propose a novel symmetry map to project them to G-symmetric space. For simplicity, here we assume $a_v^{vec} = (x, y)^\top$, where $x, y \in \mathbb{R}$, only includes one coordinate here and z-value can be learned or set to a default value. The vector value actions of all joints form a matrix: $c = (a_1^{vec}, a_2^{vec}, \cdots, a_{|V|}^{vec})$. The coordinate c should be invariant under transformation $g \in G$. Directly solving this problem is challenging, and we propose a novel symmetry map $\Pi_G : \mathbb{R}^{2\times|V|} \to \mathbb{R}^{2\times|V|}$ that can project any c into G-symmetric space:

$$\Pi_G(c) = \frac{1}{|G|} \sum_{g \in G} M_g c P_{g^{-1}}. \tag{3}$$

where M_g and P_g are the matrix and permutation representations of g. This property is verified in this theorem:

Theorem 4.2. *The projected vector values $\Pi_G(c)$ defined in Equation (3) are G-symmetric and, if c is already G-symmetric, then $\Pi_G(c) = c$.*

This theorem implies two facts: (1) realizability: the projected vector values are G-symmetric; and (2) completeness: $\Pi_G(c)$ can cover the whole G-symmetric space. The proof is provided in Appendix A.5 and Figure 2(c) shows the transformation results of K_0, which is also derived in Appendix A.5. It is straightforward to extend one coordinate to multiple coordinates by applying Equation (3) coordinate-wise. Thus, we can use $\Pi_G(c)$ as the vector value of attribute design actions to ensure that D' is G-symmetric. Putting all symmetry maps together, we prove in Appendix A.6 the following theorem:

Theorem 4.3. *The transformed design space by transformations Δ_G^{skel} (Equation (1)), Δ_G^{attr} (Equation (2)) and Π_G (Equation (3)) is equivalent to G-symmetric space.*

In the Design Stage, robots do not interact with the environments and will not receive rewards. The design policy is trained using PPO with rewards from the Control Stage.

The **Control Stage** is the same as the normal robot control problem, except that the control policy $\pi^C(a|s, D)$ is conditioning on the current design. Figure 1(c) shows a trajectory of the designed robots of Figure 2(c) in the Control Stage for Locomotion on Flat Terrain task. To incorporate D in to policies we implement policies π^D and π^C with graph neural networks (Scarselli et al., 2008; Bruna et al., 2013; Kipf & Welling, 2016), which are optimized by Proximal Policy Optimization (PPO) (Schulman et al., 2017), a standard policy gradient method (Williams, 1992).

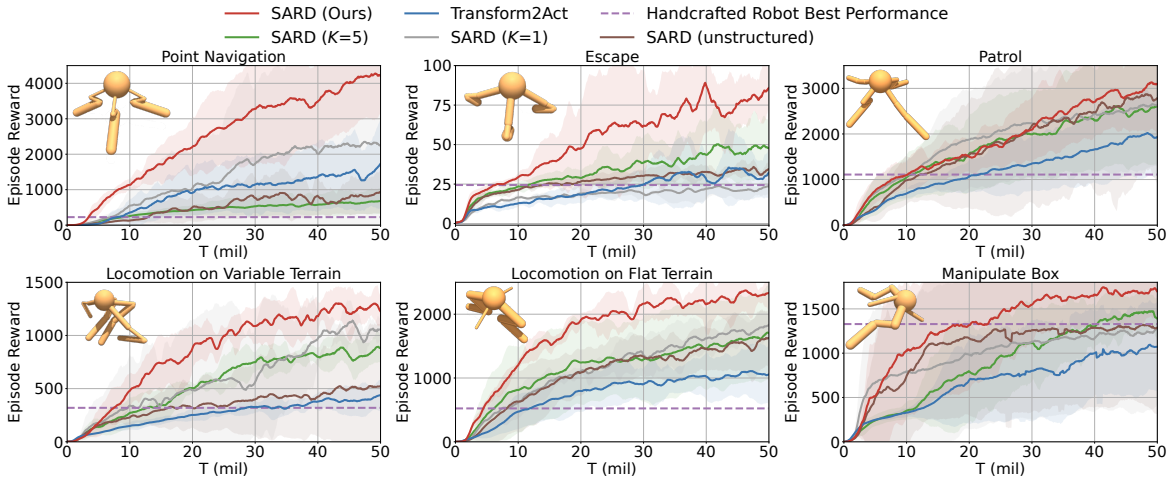

Figure 3. Training performance of SARD compared against baselines and ablations

4.2. Searching for the Optimal Symmetry

We now discuss how to find the optimal symmetry for differing tasks. One simplest way is to sample several subgroups from Dih_n, evaluate them, and choose the subgroup with the highest performance every iteration. However, in this way, the sampled symmetry type in consecutive iterations G_i and G_{i+1} might be dissimilar and thus the designed robots under these two symmetries might be vastly different, which may cause gradient conflict problems (Liu et al., 2021; Javaloy & Valera, 2021; Shi et al., 2021) as the control policy $\pi^C(a|s, D)$ is shared across all robots.

To mitigate this problem and smooth the gap between subgroups, we propose a novel search method by exploiting the structure of subgroups as in Figure 2(a). The core idea is to let subgroups G_i and G_{i+1} in consecutive iterations be similar by ensuring that they are adjacent in group structure: $G_{i+1} \in \textit{Neighbor}(G_i)$, which is defined in Appendix A.7. In Figure 2, $\textit{Neighbor}(K_0) = \{H_4, K_0, H_{2,0}\}$.

In practice, we maintain a value dict for all subgroups based on mean episode reward. At the beginning of training, we set initial subgroup $G_0 = \{e\}$ which represents the original design space to avoid introducing any prior knowledge, and then in each iteration i+1, we sample a subgroup from $\textit{Neighbor}(G_i)$ using ϵ-greedy based on their values. The sampled subgroup G_{i+1} is then used to generate robots in Section 4.1 for several episodes. At the end of each iteration, we update the value for the current symmetry.

However, it is possible that G_i, G_{i+1} are not similar enough. See Figure 2(a) for intuition. To further smooth the gap between subgroups, we consider the middle points between two adjacent subgroups G, G'. Assuming $G < G'$, there is no subgroup between them as we discussed above. However, we only need to ensure that these middle points can be used in Skeleton Design Stage (Equation (1)) and Attribution Design Stage (Equations (2) and (3)). We can prove that $\Pi_{G'}(c) = \beta_0 \Pi_G(c) + (1-\beta_0)\Pi_{G'-G}(c)$ where $\beta_0 = |G|/|G'|$ and $G'-G \triangleq \{g | g \in G', g \notin G\}$ (see Appendix A.8). We therefore define

$$\Pi_{G,G',\beta}(c) \triangleq \beta \Pi_G(c) + (1-\beta)\Pi_{G'-G}(c) \tag{4}$$

where for any β in interval $[\beta_0, 1]$, $\Pi_{G,G',\beta}$ is the symmetry map of a middle point between G, G'. Π_{G,G',β_0}=$\Pi_{G'}$ and $\Pi_{G,G',1}$=Π_G. In practice, we divide the interval equally into K parts and consider the middle points as neighbors. For example, in Figure 2(a), K=3, denoting M_2 by $\frac{1}{3}H_4 + \frac{2}{3}K_0$, we have $\textit{Neighbor}(K_0) = \{\frac{1}{3}H_4 + \frac{2}{3}K_0, \frac{1}{3}H_{2,0} + \frac{2}{3}K_0, K_0\}$. In this way, the designed robots in consecutive iterations are much more similar than before. And we can prove a similar theorem as Theorem 4.2 in Appendix A.9.

We outline our SARD algorithm in Algorithm 1 and we also provide a detailed version in Appendix A.1.

5. Experiments

In this section, we benchmark our method SARD on various MuJoCo (Todorov et al., 2012) tasks. We evaluate the effectiveness of SARD by asking the following questions: (1) Can robot design help improve performance compared with handcrafted design? (Section 5.2) (2) Can SARD outperform other robot design baselines in various tasks? (Section 5.2) (3) How does SARD search for the desired symmetry and how does symmetry facilitate control policy learning? (Section 5.3) (4) Can the searched symmetry by SARD generalize to all tasks? (Section 5.4). For qualitative results, please refer to the videos on our anonymous project website[1]. And our code is available at GitHub[2].

