Shared Modular Recurrence in Contextual MDPs for Universal Morphology Control

Laurens Engwegen
Delft University of Technology
Delft, The Netherlands
l.r.engwegen@tudelft.nl

Max Weltevrede
Delft University of Technology
Delft, The Netherlands
m.r.weltevrede@tudelft.nl

Caroline Horsch
Delft University of Technology
Delft, The Netherlands
c.c.h.horsch@tudelft.nl

Daan Brinks
Delft University of Technology
Delft, The Netherlands
d.brinks@tudelft.nl

Wendelin Böhmer
Delft University of Technology
Delft, The Netherlands
j.w.bohmer@tudelft.nl

Abstract

A universal controller for any robot morphology would greatly improve computational and data efficiency. Steps have been made towards such multi-robot control by utilizing *contextual* information about the properties of individual robots and exploiting their modular structure in the architecture of deep reinforcement learning agents. When the robots have highly dissimilar morphologies, however, this becomes a challenging problem, especially when the agent must generalize to new, unseen robots. In this paper, we consider the fact that contextual features are often only partially available, but that they can be recovered through modular interactions. This can allow for better multi-robot control and generalization to contexts that are not seen during training. To this extent, we implement a transformer-based architecture with shared modular recurrence and evaluate its (generalization) performance on a large set of MuJoCo robots. The results show a substantial improvement in zero-shot generalization performance on robots with unseen dynamics, kinematics, and topologies, in four different environments.

1 Introduction

Reinforcement Learning (RL) has shown to be very promising for robotic control [Levine et al., 2016, Kalashnikov et al., 2018, Andrychowicz et al., 2020]. In an effort to close the gap with real-world applications, a lot of work has focused on RL agents that are able to generalize control to different tasks, e.g. manipulating different objects or acting in different environments. Recently, large datasets of robot trajectories have been established and are being used to train such generalizable agents by learning from demonstrations in an offline fashion with foundation models [Brohan et al., 2022, Zitkovich et al., 2023, O'Neill et al., 2024]. Several works show promising possibilities for a single model to adapt not only to different scenes and goals, but also to different embodiments that the model has seen demonstrations of [Doshi et al., 2024, Octo Model Team et al., 2024]. Zero-shot generalization to robots that were not seen during training, however, remains very challenging.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Different robot embodiments, or *morphologies*, can have advantages over each other, dependent on their task and the environments they act in. It is infeasible to train a policy from scratch for every robot: training (or even fine-tuning) a policy for every new robot that we are interested in requires expensive compute, excessive use of data, and often tedious hyperparameter tuning. A universal controller that can generalize control to any robot morphology could drastically improve efficiency. The UNIMAL design space [Gupta et al., 2021] was developed for the evolution of diverse robots that can perform varied tasks. It contains more than 1000 different robots, a small subset of which is shown in Figure 1. The dataset of robots in the UNIMAL design space is suitable for the development and evaluation of RL agents that can generalize control to any robot.

By framing this problem as a multi-task RL problem [Vithayathil Varghese and Mahmoud, 2020], each robot can be considered as a separate task that has some specific *contextual* features, such as the mass of its different limbs or its topology. The goal in this multi-task setting is to learn a universal controller that can control any robot on basis of its observations and context. This is not only challenging because robots can have different action and state spaces, but also because robots with different morphologies and/or dynamics might learn tasks in different ways. The multi-task framework allows us to evaluate the performance of an agent during multi-robot training, and its zero-shot generalization performance [Kirk et al., 2023] to new, unseen robots.

Previous work on universal morphology control exploits contextual features that describe properties of the robot to better represent and distinguish tasks. In this paper, we reason that the context can be considered partially observable and that the available features do not encapsulate the true state needed for optimal (generalizable) control. We build upon modular architectures that were previously found to be effective for robotic control [Gupta et al., 2022, Xiong et al., 2023], and address the essentially unobservable context with shared modular recurrence to retain the system's compatibility with different action spaces while minimizing additional compute. This paper empirically shows that the proposed system improves multi-robot control on varied training morphologies, and enables better zero-shot generalization to unseen test robots. Code to reproduce the experiments is publicly available at `https://github.com/LaurensEngwegen/smr`.

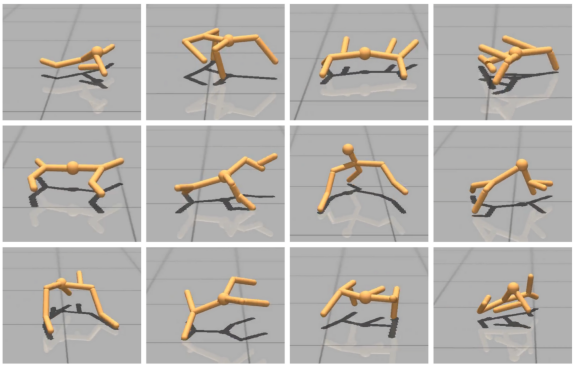

Figure 1: Example of robots that can be found in the UNIMAL design space [Gupta et al., 2021].

2 Background

2.1 Contextual Markov Decision Process

Here, the problem of learning RL policies that are trained on a set of training robots and must generalize to unseen test robots is considered. This problem can be formulated as a Markov Decision Process (MDP) where the agent can only partially access the MDP during training. An MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \rho)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathcal{T}(s_{t+1}|s_t, a_t)$, mapping current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$ to a probability distribution over next states $s_{t+1} \in \mathcal{S}$, reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and initial state distribution $\rho(s_0)$. A Contextual Markov Decision Process (CMDP [Hallak et al., 2015]) is an MDP where states can be decomposed, $s_t = (s'_t, c)$, into the underlying state $s'_t \in \mathcal{S}'$ and a context $c \in \mathcal{C}$. The context is sampled at the start of each episode and remains static until the episode ends. In multi-robot control, the context defines the robot to be controlled. This framework allows to evaluate zero-shot generalization, that is, generalization to contexts not seen during training [Kirk et al., 2023].