[1] https://sites.google.com/view/robot-design

[2] https://github.com/drdh/SARD

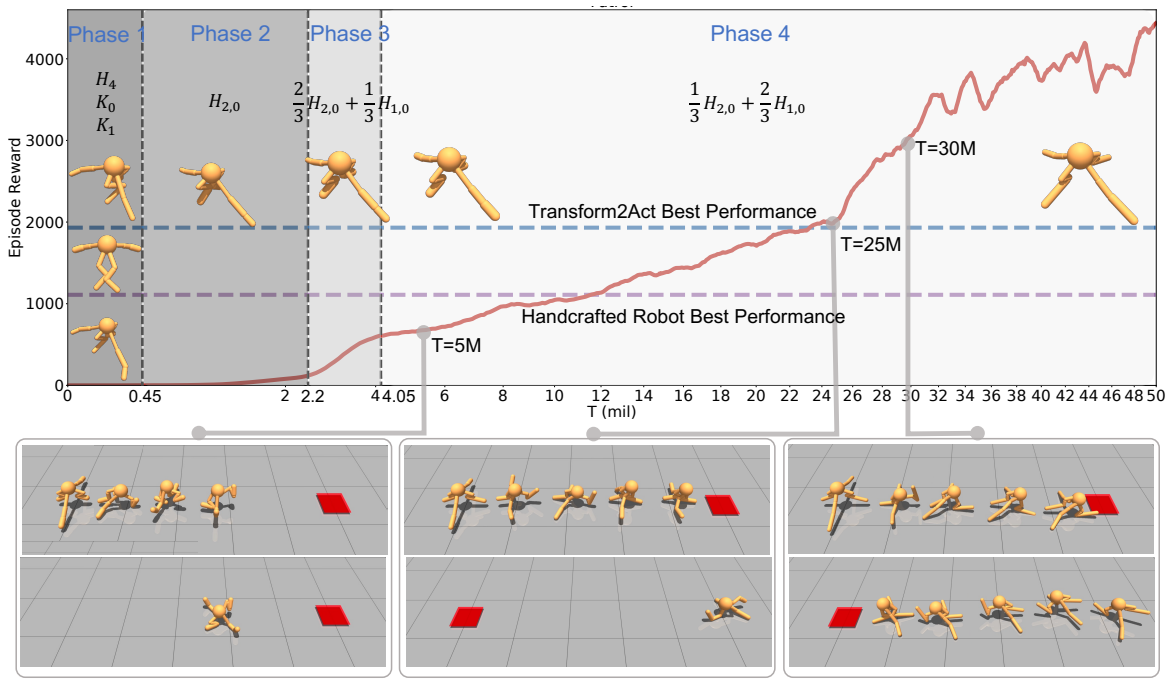

Figure 4. Visualization of the learning process of SARD. x-axis has been rescaled to better display early phases.

5.1. Experiment Setup

We run experiments on six tasks in this section. All runs are conducted with 4 random seeds and the mean performance as well as 95% confidence intervals are shown.

Environments. All six tasks are adapted from Gupta et al. (2021b), which are created based on MuJoCo (Todorov et al., 2012) physics simulator. The tasks vary in objectives (locomotion, approaching random goals, pushing an object,*etc*.), terrains (flat terrain, variable terrain, bowl-shaped terrain,*etc*.), and observation space. Here we briefly discuss the tasks: (1) Point Navigation. An agent is spawned at the center of a flat arena and has to reach a random goal in the arena; (2) Escape Bowl. An agent is spawned as the center of a bowl-shaped terrain and has to escape from the region; (3) Patrol. An agent needs to run forth and back between two goals; (4) Locomotion on Variable Terrain. An agent has to run forward as fast as possible in a variable terrain; (5) Locomotion on Flat Terrain. An agent has to run forward as fast as possible in flat terrain; (6) Manipulate Box. An agent is tasked with pushing a box into a randomly generated goal. For more details, please refer to Appendix B.1.

Baselines and Implementations. Our method SARD is implemented on the top of Transform2Act (Yuan et al., 2021), which is the previous state-of-the-art robot design method and is compared in this section. We use the same hyperparameters as Transform2Act for fair comparisons. To show the strength of robot design, we also compare SARD with Handcrafted Robot, which is the human-designed Ant using expert knowledge from OpenAI. We did not include any ES-based methods here because their sample inefficiency has been verified by Yuan et al. (2021).

Ablations. There are two contributions that characterize our method. (1) A plug-and-play transformation module that is used to generate robots under a given symmetry. (2) A search method that utilizes structured subgroups to find the optimal symmetry. Our novelties are mainly about the searching and utilization of symmetries, and these two contributions closely rely on each other. If we remove symmetry, we will get Transform2Act. Therefore, the effectiveness of symmetry-aware robot design can be demonstrated by comparing Transform2Act and SARD. To validate the effectiveness of the searching method, we further design the following ablation studies: (1) SARD (*Unstructured*). Do not consider the structure of subgroups discussed in Section 4.2 and sample subgroups directly from the whole set. (2) SARD (K=1) and SARD (K=5). Our method divides the interval between two adjacent subgroups into K(=3 in SARD) parts. We use these two ablations to show the effectiveness of smoothing the gap between subgroups.

5.2. Training Performance Comparison

We summarize the training performance in Figure 3 and we also show one representative robot designed by SARD at the end of training in the upper left corner of each subfigure to provide more intuition. SARD outperforms the previous state-of-the-art algorithm Transform2Act and handcrafted robot Ant in all tasks. This validates that (1) the designed robots can improve final performance compared with handcrafted robots and (2) symmetry-aware learning

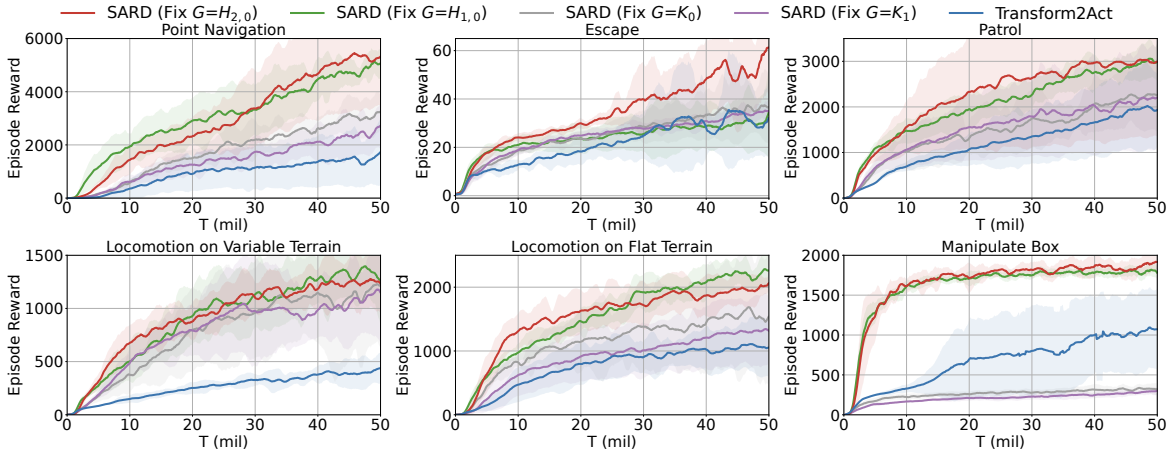

Figure 5. Generalization performance of the symmetry searched by SARD against baselines.

can effectively help solve the robot design problem.

As for ablations, SARD is better than SARD (*Unstructured*) in all tasks, which shows the usefulness of utilizing subgroup structures. Further, SARD outperforms SARD (K=1) in most tasks, which indicates the effectiveness of smoothing the gap between consecutive subgroups. Interestingly, SARD also outperforms SARD (K=5). This is because dividing the interval between two adjacent subgroups into too many parts will slower the learning such that it will take more iterations to move from one symmetry to another and the training could easily get stuck.