The UNIMAL design space contains modular robots [Gupta et al., 2021] that are simulated with the MuJoCo physics engine [Todorov et al., 2012]. Such robots consist of a set of nodes, or limbs, that share the same state space and action space, i.e. $\mathcal{S} = \{\mathcal{S}^i | i = 1, ..., N\}$ and $\mathcal{A} = \{\mathcal{A}^i | i = 1, ..., N\}$ for robots with N limbs. Besides the underlying states, a node-level context, $\mathcal{C} = \{\mathcal{C}^i | i = 1, ..., N\}$,

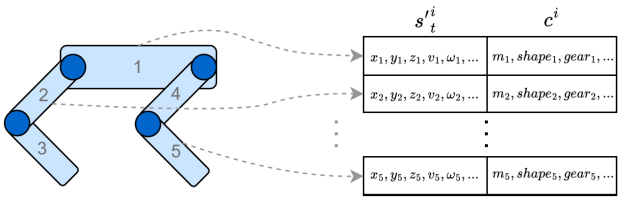

Figure 2: Example of how robot observations are modularized and can be decomposed into limb-wise underlying states and contexts.

is provided in the state that describes information of the limbs (e.g. mass, initial position with respect to the parent limb, and the initial position of each joint attached to the limb). The node-level observation and context features that are provided to the agent are listed in Table 1 in Appendix A. Figure 2 provides an example of a modular robot and how limb-wise observations, consisting of the underlying state s'_t and (observable) context c, compose the global observation.

Previous methods exploited this modular structure in combination with modern architectures for effective multi-robot control [Huang et al., 2020, Kurin et al., 2021, Gupta et al., 2022, Xiong et al., 2023]. Rather than aiming to develop context-agnostic agents, Gupta et al. [2022] and Xiong et al. [2023] condition the agent on the available contextual information with methods coined *MetaMorph* and *ModuMorph*, respectively. In doing so, they did not only show improved performance during training, but also on generalization to unseen robots. However, the gap between the performance on training and testing robots remains substantial.

2.2 Partially Observable Context

In this paper, we recognize and leverage the fact that not all task-relevant contextual information is observable to the agent. In multi-agent control settings like MuJoCo, the missing context consists of structural and abstract properties that are difficult to encode directly in observations or cannot be processed effectively in the agent architecture. First, structural information about the robot is not directly accessible. While the agent receives local positional relationships (e.g., a limb's position relative to its parent), the true adjacency structure of the morphology is not explicitly provided. As noted by Xiong et al. [2023], encoding such graph structure in modular architectures is not trivial due to challenges with positional encoding. Consequently, topology must be inferred indirectly. Moreover, some of the most critical contextual features are inherently abstract. For example, the functional influence that one limb exerts on another during coordinated movement is not directly observable nor easily encoded. Such effects of interaction cannot be captured through static observations alone.

Although the underlying state of a single robot in MuJoCo is treated as fully observable [Todorov et al., 2012], the CMDP is more accurately modeled as partially observable [Ghosh et al., 2021]. Formally, the emission or observation function defined in partially observable MDPs [Spaan, 2012], $\phi : \mathcal{S} \rightarrow \mathcal{O}$, maps a state to an observation $o_t \in \mathcal{O}$ that contains the underlying state and the observable context, c^+, for every timestep t: $o_t = \phi((s'_t, c)) = (s'_t, c^+)$. The unobserved portion of c can include both missing physical parameters and abstract features. In the single robot setting, such information never changes between episodes and can therefore effectively be treated as fully observable. Besides, even in the multi-robot setting, aforementioned methods that rely solely on explicitly provided context features can achieve good performance on training robots: the agent can simply overfit. The challenge arises when the agent must generalize to robots with (abstract) unobservable context features that were not present in the training set.

Incorporating memory into the agent architecture [Hausknecht and Stone, 2015] can allow the agent to implicitly identify and quickly adapt to latent contextual information online. Because the underlying physical state is otherwise observable, any information inferred from history *must* be from the missing context. Crucially, such memory mechanism must preserve the modular structure of the agent's architecture to remain compatible with the multi-robot setting in which input dimensionalities can vary. E.g., a naive encoding of the global action-observation history is not directly compatible with the state-of-the-art modular architectures.

3 Related Work

3.1 Universal Morphology Control

We aim to learn an RL policy that can generalize control to different robot morphologies, even when state and action dimensionalities can change. Effective approaches utilized the modularity in robots [Pathak et al., 2019] and introduced weight sharing across different modules [Huang et al., 2020]. Several works adopted graph neural networks [Scarselli et al., 2008, Wang et al., 2018] or transformers [Vaswani et al., 2017, Kurin et al., 2021] as inductive biases to more explicitly infer relationships between different limbs or modules through message-passing. More recently, multi-robot training has been evaluated on a larger scale of different morphologies with the introduction of the UNIMAL design space [Gupta et al., 2021]. Gupta et al. [2022] constructed training and testing sets of robots to evaluate multi-robot control, and showed the effectiveness of a modular transformer-based approach when contextual information is provided to the agent. By further exploiting this contextual information in the agent's architecture, Xiong et al. [2023] demonstrated improved performance.

Several recent works incorporate additional information about the robot topology through a graph encoding or sparse attention matrices [Patel and Song, 2024, Sferrazza et al., 2024] for improved generalization in simulation and the real world. Fine-tuning and policy distillation approaches have also been proposed for generalization purposes [Przystupa et al., 2025, Xiong et al., 2024]. More complex modular architectures were introduced to improve transfer to unseen robots in simulation and the real world [Bohlinger et al., 2024, Li et al., 2024], although evaluating on smaller sets of, and relatively similar robot morphologies. Bohlinger et al. [2024], specifically, distinguish between different body parts (feet, torso, joints) for targeted processing. When this distinction cannot be made, like in the UNIMAL design space, the architecture is similar to MetaMorph with extra layers. Our work builds upon MetaMorph and ModuMorph, while remaining compatible with the adaptations of the previously mentioned works that can be beneficial in other environments.