5.3. Robot Design Analysis

To investigate why SARD performs better than baselines and how can symmetry help improve performance, in this subsection, we visualize the learning process of SARD in Patrol task in Figure 4. According to the change of symmetry, the learning process can be divided into four phases. And in each phase, we also show the corresponding symmetry and a representative robot of this phase. In Phase 1, SARD explores some symmetries around the initial subgroup H_4 (see Figure 2(a)) and changed symmetries frequently. In Phase 2, SARD chooses a new subgroup $H_{2,0}$ and the episode reward is still low. In Phase 3, SARD moves to a middle point between two subgroups $H_{2,0}$ and $H_{1,0}$ that is closer to $H_{2,0}$ and the learning curve starts to rise, which indicates the usefulness of the smoothing trick of SARD. In Phase 4, SARD finally finds a suitable subgroup that still lies between two subgroups but closer to $H_{1,0}$. This subgroup is then left unchanged and SARD optimizes robot designs under this symmetry, in which case the search space is legitimately reduced. The left and right robots in Phase 4 are loaded from the beginning and end of this phase, respectively. Only some attributes of the robots are changed.

We also visualize the time-lapse images of control policies in Figure 4 to show how symmetry can foster learning to control. Patrol task requires robots to run forth and back between two goals as fast as possible. At t=$5M$, SARD still has trouble in running to the goal ahead. At t=$25M$, SARD learns to reach the goal ahead but fails to learn to run backward and Transform2Act also gets stuck at this point. However, SARD eventually exceeds the best performance of Transform2Act at t=$30M$. This is because, in Patrol task, Transform2Act usually needs to learn to turn around, which can easily lead to a tumble and is hard for robots. But for SARD, thanks to its symmetry, it does not have to turn around and can directly run in the opposite direction, which shows one of the benefits of symmetry.

The visualization results reflect two facts: (1) symmetry in robots is helpful in terms of reducing the control complexity, and (2) SARD can help learn symmetry by the means of searching in structured subgroups.

5.4. Generalization of the Learned Symmetry

To further validate the generalization of the learned symmetry by SARD. We counted all results of SARD in Figure 3 and if the symmetry is a middle point between two subgroups, we choose to count the one closest to it. We find that $3/4$ of the experiments ended up with $H_{2,0}$ and $H_{1,0}$ ($H_{2,0}$ accounted for 45.83% and $H_{1,0}$ accounted for 29.17%). Therefore, in this subsection, we run new experiments that fix symmetry to several potential ones to further test the effectiveness of SARD. The results are summarized in Figure 5. Here K_1 and K_0 is the bilateral symmetry used in previous works (Gupta et al., 2021b; Wang et al., 2019). SARD (Fix $G = H_{2,0}$) and SARD (Fix $G = H_{1,0}$) outperform other symmetries in most tasks, which validate the generalization of the learned symmetries.

6. Conclusions

In this paper, we exploit the structure of design space in the robot design problem with the symmetry characteristics of robot. Our method SARD can generate efficient symmetric robots while still covering the original design space. The theoretical analyses and empirical evaluations of SARD have shown its superior strength.

Acknowledgments

This work is supported in part by Science and Technology Innovation 2030 - "New Generation Artificial Intelligence" Major Project (No. 2018AAA0100904) and the National Natural Science Foundation of China (62176135).

References

Abrams, M. J., Basinger, T., Yuan, W., Guo, C.-L., and Goentoro, L. Self-repairing symmetry in jellyfish through mechanically driven reorganization. *Proceedings of the National Academy of Sciences*, 112(26):E3365–E3373, 2015.

Auerbach, J. E. and Bongard, J. C. Environmental influence on the evolution of morphological complexity in machines. *PLoS computational biology*, 10(1):e1003399, 2014.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Cheney, N., Bongard, J., SunSpiral, V., and Lipson, H. Scalable co-optimization of morphology and control in embodied machines. *Journal of The Royal Society Interface*, 15(143):20170937, 2018.

Colson, B., Marcotte, P., and Savard, G. An overview of bilevel optimization. *Annals of operations research*, 153 (1):235–256, 2007.

Dong, H., Wang, T., Liu, J., and Zhang, C. Low-rank modular reinforcement learning via muscle synergy. *arXiv preprint arXiv:2210.15479*, 2022.

Enquist, M. and Arak, A. Symmetry, beauty and evolution. *Nature*, 372(6502):169–172, 1994.

Evans, S. D., Hughes, I. V., Gehling, J. G., and Droser, M. L. Discovery of the oldest bilaterian from the ediacaran of south australia. *Proceedings of the National Academy of Sciences*, 117(14):7845–7850, 2020.

Evans, T. A. and Bashaw, G. J. Axon guidance at the midline: of mice and flies. *Current opinion in neurobiology*, 20(1):79–85, 2010.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

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

Gallian, J. A. *Contemporary abstract algebra*. Chapman and Hall/CRC, 2021.

Graham, J. H., Raz, S., Hel-Or, H., and Nevo, E. Fluctuating asymmetry: methods, theory, and applications. *Symmetry*, 2(2):466–540, 2010.

Gupta, A., Fan, L., Ganguli, S., and Fei-Fei, L. Metamorph: Learning universal controllers with transformers. In *International Conference on Learning Representations*, 2021a.

Gupta, A., Savarese, S., Ganguli, S., and Fei-Fei, L. Embodied intelligence via learning and evolution. *Nature communications*, 12(1):1–12, 2021b.

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

Ha, S., Coros, S., Alspach, A., Kim, J., and Yamane, K. Joint optimization of robot design and motion parameters using the implicit function theorem. In *Robotics: Science and systems*, volume 8, 2017.

Huang, W., Mordatch, I., and Pathak, D. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pp. 4455–4464. PMLR, 2020.

Javaloy, A. and Valera, I. Rotograd: Gradient homogenization in multitask learning. *arXiv preprint arXiv:2103.02631*, 2021.

Jelisavcic, M., Glette, K., Haasdijk, E., and Eiben, A. Lamarckian evolution of simulated modular robots. *Frontiers in Robotics and AI*, 6:9, 2019.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kurin, V., Igl, M., Rocktäschel, T., Boehmer, W., and Whiteson, S. My body is a cage: the role of morphology in graph-based incompatible control. In *International Conference on Learning Representations*, 2020.

Liu, B., Liu, X., Jin, X., Stone, P., and Liu, Q. Conflict-averse gradient descent for multi-task learning. *Advances in Neural Information Processing Systems*, 34:18878–18890, 2021.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.

Nolfi, S. and Floreano, D. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

Pappas, J. L., Tiffany, M. A., and Gordon, R. The uncanny symmetry of some diatoms and not of others: A multiscale morphological characteristic and a puzzle for morphogenesis. *Diatom Morphogenesis*, pp. 19–67, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Pathak, D., Lu, C., Darrell, T., Isola, P., and Efros, A. A. Learning to control self-assembling morphologies: a study of generalization via modularity. *Advances in Neural Information Processing Systems*, 32, 2019.

Polak, M. and Trivers, R. The science of symmetry in biology. *Trends in ecology & evolution*, 9(4):122–124, 1994.

Raibert, M. H. *Legged robots that balance*. MIT press, 1986a.

Raibert, M. H. Running with symmetry. In *Autonomous Robot Vehicles*, pp. 45–61. Springer, 1986b.

Raibert, M. H. Symmetry in running. *Science*, 231(4743): 1292–1294, 1986c.

Savriama, Y. and Klingenberg, C. P. Beyond bilateral symmetry: geometric morphometric methods for any type of symmetry. *BMC evolutionary biology*, 11(1):1–24, 2011.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

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

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

Shao, X., Mugler, A., Kim, J., Jeong, H. J., Levin, B. R., and Nemenman, I. Growth of bacteria in 3-d colonies. *PLoS computational biology*, 13(7):e1005679, 2017.

Shi, Y., Seely, J., Torr, P. H., Siddharth, N., Hannun, A., Usunier, N., and Synnaeve, G. Gradient matching for domain generalization. *arXiv preprint arXiv:2104.09937*, 2021.

Sims, K. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 15–22, 1994a.

Sims, K. Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372, 1994b.

Sinha, A., Malo, P., and Deb, K. A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2017.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

van der Pol, E., Worrall, D., van Hoof, H., Oliehoek, F., and Welling, M. Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:4199–4210, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vemula, A., Sun, W., and Bagnell, J. Contrasting exploration in parameter and action space: A zeroth-order optimization perspective. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2926–2935. PMLR, 2019.

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. In *International conference on learning representations*, 2018.

Wang, T., Zhou, Y., Fidler, S., and Ba, J. Neural graph evolution: Towards efficient automatic robot design. *arXiv preprint arXiv:1906.05370*, 2019.

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

Yu, W., Liu, C. K., and Turk, G. Policy transfer with strategy optimization. *arXiv preprint arXiv:1810.05751*, 2018.

Yuan, Y., Song, Y., Luo, Z., Sun, W., and Kitani, K. Transform2act: Learning a transform-and-control policy for efficient agent design. *arXiv preprint arXiv:2110.03659*, 2021.

Zhao, A., Xu, J., Konaković-Luković, M., Hughes, J., Spielberg, A., Rus, D., and Matusik, W. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

A. Method Details

A.1. Algorithm

We implement SARD based on Transform2Act in Section 5 and outline the algorithm in Algorithm 2. The highlighted lines are the changed procedures Transform2Act. The main components of this algorithm are:

Line 2, 29-30: Initialize and update current symmetry type;

Line 7-12: Skeleton Design Stage, in which SARD applies skeleton symmetry map Δ_G^{skel} (defined in Equation (1)) to transform skeleton design actions.

Line 13-19: Attribute Design Stage, in which SARD applies attribute symmetry map Δ_G^{attr} (defined in Equation (2)) and Π_G (defined in Equation (4)) to transform attribute design actions.

Line 20-26: Control Stage.

Algorithm 2 SARD: Symmetry-Aware Robot Design implemented based on Transform2Act

input group Dih_n; number of intervals between symmetries K; symmetry sampling exploration rate ϵ; skeleton design steps N^{skel}; attribute design steps N^{attr}; max steps of each episode H;
output symmetry G; design policy $\pi^D = (\pi^{skel}, \pi^{attr})$, control policy π^C;

1: Initialize π^D and π^C;
2: Initialize symmetry $G \leftarrow \{e\}$; initialize value dict for symmetries $V \leftarrow \mathbf{0}$;
3: **while** not reaching max iterations **do**
4: Memory $\mathcal{M} \leftarrow \emptyset$;
5: **while** $\mathcal{M}$ not reaching batch size **do**
6: $D_0 \leftarrow$ initial robot design;
7: **for** $t = 1, 2, \ldots, N^{skel}$ **do**
8: Sample skeleton design action $a_{v,t}^{skel} \sim \pi^{skel}$;
9: Apply skeleton symmetry map Δ_G^{skel} defined in Equation (1) to transform $a_{v,t}^{skel}$;
10: $D_{t+1} \leftarrow$ apply $\{a_{v,t}^{skel}\}_{v \in V_t}$ to modify skeleton (V_t, E_t) in D_t;
11: $r_0 \leftarrow 0$; store $(r_t, \{a_{v,t}^{skel}\}_{v \in V_t}, D_t)$ into $\mathcal{M}$;
12: **end for**
13: **for** $t = N^{skel}{+}1, \ldots, N^{skel}{+}N^{attr}$ **do**
14: Sample attribute design action $(a_{v,t}^{sca}, a_{v,t}^{vec}) = a_{v,t}^{attr} \sim \pi^{attr}$;
15: Apply attribute symmetry map for scalar values Δ_G^{atrr} defined in Equation (2) to transform $a_{v,t}^{sca}$;
16: Apply attribute symmetry map for vector values Π_G defined in Equation (4) to transform $a_{v,t}^{vec}$;
17: $D_{t+1} \leftarrow$ apply $\{a_{v,t}^{attr}\}_{v \in V_t}$ to modify attributes Z_t in D_t;
18: $r_t \leftarrow 0$; store $(r_t, \{a_{v,t}^{attr}\}_{v \in V_t}, D_t)$ into $\mathcal{M}$;
19: **end for**
20: $s_{t+1} \leftarrow$ initial environment state;
21: **for** $t = N^{skel}{+}N^{attr}{+}1, \ldots, H$ **do**
22: Sample control actions $a_t^C \sim \pi^C$;
23: $s_{t+1} \leftarrow$ environment dynamics $\mathcal{T}(s_{t+1}|s_t, a_t^C, D_t)$;
24: $r_t \leftarrow$ environment reward $R(s_t, a_t^C, D_t)$; $D_{t+1} \leftarrow D_t$;
25: Store (r_t, a_t^C, s_t, D_t) into $\mathcal{M}$;
26: **end for**
27: **end while**
28: Update π^C, π^D with PPO using samples in $\mathcal{M}$;
29: Update symmetry values $V(G) \leftarrow$ mean episode rewards in $\mathcal{M}$;
30: Sample a new symmetry from neighbors $G \leftarrow \textit{Neighbor}(G)$ with ϵ-greedy using symmetry values V;
31: **end while**

A.2. Group Theory

A group G is a set with a binary operation $(\cdot)$ such that it has four basic mathematical properties (Gallian, 2021):

- associativity: $g_1 \cdot (g_2 \cdot g_3) = (g_1 \cdot g_2) \cdot g_3, \forall g_1, g_2, g_3 \in G$.
- closure: $g_1 \cdot g_2 \in G, \forall g_1, g_2 \in G$.
- the existence of an identity: $e \in G$
- the existence of an inverse of each element: $g^{-1} \in G, \forall g \in G$.

A.3. Dihedral Groups

Definition. The dihedral group is a finite discrete group containing rotation and reflection transformation. A dihedral group $\text{Dih}_n (n \geq 3)$ can be generated by rotation transformation ρ (counterclockwise rotation by $360°/n$) and reflection transformation π (reflection along x-axis), *i.e.*, $\text{Dih}_n = \langle \rho, \pi | \rho^n = \pi^2 = 1, \pi\rho\pi^{-1} = \rho^{-1} \rangle$. Concretely, $\text{Dih}_n = \{\rho_k, \pi_{k-1} | k = 1, 2, \cdots, n\}$, where $\rho_k = \rho^k, \rho_0 = \rho_n = e$ and $\pi_k = \rho^k \pi$.

Group Element Representation. Each group element can have multiple representations. We consider permutation representation and matrix representation in this paper. The permutation and matrix representations of $g (\in \text{Dih}_4)$ are list below:

$$P_{\rho_0} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_{\rho_0} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad P_{\rho_1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, M_{\rho_1} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix},$$

$$P_{\rho_2} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M_{\rho_2} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \quad P_{\rho_3} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, M_{\rho_3} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},$$

$$P_{\pi_0} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, M_{\pi_0} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad P_{\pi_1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, M_{\pi_1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

$$P_{\pi_2} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_{\pi_2} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \quad P_{\pi_3} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, M_{\pi_3} = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix},$$

Here we take $\pi_0 \in \text{Dih}_4$ as an example. P_{π_0} exchanges $2, 4$-columns and keeps the other unchanged when right multiplied with coordinate pairs ($\mathbb{R}^{2\times 4}$), and M_{π_0} reflects the coordinates along x-axis when left multiplied with the original coordinates. Taking the designed robot in Figure 2(c) as an example, P_{π_0} exchanges joint v_2, v_4 and M_{π_0} reflects their coordinates along x-axis.

Subgroups. The subgroups of dihedral groups can be classed into three categories:

1. $H_d = \langle \rho_d \rangle$, where $1 \leq d \leq n$ and $d|n$ (n is divisible by d).
2. $K_i = \langle \pi_i \rangle$, where $0 \leq i \leq n{-}1$.
3. $H_{k,l} = \langle \rho_k, \pi_l \rangle$, where $0 \leq l < k \leq n{-}1$ and $k|n$.

The group structure of Dih_4 is shown in Figure 2(a). An interesting result is that the number of the subgroups of Dih_n is $\sum_{w \in W}(1 + w)$, where W is the set of all positive divisors of n. For example, $W = \{1, 2, 4\}$ for Dih_4 and thus it has 10 subgroups.