None of the aforementioned approaches consider this generalization problem as partially observable. Earlier works did suggest that "implicit system identification" with memory-based policies can benefit robotic control, but did not utilize a modular system, nor evaluate generalization on varied morphologies [Yu et al., 2017, Peng et al., 2018]. LocoFormer Liu et al. [2025] exploits long history contexts, spanning multiple episodes in a meta-learning fashion, to train generalist agents in an environment with procedurally generated robots. The Transformer-XL Dai et al. [2019] model in this setup requires significantly more computational resources, but the improved generalization serves as additional evidence that observation histories can be beneficial, also without a meta-learning setting. We are the first to show that the addition of shared modular recurrence can improve (zero-shot generalization) performance on a diverse set of robot morphologies.

3.2 Neural Architectures

The effectiveness of the transformer architecture [Vaswani et al., 2017] for multi-robot control, lies in its capability to model pairwise dependencies between limbs with self-attention. Self-attention can be defined as $A = \sigma(QK^T/\sqrt{d})V$, with query, key and value matrices $Q, K, V \in \mathbb{R}^{N\times d}$, for robots with N limbs and a hidden size of d. Learnable parameters W_Q, W_K and W_V map the input $X \in \mathbb{R}^{N\times d}$ to those matrices, i.e. $Q = XW_Q$, $K = XW_K$ and $V = XW_V$, and $\sigma(\cdot)$ is a row-wise softmax function. In addition to the attention mechanism, Xiong et al. [2023] utilize hypernetworks [Ha et al., 2017] to more explicitly condition the agent on the available node-wise contextual information. Briefly, they train a hypernetwork that is conditioned on the observable context to generate (1) the parameters of a node-wise encoder that produces X_V with which the value in the subsequent transformer encoder layer is calculated as $V = X_V W_V$, (2) X_Q and X_K, where the query and key matrices are now defined as $Q = X_Q W_Q$, $K = X_K W_K$, respectively, and (3) the parameters of a node-wise decoder that projects the output of the transformer encoder. Recurrent neural networks (RNNs), like LSTM and GRU [Hochreiter and Schmidhuber, 1997, Cho et al., 2014], are useful architectures to deal with partially observable domains in deep RL. Tang and Ha [2021] combined LSTMs with attention to develop systems that can adapt to changes (permutations) of the input. Here, we build upon these methods to develop a transformer-based architecture with shared modular recurrence and hypernetworks to improve multi-robot control.

4 Shared Modular Recurrence

4.1 Recurrent PPO

The goal is to learn a universal control policy that is effective for any robot we can encounter in the UNIMAL design space by training on a limited set of K robots. An effective approach to RL in partially observable domains is to learn an encoding of the belief over the agent's true state. This is often done by, at every timestep t, encoding the action-observation history (AOH) $\tau_t^k = (o_0^k, a_0^k, \ldots, o_{t-1}^k, a_{t-1}^k, o_t^k)$ with an RNN, for (in the current domain) robot k the agent is controlling. In this way, the training objective can be formulated as finding parameters θ for policy $\pi_\theta(a_t^k|\tau_t^k)$ that maximize the (discounted) cumulative reward, averaged over all training robots: $\max_\theta \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{H} \gamma^t r_t^k]$, with task horizon H. We implemented a recurrent version of Proximal Policy Optimization (PPO) [Schulman et al., 2017] to optimize this objective.

To make recurrent training tractable at scale, we incorporate R2D2-style sequence-chunking and burn-in [Kapturowski et al., 2018], originally proposed for (off-policy) DRQN [Hausknecht and Stone, 2015], on the PPO rollout buffer. Since episode lengths can vary from 1 timestep up to $H = 1000$, storing complete, padded episodes for batch processing with RNNs is impractical and inefficient. Instead, we store overlapping chunks of episodes. For each chunk of transitions, the RNN hidden states are initialized with the hidden states that are stored during rollout at the beginning of each chunk. After that, a burn-in phase is applied to reconstruct meaningful hidden states for the current parametrization. We use a chunk size of $m = 80$ and a burn-in period of $l = 20$, as those values were reported by Kapturowski et al. [2018] to be effective. A default recurrent PPO setup for universal morphology control settings is insufficient: naively encoding global action-observation histories breaks both the compatibility with varying input dimensionalities and the structural invariances required for generalization. This motivates our architectural contribution.

4.2 Shared Recurrent Network

In order to retain the agent's modularity, we adopt and adapt a recurrent architecture that processes components of the input separately [Tang and Ha, 2021]: every limb-level action and observation are processed individually through an RNN to encode local AOHs, $\tau_t^i = (o_0^i, a_0^i, \ldots, o_{t-1}^i, a_{t-1}^i, o_t^i)$, for every limb i (we omit the superscript that indicates the robot as our policy is controlling only one robot at a time). Since nodes share the same state space, the parameters of the RNN can be shared to increase scalability and retain modularity. We only keep track of the hidden states of all limbs, $h_t^i = \mathrm{RNN}(o_t^i, a_{t-1}^i, h_{t-1}^i)$, that encode the local AOHs, τ^i, which are initialized with zeros at the start of an episode. In this way, the agent can approximate the relevant history, necessary for implicit inference of the true state, for each limb individually. We build upon state-of-the art architectures for universal morphology control: MetaMorph [Gupta et al., 2022] and ModuMorph [Xiong et al., 2023].