We list all subgroups of Dih_4 below:

$$
\begin{aligned}
&H_1 = \{\rho_0, \rho_1, \rho_2, \rho_3\},\\
&H_2 = \{\rho_0, \rho_2\},\\
&H_4 = \{\rho_0\},\\
&K_0 = \{\rho_0, \pi_0\},\\
&K_1 = \{\rho_0, \pi_1\},\\
&K_2 = \{\rho_0, \pi_2\},\\
&K_3 = \{\rho_0, \pi_3\},\\
&H_{1,0} = \{\rho_0, \rho_1, \rho_2, \rho_3, \pi_0, \pi_1, \pi_2, \pi_3\},\\
&H_{2,0} = \{\rho_0, \rho_2, \pi_0, \pi_2\},\\
&H_{2,1} = \{\rho_0, \rho_2, \pi_1, \pi_3\}.
\end{aligned}
$$

Extend the Dimension of P_g. The original permutation representation $P_g \in \{0,1\}^{n\times n}$, we extend it to $\{0,1\}^{|V|\times|V|}$. In the design $D = \{V, E, Z\}$, the number of joints near the torso is exactly n. And for other joints in layer k, we only need to use their parent joints' permutation representation. Specifically, we select the corresponding rows and columns of permutation representation belonging to their parent joints and denote it by P_k, then the new permutation is:

$$
P_g = \begin{pmatrix} P_g & & & & \\ & P_2 & & & \\ & & P_3 & & \\ & & & \ddots & \\ & & & & P_J \end{pmatrix}
$$

where J is the total number of layers (see Appendix A.4 for details).

A.4. Formal Definitions of the Transformation Functions

We now define the transformation functions used in G-symmetric in Definition 4.1: $\alpha_g^V(v), \alpha_g^E(e), \alpha_g^Z(z)$.

Based on the distance to the root node (torso in robots), nodes V can be divided into several disjoint subsets: $V = \bigcup_{j=1}^{J} V_j$, where the distance of $v \in V_j$ to root node is j. And thus the attribute set Z can also be divided: $Z = \bigcup_{j=1}^{J} Z_j$. Note that each attribute corresponds to a joint, thus we always have $|V_j| = |Z_j|, \forall j$. Assuming the current dihedral group is Dih_n, to ensure the designed robot is G-symmetric (G<Dih_n), we let $|V_1| = n$ by setting the initial design $D_0 = (V = \{v_1, v_2, \cdots, v_n\}, E = \emptyset, Z = \mathbf{0})$ and forbidding growing any new joints from the root node (torso). We further set the embedding of v_j to $\delta_j \in \{0,1\}^{1\times n}$, where only the j-th row of δ_j is 1. And define $Enc(v_j) = \delta_j, Dec(\delta_j) = v_j$, we have

$$
\alpha_g^V(v) = Dec(Enc(v)^\top P_g), \forall v \in V_1, \tag{5}
$$

where P_g is the permutation representation of g. For the nodes in other layers, the embeddings are the of their parent nodes and α_g^V the same. Using α_g^V, we have:

$$
\alpha_g^E(e) = \alpha_g^E((v_i, v_j)) = (\alpha_g^V(v_i), \alpha_g^V(v_j)), \forall e \in E. \tag{6}
$$

As for $\alpha_g^Z(z)$, where $z = (z^{sca}, z^{vec})$, the scalar value z^{sca} is unchanged and the vector value z^{vec} is transformed using M_g, *i.e.*, the matrix representation of g:

$$
\alpha_g^Z(z) = \alpha_g^Z((z^{sca}, z^{vec})) = (z^{sca}, M_g z^{vec}). \tag{7}
$$

A.5. Proof of Theorem 4.2 of Π_G

To prove Theorem 4.2, we first discuss the feasible symmetric space of a_v^{vec}, then prove the symmetry map can project any design into this symmetric space and leave any design in the symmetric space unchanged. For simplicity, here we assume

$a_v^{vec} = (x, y)^\top$, where $x, y \in \mathbb{R}$, only includes one coordinate. The vector value actions of all joints form a matrix:

$$c = (a_1^{vec}, a_2^{vec}, \cdots, a_{|V|}^{vec}) = \begin{pmatrix} x_1 & x_2 & \cdots, x_{|V|} \\ y_1 & y_2 & \cdots, y_{|V|} \end{pmatrix} \tag{8}$$

The coordinate c should be invariant under transformation $g \in G$. That is, the transformed coordinate c'=$M_g c$ should be the same as c, where M_g is the matrix representation of g. In Figure 2(c), we only consider the first four joints here and have

$$c' = M_{\pi_0} c = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ -y_1 & -y_2 & -y_3 & -y_4 \end{pmatrix}. \tag{9}$$

However, the order of the items in c and c' might not be aligned, *e.g.*, v_2 in c' should be aligned with v_4 in c. To solve this issue, we inversely permutate the order of the items in c' by left multiplying it with the permutation representation of g^{-1}: $P_{g^{-1}}$. Since $\pi_0^{-1} = \pi_0$, we have

$$c' P_{\pi_0} = \begin{pmatrix} x_1 & x_4 & x_3 & x_2 \\ -y_1 & -y_4 & -y_3 & -y_2 \end{pmatrix}. \tag{10}$$

Note that the original permutation representation $P_{g^{-1}} \in \{0, 1\}^{n \times n}$, we extend it to $\{0, 1\}^{|V| \times |V|}$ in Appendix A.3. Finally we need to ensure that $M_g c P_{g^{-1}} = c, \forall g \in G$. That is,

$$c \in \mathcal{C}_G \triangleq \{c \in \mathbb{R}^{2 \times |V|} | M_g c P_{g^{-1}} = c, \forall g \in G\}. \tag{11}$$

Now we prove that The symmetry map Π_G defined in Equation (3) projects any coordinate $c \in \mathbb{R}^{2 \times |V|}$ into the set $\mathcal{C}_G$ defined in Equation (11). And if $c \in \mathcal{C}_G$, the symmetry map will leave c unchanged. Similar proofs are provided in van der Pol et al. (2020) for a different purpose.

For $\forall h \in G, c \in \mathcal{C}_G$,

$$\begin{aligned}
M_h \Pi_G(c) P_{h^{-1}} &= M_h \left(\frac{1}{|G|} \sum_{g \in G} M_g c P_{g^{-1}} \right) P_{h^{-1}} && (12) \\
&= \frac{1}{|G|} \sum_{g \in G} M_h M_g c P_{g^{-1}} P_{h^{-1}} && (13) \\
&= \frac{1}{|G|} \sum_{g \in G} M_{hg} c P_{g^{-1} h^{-1}} && (14) \\
&= \frac{1}{|G|} \sum_{g \in G} M_{hg} c P_{(hg)^{-1}} && (15) \\
&= \frac{1}{|G|} \sum_{h^{-1} r \in G} M_r c P_{(r)^{-1}} && (16) \\
&= \frac{1}{|G|} \sum_{r \in hG} M_r c P_{(r)^{-1}} && (17) \\
&= \frac{1}{|G|} \sum_{r \in G} M_r c P_{(r)^{-1}} && (18) \\
&= \Pi_G(c). && (19)
\end{aligned}$$

Thus we have $\Pi_G(c) \in \mathcal{C}_G$.

As for the other property, $\forall g \in G, c \in \mathcal{C}_G$, we have $M_g c P_{g^{-1}} = c$, that is $M_g c = c P_{g^{-1}}^{-1} = c P_g$. This follows:

$$\Pi_G(c) = \frac{1}{|G|} \sum_{g \in G} M_g c P_{g^{-1}} \tag{20}$$

$$= \frac{1}{|G|} \sum_{g \in G} c P_g P_{g^{-1}} \tag{21}$$

$$= \frac{1}{|G|} \sum_{g \in G} c \tag{22}$$

$$= c. \tag{23}$$

thus $\forall c \in \mathcal{C}_G, \Pi_G(c) = c$.

Combining these two properties, Theorem 4.2 is proved.