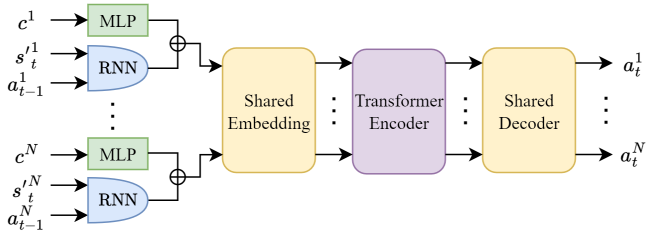

Figure 3: *Recurrent MetaMorph (**R-MeMo**)*

Recurrent MetaMorph. MetaMorph exploited a modular transformer-based architecture in which limb-wise observations are processed through a shared embedding layer, before applying multi-head attention. The resulting embeddings are mapped to actions with a shared decoder. Consult Figure 2 for a depiction of a limb-wise observation, decomposed into underlying state $s_t'^i$, and context c^i. As shown in Figure 3, our recurrent version of MetaMorph (**R-MeMo**) first encodes the underlying state and previous action with an RNN. This is done separately for each limb, but with shared RNN parameters. The hidden state for each limb at the start of each episode chunk is stored in the rollout buffer for initialization during training. The observable context is mapped with an MLP, as this part

of the observation remains constant throughout an episode. The embeddings are added together to establish a latent representation of τ_t.

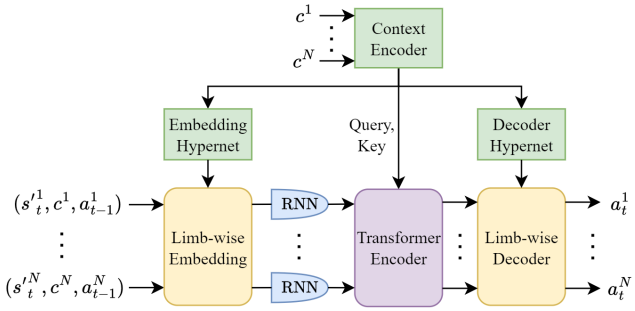

Figure 4: *Recurrent ModuMorph (**R-MoMo**)*

Recurrent ModuMorph. ModuMorph was built on MetaMorph by introducing a hypernetwork that takes context as input and generates the parameters for the embedding layer, the query and key of multi-head attention, and the parameters for the decoder layer. Our recurrent ModuMorph version (**R-MoMo**), shown in Figure 4, uses the same hypernetwork and fixed attention as in the original architecture, but incorporates the shared modular RNN before the transformer to approximate local AOHs from latent encodings of the observation and previous action. The RNN could also be inserted before the embedding layer. This would, however, result in larger inputs for the embedding layer (if we keep the size of the hidden states the same): 128 instead of 54 per limb. This, in turn, leads to an increase of more than 1M trainable parameters in the hypernetwork with respect to ModuMorph. This was avoided to ensure a fair comparison between ModuMorph and R-MoMo.

In both proposed architectures, a transformer receives the local (AOH) encodings from the RNN to infer relationships between different limbs and finally map them to limb-wise actions. The retained modular structure allows for robots with different input dimensionalities (i.e., different numbers of limbs). The introduction of shared modular recurrence only causes a modest increase in trainable parameters, as shown in Table 3 in Appendix B. Augmenting MetaMorph and ModuMorph with MLPs of similar size caused stability problems, demonstrated in Figure 12 in Appendix B, and was therefore avoided. Besides, the individual runs with additional parameters that did show stable learning did not achieve better performance.

5 Experiments

In this Section, experiments are performed with MetaMorph, ModuMorph, and their recurrent counterparts R-MeMo and R-MoMo, respectively. We use the MetaMorph version from Xiong et al. [2023], which they report to perform better than the original implementation.

5.1 Experimental setup

Agents are trained with the dataset of 100 training robots as constructed by Gupta et al. [2022]. We then first evaluate the agent's generalization performance on unseen variations of these training robots, where parameters that influence the dynamics and kinematics are altered (such as the damping of limbs or the angles joints can make). Subsequently, the performance on robots with unseen topologies is evaluated. The provided test set of robots with unseen topologies is randomly split into a validation (32 robots) and test (70 robots) set to experiment with different hyperparameters.

We only tuned one regularization hyperparameter that Xiong et al. [2023] found to have a big impact on performance. See Appendix B for further details. All other hyperparameter values were taken from Xiong et al. [2023] and are listed in Table 2. Parameters of the introduced shared modular recurrence were assigned default values (and, thus, not optimized), with a hidden and cell state size of 128 for the LSTM that was used as RNN. Figure 13 in Appendix C shows that we cannot simply train ModuMorph with (fixed size) recent history as input to observe similar performance gains as with shared modular recurrence. Other ablations, e.g., in which the LSTM is replaced by a GRU, and with recurrence in the hypernetwork, can also be found in Appendix C.

Agents are trained and evaluated in four different environments. In each of those, the agent has to maximize the robot's locomotion distance. In **Flat Terrain**, the agent needs to traverse a flat surface, while in **Incline** the surface is inclined by 10 degrees. **Variable Terrain** contains a sequence of randomly generated hills, steps and rubble, interleaved with flat terrain. Finally, **Obstacles** is a flat terrain with randomly generated obstacles. In the latter two environments, the agent additionally receives a 2D heightmap of its close surrounding to be able to react to changes in terrain. For more details on the environments, we refer to Gupta et al. [2021].

5.2 Multi-Robot Training Performance

The training performance of the different methods on the 100 training robots, averaged over 10 seeds, is shown in Figure 5. Across all environments, the recurrent architectures (R-MeMo and R-MoMo) obtain either similarly high or higher returns than their non-recurrent baselines MetaMorph and ModuMorph. Particularly in the Incline environment, which is more difficult as dynamics play a more important role, shared modular recurrence results in better training performance. In general, ModuMorph seems to perform better than MetaMorph and R-MeMo on the training robots, illustrating the effectiveness of the hypernetworks conditioned on the available context during multi-robot training. The training time of R-MeMo and R-MoMo increases slightly with respect to their baselines, which is reported in Appendix D. However, this is not the case during inference, as the agent then always processes observations sequentially.

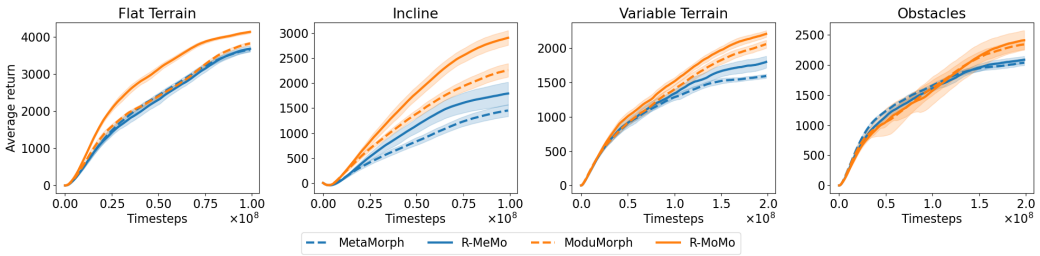

Figure 5: Training performance on the 100 training robots in the different environments. The average return with a 95% confidence interval (CI) over 10 seeds is visualized.

5.3 Zero-Shot Generalization to Different Dynamics and Kinematics

Gupta et al. [2022] constructed a set of robots that have the same topologies as the training robots, but differ in a contextual feature, to evaluate zero-shot generalization to different dynamics or kinematics. For each training robot, they created four test robots with variations in armature, damping, gear, density, limb shapes or joint angles, resulting in a new set of 2400 test robots. Figure 6 shows the performance of the four evaluated methods on the test robots with unseen dynamics and kinematics parameters. Over all changes, R-MoMo obtains a higher average return than ModuMorph, consistently throughout the different environments. Only in Obstacles confidence intervals overlap. Similarly, R-MeMo performs better than MetaMorph, and in some environments and parameter changes even outperforms ModuMorph. These results demonstrate improved generalization performance obtained through the shared modular recurrence across environments with different levels of complexity.



Figure 6: Performance on robots with changes in contextual features that result in unseen dynamics or kinematics. The average return with a 95% CI over 10 seeds is reported.

5.4 Zero-Shot Generalization to Unseen Robot Topologies

Training and test robot topologies, $\mathcal{C}_{\text{train}}$ and $\mathcal{C}_{\text{test}}$, for which $\mathcal{C}_{\text{train}} \cap \mathcal{C}_{\text{test}} = \emptyset$, were constructed by Gupta et al. [2022]. These robots are very diverse, as they have a variable amount of limbs (up to 12), with different context features, and a variable amount of joints per limb (up to 2). After training on the 100 training robots, the methods were evaluated on the 70 test robots with unseen topologies. The averaged returns of the different methods in the four environments are shown in Figure 7. In each of the environments, ModuMorph and/or R-MoMo dominate training, but R-MoMo significantly outperforms the other methods in zero-shot generalization to the test robots. Additionally, R-MeMo is competitive with ModuMorph on the unseen test robots, even though ModuMorph performs better during training. These results show that architectures with shared modular recurrence can learn policies that generalize much better to unseen test robots than their non-recurrent baselines across environments with varying levels of difficulty.

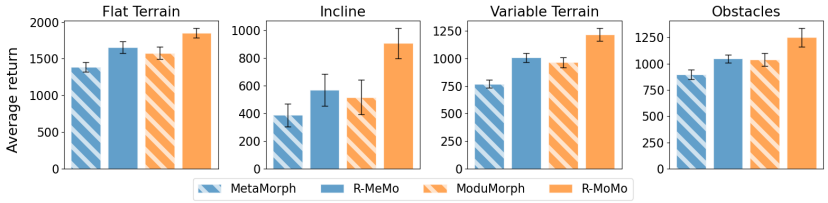

Figure 7: Test performance on robots with unseen topologies in four different environments. The average return with a 95% CI is shown.

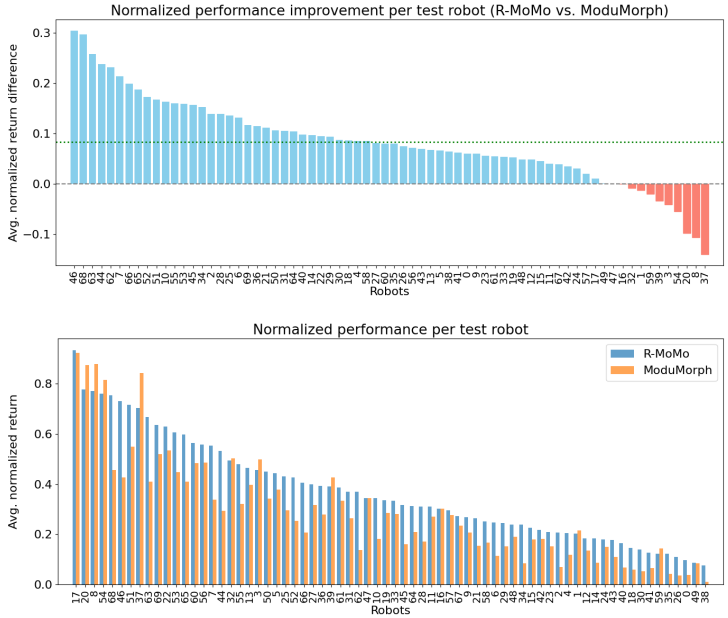

Figure 8: The difference in return between R-MoMo and ModuMorph (top) and the obtained returns (bottom) on all test robots. The returns are normalized for each environment and averaged over 10 seeds. The green dotted line indicates the average performance improvement.

Returns in each environment can range from very low (< 0) to very high (> 4000) values. Averaged returns over 70 test robots could therefore be misleading, as differences can be caused by only a small set of test robots. We therefore additionally report the difference in return between R-MoMo and ModuMorph for every test robot individually, next to the obtained returns, in Figure 8. To compare the methods across all environments, the average returns are normalized using the minimal (52, -74, 71, 126) and maximal (5008, 3751, 3000, 2403) returns found in Flat Terrain, Incline, Variable Terrain, and Obstacles, respectively. Per-robot performance comparisons for every environment separately can be found in Appendix D. This comparison shows a consistent improvement in a majority of the test robots, rather than only on a limited set of robots. Additionally, robots that ModuMorph performs

better on are often also well controlled by R-MoMo. In contrast, ModuMorph struggles to control various robots across environments, while R-MoMo shows substantially fewer robots that are poorly controlled. Analyses in Appendix D validate the consistency of the improved policy throughout the entire episode (rather than only at certain timesteps) for the agents with shared modular recurrence.