An Example for Symmetry Map Π_G. We provide an example of K_0 of symmetry map Π_G here. In Figure 2(c), note that $K_0 = \{\pi_0, e\}$ and we thus have

$$\Pi_{K_0}(c) = \frac{1}{2}\left(M_{\pi_0} c P_{\pi_0^{-1}} + M_e c P_{e^{-1}}\right) \tag{24}$$

$$= \frac{1}{2}\left(\begin{pmatrix} x_1 & x_4 & x_3 & x_2 \\ -y_1 & -y_4 & -y_3 & -y_2 \end{pmatrix} + \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{pmatrix}\right) \tag{25}$$

$$= \begin{pmatrix} x_1 & (x_2 + x_4)/2 & x_3 & (x_2 + x_4)/2 \\ 0 & (y_2 - y_4)/2 & 0 & (y_4 - y_2)/2 \end{pmatrix}. \tag{26}$$

A.6. Proof of Theorem 4.3

Define $\Xi \triangleq (\Delta_G^{skel}, \Delta_G^{attr}, \Pi_G)$, where Δ_G^{skel} is defined in Equation (1), Δ_G^{attr} is defined in Equation (2), Π_G is defined in Equation (3). Here we prove the equivalence between transformed design space by transformations Ξ and G-symmetric space.

In Section 4.1, we have shown that any transformed robots by Ξ are G-symmetric. Thus we only need to show that any G-symmetric robots can be generated by Ξ. To prove this, a sufficient condition is the fixing property of Ξ, *i.e.*, Ξ will leave any G-symmetric robots unchanged. Since we have proved the fixing property of Π_G, here we only need to show that Δ_G^{skel} and Δ_G^{attr} also have this property.

In a G-symmetric robot design $D = \{V, E, Z\}$, for any joint $v \in V$ and any joint $u \in \mathcal{O}_G(v)$, v and u must have the same number of child joints and same attributes (according to the definition of G-symmetric). Assuming D is generated layer by layer and the current design is $D' = (V', E', Z')$, in layer j, $\forall v \in V_j$, we have that all joints in $\mathcal{O}_G(v)$ can choose the same skeleton action and scalar attribute action as D. That is $a_v^{skel} =$ AddJoint if $\sum_{(v,u) \in E, (v,u) \notin E'} 1 > 0$ else NoChange, and $a_v^{atrr} = z_v \in Z$. Thus the design is left unchanged.

A.7. Neighbors of a Subgroup

$\forall G' \in$ *Neighbor*(G), we have

- $G' < G$ or $G < G'$;
- if $G < G'$ and $\exists H, G < H < G'$, we have $G = H$ or $G' = H$;
- if $G' < G$ and $\exists H, G' < H < G$, we have $G = H$ or $G' = H$;

A.8. Derivation of $\Pi_{G,G',\beta}$

Here we prove that

$$\Pi_{G'}(c) = \beta_0 \Pi_G(c) + (1 - \beta_0) \Pi_{G' - G}(c) \tag{27}$$

where $\beta_0 = |G|/|G'|$ and $G'\text{–}G \triangleq \{g|g \in G', g \notin G\}$.

$$\Pi_{G'}(c) = \frac{1}{|G|}\sum_{g\in G'} M_g c P_{g^{-1}} \tag{28}$$

$$= \frac{|G|}{|G'|}\cdot\frac{1}{|G'|}\sum_{g\in G} M_g c P_{g^{-1}} + \frac{|G'|-|G|}{|G'|}\cdot\frac{1}{|G'|-|G|}\sum_{g\in G'-G} M_g c P_{g^{-1}} \tag{29}$$

$$= \frac{|G|}{|G'|}\Pi_G(c) + \frac{|G'|-|G|}{|G'|}\Pi_{G'\text{-}G}(c) \tag{30}$$

$$= \beta_0\Pi_G(c) + (1-\beta_0)\Pi_{G'\text{-}G}(c). \tag{31}$$

A.9. A Theorem for $\Pi_{G,G',\beta}$

For symmetry map $\Pi_{G,G',\beta}$ defined in Equation (4), we also have a similar theorem as Theorem 4.2 in the following:

Theorem A.1. *The projected vector values $\Pi_{G,G',\beta}$ defined in Equation* (4) *are G-symmetric. And if c is already G'-symmetric, $\Pi_{G,G',\beta}(c) = c$.*

For $\forall h \in G, c \in \mathcal{C}_G$,

$$M_h\Pi_{G,G',\beta}P_{h^{-1}} = M_h\left(\beta\Pi_G(c) + (1-\beta)\Pi_{G'\text{-}G}(c)\right)P_{h^{-1}} \tag{32}$$

$$= M_h\left(\frac{\beta}{|G'|}\sum_{g\in G} M_g c P_{g^{-1}} + \frac{(1-\beta)}{|G'|-|G|}\sum_{g\in G'-G} M_g c P_{g^{-1}}\right)P_{h^{-1}} \tag{33}$$

$$= \frac{\beta}{|G'|}\sum_{g\in G} M_h M_g c P_{g^{-1}} P_{h^{-1}} + \frac{(1-\beta)}{|G'|-|G|}\sum_{g\in G'-G} M_h M_g c P_{g^{-1}} P_{h^{-1}} \tag{34}$$

$$= \frac{\beta}{|G'|}\sum_{g\in G} M_{hg} c P_{(hg)^{-1}} + \frac{(1-\beta)}{|G'|-|G|}\sum_{g\in G'-G} M_{hg} c P_{(hg)^{-1}} \tag{35}$$

$$= \frac{\beta}{|G'|}\sum_{h^{-1}r\in G} M_r c P_{r^{-1}} + \frac{(1-\beta)}{|G'|-|G|}\sum_{h^{-1}r\in G'-G} M_r c P_{r^{-1}} \tag{36}$$

$$= \frac{\beta}{|G'|}\sum_{r\in hG} M_r c P_{r^{-1}} + \frac{(1-\beta)}{|G'|-|G|}\sum_{r\in h(G'-G)} M_r c P_{r^{-1}} \tag{37}$$

$$= \frac{\beta}{|G'|}\sum_{r\in G} M_r c P_{r^{-1}} + \frac{(1-\beta)}{|G'|-|G|}\sum_{r\in G'-G} M_r c P_{r^{-1}} \tag{38}$$

$$= \beta\Pi_G(c) + (1-\beta)\Pi_{G'\text{-}G}(c) \tag{39}$$

$$= \Pi_{G,G',\beta}(c). \tag{40}$$

Here in Equation (38), $hG = G$ is a basic property of group theory, and we prove $h(G'\text{–}G) = G'\text{–}G$. Any element in $h(G'\text{–}G)$ can be represented by hg where $g \in G'\text{–}G$. If $hg \notin G'\text{–}G$, we have $hg \in G$. Assuming $g_1 \in G$ such that $hg = g_1$, we have $g = h^{-1}g_1 \in G$, which leads to a contradiction and thus $hg \in G'\text{–}G$. On the other hand, $\forall g_2 \in G'\text{–}G$, we have $g_2 = h(h^{-1}g_2)$ and only need to show that $h^{-1}g_2 \in G'\text{–}G$. Otherwise, assuming $h^{-1}g_2 \notin G'\text{–}G$, we have $h^{-1}g_2 \in G$ and thus $g_2 \in G$, which also leads to a contradiction and thus $g_2 \in h(G'\text{–}G)$. In conclusion, $h(G'\text{–}G) = G'\text{–}G$

Therefore, we have $\Pi_{G,G',\beta}(c) \in \mathcal{C}_G$.

As for the fixing property, $\forall g \in G, c \in \mathcal{C}_{G'}$, we have $M_g c P_{g^{-1}} = c$, that is $M_g c = c P_{g^{-1}}^{-1} = c P_g$. This follows:

$$\Pi_{G,G',\beta}(c) = \frac{\beta}{|G'|} \sum_{g \in G} M_g c P_{g^{-1}} + \frac{(1-\beta)}{|G'| - |G|} \sum_{g \in G' - G} M_g c P_{g^{-1}} \tag{41}$$

$$= \frac{\beta}{|G'|} \sum_{g \in G} c P_g P_{g^{-1}} + \frac{(1-\beta)}{|G'| - |G|} \sum_{g \in G' - G} c P_g P_{g^{-1}} \tag{42}$$

$$= \frac{\beta}{|G'|} \sum_{g \in G} c + \frac{(1-\beta)}{|G'| - |G|} \sum_{g \in G' - G} c \tag{43}$$

$$= c. \tag{44}$$

thus $\forall c \in \mathcal{C}_{G'}, \Pi_{G,G',\beta}(c) = c$.