5.5 Training Set Size

To further validate the increased generalization performance and scalability of the introduced approach, we performed experiments with different training set sizes. Figure 9 shows the return obtained by the different methods for different amounts of robots in the training set: 25, 50, 75, and the complete set of 100 training robots in Flat Terrain and Incline. The improvement in zero-shot generalization performance of R-MoMo over ModuMorph is consistently visible for the different sizes of training sets, especially from 50 robots onwards. The difference increases when more training robots are available, which is specifically demonstrated in the more difficult Incline environment. This shows the consistency and scalability of R-MoMo with respect to the amount of training examples. Furthermore, these results corroborate the scaling law hypothesis [Ai et al., 2025] stating that test performance scales with the number of available training embodiments. Interestingly, we do observe a difference in the slope of scaling across different methods: specifically in Incline, R-MoMo scales better than its baseline.

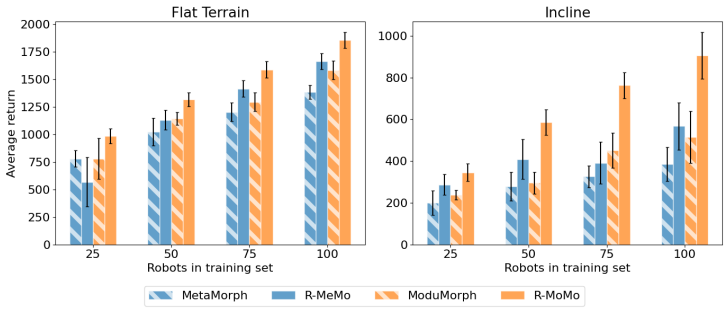

Figure 9: Test performance with smaller training set sizes, averaged over 10 seeds with 95% CI error bars.

5.6 Single Contextual Features

We experimented with a scenario in which a very limited number of contextual features would be available to the agent, to find potential differences in robustness against this lack of information. ModuMorph and R-MoMo were trained and evaluated under this constraint. Test performance using single contextual features (or using *all available* features) in Flat Terrain and Incline is shown in Figure 10. Large error bars (e.g. for ModuMorph with body_mass in Flat Terrain) correspond to highly instable performance across different random seeds.

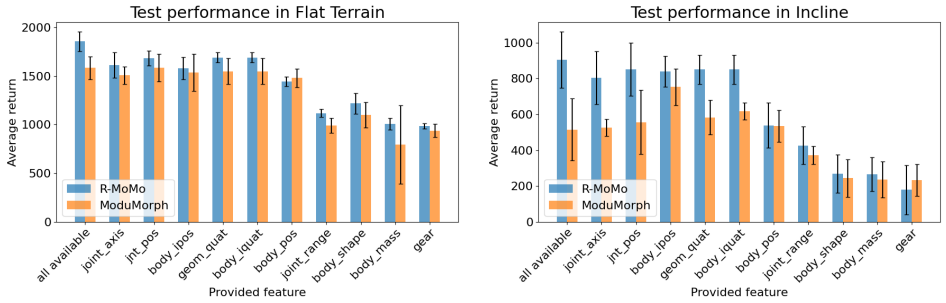

Figure 10: Test performance of ModuMorph and R-MoMo on Flat Terrain and Incline when providing either *all available* contextual features or only a single contextual feature. Mean performance and standard deviation over 5 seeds is shown.

These results show that R-MoMo frequently outperforms ModuMorph in the single feature setting. This observation is more evident in Incline, the more difficult environment. However, this advantage is not consistent across all features: some features are insufficiently informative to induce strong performance when used in isolation for both methods. When the provided contextual feature *is* informative, R-MoMo's performance accelerates more than ModuMorph's. This sensitivity likely

stems from the hypernetwork, which conditions exclusively on the available context. Notably, ModuMorph's performance in the sparse context regime does not drop for the first six features in both environments. A possible explanation is overfitting to the training robots, reducing the effect of the available feature. Future work could investigate such sparse context availability and feature importance in more detail.

6 Discussion

This work explores the introduction of shared modular recurrence in transformer-based architectures for improved multi-robot control. It was hypothesized that modular recurrence could improve performance by enabling the agent to infer relevant latent context from local histories. The results have shown a consistent increase in both training and zero-shot generalization performance when such memory mechanism was introduced across different environments for robots with different dynamics, kinematics, and topologies. This clearly indicates that the RNN extracts some unobservable contextual information from the history for better policies. Moreover, the improvement is consistent over a large set of test robots and for agents trained on varying amounts of training robots.

Experiments in which only a single context feature is provided did show an increase in performance, although not consistently for all features. Specifically in a more difficult environment, informative contextual features can enable the recurrent architecture to learn better policies. The results indicated that the implicit inference of more contextual information is dependent on the quality or informativeness of available context features. This is an interesting observation, as it implies that learning of partially observable contexts greatly benefits from observing some features, but not others. We leave it to future work to characterize what kind of features assist this inference best, and to what extent this is an artifact of the architecture's dependence on the hypernetwork.

A limitation of the explored recurrent architecture, is that hidden states have to be stored for each limb. Scaling to robots with a large number of limbs requires, therefore, more memory. Nonetheless, using episode chunking with a burn-in period during training only requires storing hidden states at the start of a chunk, minimizing the additional memory usage. An interesting direction for future research would be to investigate a more efficient memory mechanism in the architecture. Secondly, the sequential processing of RNNs results in longer training times. This disadvantage only holds for training, due to the inherent sequential processing during inference. Lastly, there is still room for improvement regarding the generalization gap. Nonetheless, the combination of shared modular recurrence with transformers has shown to be promising for multi-robot control and could be effective in other problems with partially observable contexts in graph-like structures.

Acknowledgments and Disclosure of Funding

This project was funded by the Delft AI initiative within the BIOlab. This work was also partially funded by the Dutch Research Council (NWO) project *Reliable Out-of-Distribution Generalization in Deep Reinforcement Learning* with project number OCENW.M.21.234. DB acknowledges support from an ERC Starting Grant (850818 - MULTI-VIsion) and the Convergence Health and Technology (Integrative Neuromedicine flagship). Computational resources in this work were provided by the Delft AI Cluster (DAIC) [2024] at TU Delft (RRID: SCR_025091).

References

Bo Ai, Liu Dai, Nico Bohlinger, Dichen Li, Tongzhou Mu, Zhanxin Wu, K Fay, Henrik I Christensen, Jan Peters, and Hao Su. Towards embodiment scaling laws in robot locomotion. In *Conference on Robot Learning*, pages 3483–3515. PMLR, 2025.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Nico Bohlinger, Grzegorz Czechmanowski, Maciej Krupka, Piotr Kicki, Krzysztof Walas, Jan Peters, and Davide Tateo. One policy to run them all: an end-to-end learning approach to multi-embodiment locomotion. *arXiv preprint arXiv:2409.06366*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, eighth workshop on syntax, semantics and structure in statistical translation*, pages 103–111, 2014.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 2978–2988, 2019.

Delft AI Cluster (DAIC). The Delft AI Cluster (DAIC), RRID:SCR_025091, 2024. URL `https://daic.tudelft.nl/`.

Ria Doshi, Homer Walke, Oier Mees, Sudeep Dasari, and Sergey Levine. Scaling cross-embodied learning: One policy for manipulation, navigation, locomotion and aviation. *arXiv preprint arXiv:2408.11812*, 2024.

Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in RL is difficult: Epistemic POMDPs and implicit partial observability. *Advances in Neural Information Processing Systems*, 34:25502–25515, 2021.

Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *Nature Communications*, 12(1):5721, 2021.

Agrim Gupta, Linxi Fan, Surya Ganguli, and Li Fei-Fei. Metamorph: Learning universal controllers with transformers. *International Conference on Learning Representations*, 2022.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *International Conference on Learning Representations*, 2017.

Assaf Hallak, Dotan Di Castro, and Shie Mannor. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

Matthew J Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *2015 AAAI Fall Symposia*, volume 45, page 141, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

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

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, pages 651–673. PMLR, 2018.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2018.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76: 201–264, 2023.

Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Böhmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. *International Conference on Learning Representations*, 2021.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Boyu Li, Haoran Li, Yuanheng Zhu, and Dongbin Zhao. Mat: Morphological adaptive transformer for universal morphology policy learning. *IEEE Transactions on Cognitive and Developmental Systems*, 16(4):1611–1621, 2024.

Min Liu, Deepak Pathak, and Ananye Agarwal. Locoformer: Generalist locomotion via long-context adaptation. In *Conference on Robot Learning*, pages 532–546. PMLR, 2025.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open X-embodiment: Robotic learning datasets and RT-X models: Open X-embodiment collaboration[0]. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

Austin Patel and Shuran Song. Get-zero: Graph embodiment transformer for zero-shot embodiment generalization. *arXiv preprint arXiv:2407.15002*, 2024.

Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. Learning to control self-assembling morphologies: a study of generalization via modularity. *Advances in Neural Information Processing Systems*, 32, 2019.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

Michael Przystupa, Hongyao Tang, Martin Jagersand, Santiago Miret, Mariano Phielipp, Matthew E Taylor, and Glen Berseth. Efficient morphology-aware policy transfer to new embodiments. *Reinforcement Learning Journal*, 2025.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Carmelo Sferrazza, Dun-Ming Huang, Fangchen Liu, Jongmin Lee, and Pieter Abbeel. Body transformer: Leveraging robot embodiment for policy learning. *arXiv preprint arXiv:2408.06316*, 2024.

Matthijs TJ Spaan. Partially observable markov decision processes. In *Reinforcement learning: State-of-the-art*, pages 387–414. Springer, 2012.

Yujin Tang and David Ha. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 22574–22587, 2021.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Nelson Vithayathil Varghese and Qusay H Mahmoud. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9):1363, 2020.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

Zheng Xiong, Jacob Beck, and Shimon Whiteson. Universal morphology control via contextual modulation. In *International Conference on Machine Learning*, pages 38286–38300. PMLR, 2023.

Zheng Xiong, Risto Vuorio, Jacob Beck, Matthieu Zimmer, Kun Shao, and Shimon Whiteson. Distilling morphology-conditioned hypernetworks for efficient universal morphology control. *International Conference on Machine Learning*, 2024.

Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

A Observations and Context

The observation provided to the agent in the current environments consists of various features for each limb. Table 1 lists all of these features with a short description, the dimensionality of the feature and whether the feature is part of the context. We refer to the MuJoCo documentation [Todorov et al., 2012] for more details.

Table 1: Features of observations and context with description and dimensionality. $*$ indicates that each limb contains this feature twice, as every limb can contain two joints.

Feature	Description	Dim.	Context
body_xpos	Current x,y,z position of each limb	3	No
body_xvelp	Linear velocity of each limb	3	No
body_xvelr	Angular velocity of each limb	3	No
body_xquat	Orientation of each limb	4	No
qpos	Generalized coordinates of each joint	1*	No
qvel	Generalized velocity of each joint	1*	No
body_pos	Initial x,y,z position of limb w.r.t. parent limb	3	Yes
body_ipos	Initial x,y,z position of center of mass w.r.t. parent limb	3	Yes
body_iquat	Inverse quaternion of limb orientation	4	Yes
geom_quat	Quaternion of geom relative to the body	4	Yes
body_mass	Limb mass	1	Yes
body_shape	Limb shape	2	Yes
jnt_pos	Initial (x,y,z) coordinate of each joint	3*	Yes
joint_range	Range of motion (lower and upper bound) of each joint	2*	Yes
joint_axis	Axis of rotation/translation of each joint (one-hot for x,y,z)	3*	Yes
gear	Gear ratio for each joint	1*	Yes