B. Experiment Details

B.1. Details of the Tasks

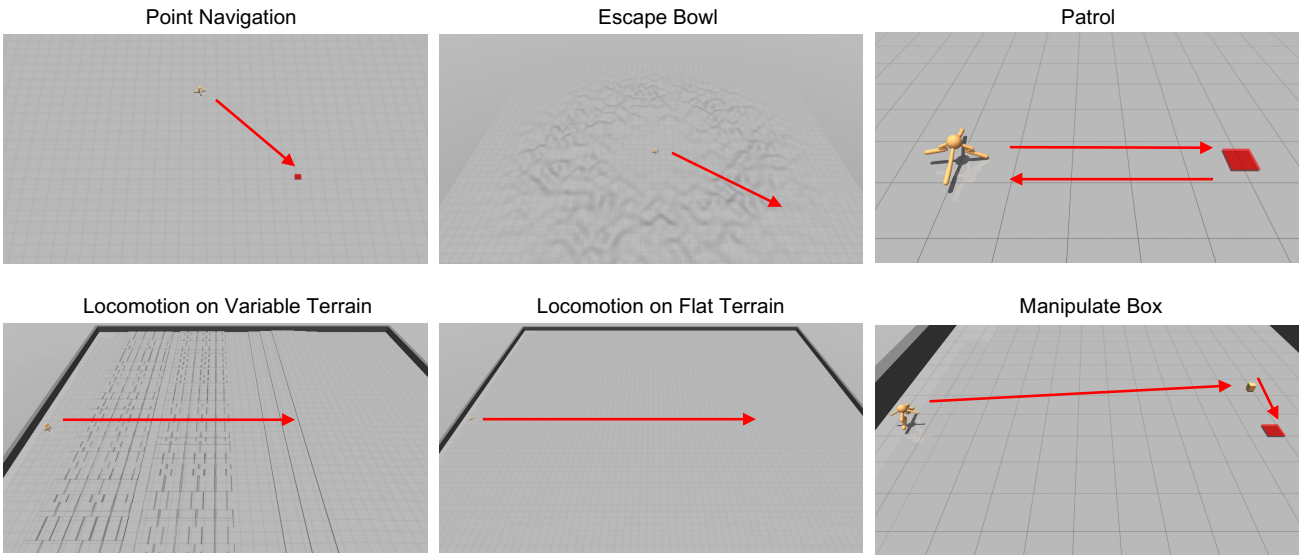

Figure 6. Visualization of the training tasks adapted from Gupta et al. (2021b)

We run experiments on six MuJoCo (Todorov et al., 2012) tasks adapted from Gupta et al. (2021b). These tasks can be categorized into 3 domains to test agility (Patrol, Point Navigation), stability (Escape Bowl,Locomotion on Variable Terrain,Locomotion on Flat Terrain), and manipulation (Manipulate Box) abilities of the designed robots. The detailed descriptions of these tasks are listed below.

Point Navigation. The agent is generated at the center of a $100 \times 100\ m^2$ flat arena and needs to reach a random goal (red square in Figure 6) in this arena. The ability to move in any specified direction quickly leads to success in this task. At each time step, the agent receives a reward r_t shown below,

$$r_t = w_{ag} d_{ag} - w_c \|a\|^2$$

where d_{ag} is the geodesic distance difference in the current time step and previous time step between the agent and the goal, $w_{ag} = 100$, $w_c = 0.001$, w_c is a penalty term for action a.

Escape Bowl. Generated at the center of a bowl-shaped terrain surrounded by small hills, the agent has to escape from the hilly region. This task requires the agent to maximize the geodesic distance from the initial location while going through a

random hilly terrain. At each time step, the agent receives a reward r_t shown below,

$$r_t = w_d d_{as} - w_c \|a\|^2$$

where d_{as} is the geodesic distance difference in the current time step and previous time step between the agent and the initial location, $w_d = 1$, $w_c = 0.001$.

Patrol. In the task, the agent is required to run back and forth between two target locations at a distance of 10 meters along the x-axis. Quick change of direction when the goal (red square in Figure 6) alters and rapid movement leads to the success of this task. The reward function is similar to the point navigation task. Additionally, we flip the goal location and provide the agent a sparse reward of 10 as it is within $0.5m$ from the goal location.

Locomotion on Variable Terrain. At the beginning of an episode, an agent is generated on the one end of a $100 \times 100\ m^2$ square arena. By randomly sampling a sequence of obstacles from a uniform distribution over a predefined range of parameter values, we can build a brand new terrain in each episode. While the length of flat segments in variable terrain $l \in [1, 3]m$ along the desired direction of motion, the length of obstacle segments $l \in [4, 8]m$. We primarily utilize 3 types of obstacles. 1) Hills, which is parameterized by the amplitude a of sin wave in which $a \in [0.6, 1.2]m$. 2) Steps, a sequence of 8 steps of height 0.2m. 3) Rubble, a sequence of random bumps (small hills) created by clipping a repeating triangular sawtooth wave at the top and height h of each bump clip samples from $[0.2, 0.3]m$ stochastically. The goal of the agent is to maximize forward displacement over an episode and this environment is quite challenging for the agent to perform well.

Locomotion on Flat Terrain. Similar to locomotion on variable terrain task, an agent is initialized on the one end of a $150 \times 150\ m^2$ square arena and aims at maximizing forward displacement over an episode.

Manipulate Box. In a $60 \times 40\ m^2$ arena similar to variable terrain, the agent is required to move a box (small cube with $0.2m$ shown in Figure 6) from the initial position to the target place (red square). Both the initial box location and final target location are randomly chosen with constraints that lead to a further path to the destination in each episode.

B.2. Implementation of SARD

We implement SARD based on Transform2Act (Yuan et al., 2021), which uses GNN-based (Scarselli et al., 2008; Bruna et al., 2013; Kipf & Welling, 2016) control policies. GNN-based policies can deal with variable input sizes across different robot designs by sharing parameters between joints. This property allows us to share policies across all designed robots. Note that our method is general and can be combined with any other network structures used in modular RL, *e.g.*, message passing networks (Huang et al., 2020) and Transformers (Dong et al., 2022; Kurin et al., 2020).

However, this sharing also brings negative impacts, *e.g.*, joints in similar states will choose similar actions, which may severely hinder performance. To solve this problem, Transform2Act proposed to add a joint-specialized MLP (JSMLP) after the GNNs. We follow this setting for fair comparisons.

For design policy and control policy learning, we use Proximal Policy Optimization (PPO) (Schulman et al., 2017), a standard policy gradient method (Williams, 1992) for optimizing these two policies.

Here we provide the hyperparameters needed to replicate our experiments in Table 1, and we also include our codes in the supplementary.

Experiments are carried out on NVIDIA GTX 2080 Ti GPUs. Taking Point Navigation as an example, SARD requires approximately 10G of RAM and 4G of video memory and takes about 36 hours to finish 50M timesteps of training.

B.3. Details of Baselines

Transform2Act. We use the official implementation of Transform2Act, where all networks and optimizations are implemented with PyTorch (Paszke et al., 2019). The GNN layers are GraphConv (Morris et al., 2019) implemented in PyTorch Geometric package (Fey & Lenssen, 2019). All policies are optimized with PPO (Schulman et al., 2017) with generalized advantage estimation (GAE) (Schulman et al., 2015). The authors searched the hyperparameters and we also list the selected values in Table 1. We also removed the initial design of Transform2Act to avoid any prior knowledge.

Handcrafted Robot. To show the strength of robot design, we also compare SARD with Handcrafted Robot, which is the human-designed robot Ant[2] using expert knowledge from OpenAI. We directly load the XML file and skip the Design Stage.

[2] https://github.com/openai/gym/blob/master/gym/envs/mujoco/assets/ant.xml

Table 1. Hyperparameters of SARD and Transform2Act.

Hyperparameters	Value
Skeleton Design Stage Time Steps N^{skel}	5
Attribute Design Stage Time Steps N^{attr}	1
GNN Layer Type	GraphConv
JSMLP Activation Function	Tanh
GNN Size	(64,64,64)
JSMLP Size	(128,128)
Policy Learning Rate	5e-5
Value Learning Rate	3e-4
PPO Clip	0.2
PPO Batch Size	50000
PPO MiniBach Size	2048
PPO Iterations Per Batch	10
Training Epochs	1000
Discount Factor γ	0.995
GAE λ	0.95
Subgroup Exploration Rate ϵ	0.01

Table 2. Training performance of SARD based on different base algorithms.

	NGE	Transform2Act	SARD+NGE	SARD+Transform2Act
Point Navigation	1131.50 ±458.45	1618.10±1022.01	**4729.00±835.62**	4262.78±738.17
Escape Bowl	8.65±1.88	32.37±13.62	15.55±3.69	**88.61±13.23**
Patrol	1120.30±425.89	1995.95±709.07	3104.67±1082.03	**3116.47±801.43**
Locomotion on Variable Terrain	170.85±48.05	443.22±74.72	408.65±74.95	**1204.01±96.16**
Locomotion on Flat Terrain	238.65±86.75	1067.16±463.55	835.75±366.25	**2438.26±297.09**
Manipulate Box	1061.90±541.47	1073.11±467.38	**1793.00±27.80**	1604.27±137.72

The Control Stage and optimization are the same as ours. We run 50M time steps for it and show the best performance.

B.4. Details of Ablations

SARD (*Unstructured*). The $Neighbor(G_i)$ function at iteration i is set to contain all subgroups and other components are the same as SARD.

SARD (*K*=5). Divide the interval between two adjacent subgroups into $K = 5$ parts and keep others the same as SARD.

SARD (*K*=1). Do not divide the interval between two adjacent subgroups and only use original group structures to define $Neighbor(G_i)$ function at iteration i.

C. Extra Results

C.1. Combine SARD with Other Robot Design Method

Our method is a plug-and-play module that can be utilized in other robot design methods. Here we provide experimental results of combining our method (SARD) with NGE (Wang et al., 2019), which is an ES-based robot design method. We report the results in Table 2. Here SARD+NGE is the implementation of SARD based on NGE, and SARD+Transform2Act is our original implementation of SARD based on Transform2Act, denoted by 'SARD (*K*=3)' in our paper. All runs are conducted with 3 random seeds and each data item in the table is formatted as 'mean±std'. As shown in Table 2, SARD+NGE outperforms vanilla NGE in all tasks and is even better than our original implementation SARD+Transform2Act in two tasks. The performance improvement of SARD+BaseAlgo over BaseAlgo (BaseAlgo $\in$ {NGE, Transform2Act}) showcases the generality of SARD.

Table 3. Training performance of SARD compared with Transform2Act on its original tasks.

	Swimmer	2D Locomotion	Gap Crosser
Transform2Act	607.50±89.02	**3329.00±2094.56**	1352.20±558.07
SARD	**975.50±9.10**	3194.00±1695.06	**1824.43±1322.30**

C.2. Results of SARD on the Tasks Used in Transform2Act

For a complete comparison, we also provide extra results on the tasks used in Transform2Act (Yuan et al., 2021). We show the final performance comparison between our method SARD and Transform2Act in Table 3. Here SARD is our original implementation of SARD based on Transform2Act, denoted by 'SARD (K=3)' in our paper. All runs are conducted with 3 random seeds and each data item in the table is formatted as 'mean±std'. The results that SARD outperforms Transform2Act in most tasks further validate the strength of our method. Also, please note that the '3D Locomotion' task in their paper is similar to 'Locomotion on Flat Terrain' in Table 2, thus we omit this task here. The reported results of Transform2Act are based on their released code.

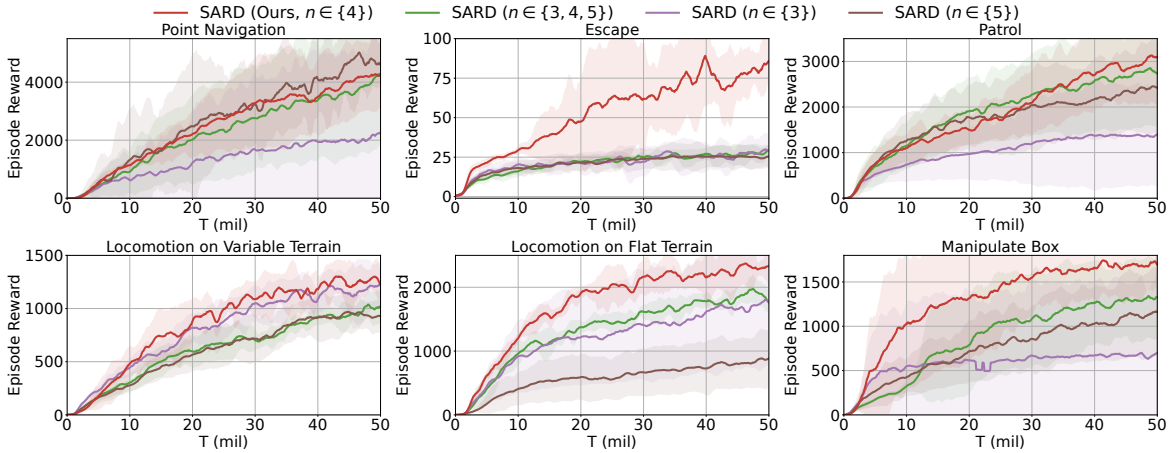

Figure 7. Hyperparameter search of Dih_n.

C.3. Different Dihedral Groups

In this paper, we use the subgroups of the dihedral group Dih_n to represent various symmetries and in Section 5, we set the hyperparameter n to 4. Here we conduct a hyperparameter search to verify this choice. The result is shown in Figure 7.

SARD($n \in \{3\}$), SARD($n \in \{4\}$), SARD($n \in \{5\}$) are SARD with different dihedral group, *i.e.*, Dih_3, Dih_4, Dih_5, respectively. As for SARD($n \in \{3, 4, 5\}$), we use these three groups simultaneously by regarding group elements with the same matrix representations as neighbors.

SARD outperforms all others in most tasks and is only a little worse than SARD($n \in \{5\}$ in Point Navigation task. This result validates our hyperparameter choice.

D. Discussions of Dihedral Groups

In this paper, we use Dihedral groups to describe the symmetry of robots mainly for two reasons. (1) The Dihedral groups are generally enough to represent a wide range of symmetries of the robot's morphologies. This is because the Dihedral groups are generated by basic reflectional and rotational symmetries, which can describe the characteristics of most effective robot morphologies. Besides, related works from biology (Savriama & Klingenberg, 2011; Pappas et al., 2021; Graham et al., 2010) also use the Dihedral group as an effective tool to study the symmetry of real-world creatures. (2) Using larger groups may bring in extra learning complexity and lead to poor performance, even though larger groups can contain more symmetries. In general, the Dihedral group is a good trade-off between expressiveness and complexity

E. Limitations

In this paper, we use the subgroups of dihedral groups Dih_n to represent a wide range of symmetries while still avoiding extra learning complexity. Our method has shown superior efficiency, but the dihedral group is a 2D symmetry group and only contains transformations in the xy-plane. Perhaps because of the influence of gravity, dihedral groups are enough for representing the symmetries of most real-world creatures. However, it is still worthwhile to explore 3D symmetry groups (Savriama & Klingenberg, 2011) in the virtual robot design problem, which might be a promising future work.

In addition, although the idea of symmetry can be applied to a wide range of tasks, it may not be suitable for tasks that do not require symmetry, such as single-arm robotic manipulation tasks where we need to design a robot arm as well as its gripper for a particular manipulation task. Intuitively, efficient designs are mostly asymmetric in these situations, and a symmetry constraint might prevent the arm and manipulator from operating in a more effective way, thus hindering training.