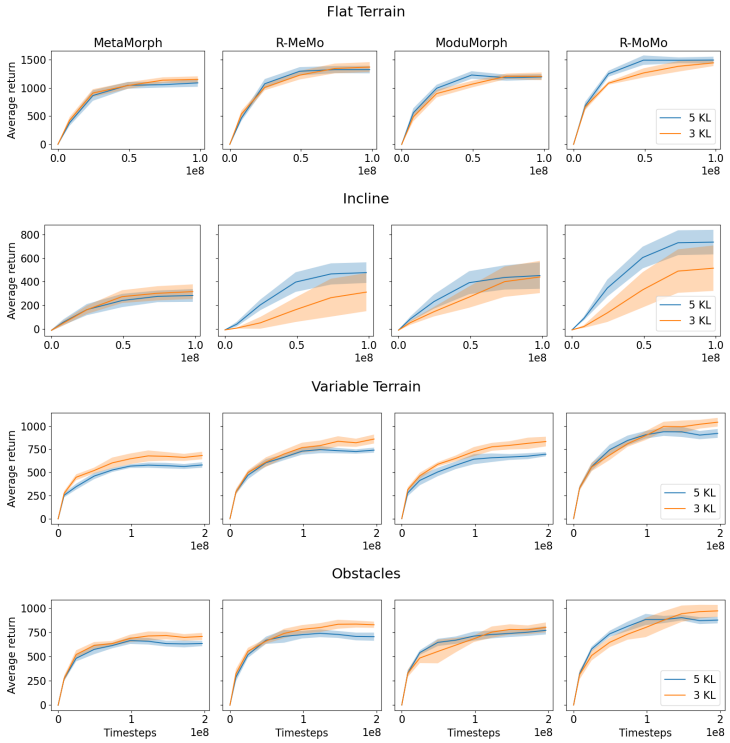

Figure 11: Validation performance on the 32 robots in the validation set for different values of the regularization parameter that defines the maximum approximate KL-divergence between the old and the new policy, averaged over 10 seeds with 95% CIs.

B Implementation Details

B.1 Regularization

In our experiments, we use the same hyperparameter values as in ModuMorph (see Table 2). We only evaluate two different values for a regularization parameter on the validation set. Xiong et al. [2023] namely argued that this parameter can have a big impact on performance. This parameter defines the maximum approximate KL-divergence between the old and the new policy for every mini-batch before the update step. If this value is exceeded, the iteration of updates ends, and we return to sampling new trajectories. Figure 11 shows the average performance of the different methods with the two validated values (3×10^{-2} and 5×10^{-2}) that were also evaluated in ModuMorph. In most cases, there is not a big difference in performance. For every method, the value that resulted in the highest average return on the validation set was used for the reported results in Section 5.

B.2 Hyperparameters

Xiong et al. [2023] provide a link to the datasets of robots and to their implementation on which we built. Table 2 lists all the hyperparameter values that are used for all four methods that are considered in this paper. These values correspond to those used by Xiong et al. [2023]. At the start of each episode rollout, a robot is sampled following the *dynamic replay buffer balancing scheme* introduced by Gupta et al. [2022] (and used in both MetaMorph and ModuMorph). Note that the RNN is only used in R-MeMo and R-MoMo.

Table 2: Hyperparameters used in the experiments with all methods.

	Hyperparameter	**Value**
Environment	Total timesteps	$\{1 \times 10^8, 2 \times 10^8\}$
	Vectorized environments	32
PPO	Discount factor (γ)	0.99
	GAE parameter (λ)	0.95
	PPO clip range (ϵ)	0.2
	Reward normalization	Yes
	Observation normalization	Yes
	Batch size	5120
	Timesteps per rollout	2560
	Updates per rollout	8
	Gradient clipping	0.5
	Clipped value function	Yes
	Value loss coefficient	0.5
Optimizer	Type	Adam
	Epsilon	1×10^{-5}
	Initial learning rate	3×10^{-4}
	Learning rate schedule	Linear warmup, cosine decay
	Warmup iterations	5
Transformer	Number of layers	5
	Number of attention heads	2
	Embedding dimension	128
	Feedforward dimension	1024
	Activation function	ReLU
	Dropout	0.0
RNN	Type	LSTM
	Number of layers	1
	Embedding dimension	128
	Dropout	0.0

B.3 Increased Trainable Parameters

Table 3 shows the number of trainable parameters for all of the methods. The shared modular recurrence introduces a small increase. We experimented with versions of MetaMorph and ModuMorph with a similar number of additional trainable parameters. For MetaMorph, the first embedding block is increased by two fully connected layers (with a hidden size of 256 and ReLU activation). In ModuMorph these layers were added after the first embedding block, where the RNN is placed in R-MoMo. Figure 12 shows that the extra trainable parameters cause instability issues during training. Therefore, we stick to the architectures with the original number of parameters for the other experiments reported in this paper.

Table 3: Number of trainable parameters for each method.

	Trainable parameters
MetaMorph	3,312,515
R-MeMo	3,610,499
ModuMorph	5,232,774
R-MoMo	5,563,526

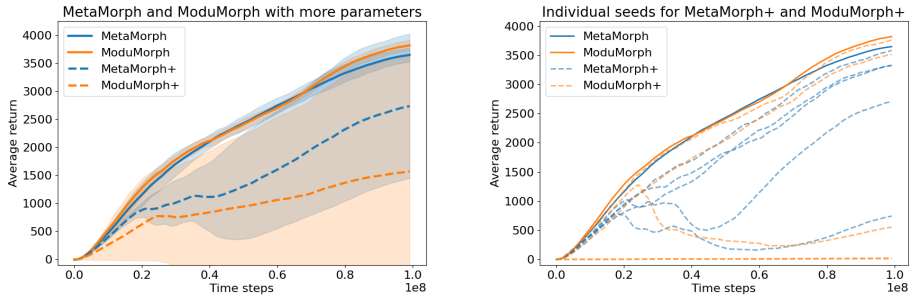

Figure 12: Training performance of MetaMorph and ModuMorph with additional trainable parameters (MetaMorph+ and ModuMorph+), comparable to the amount of additional parameters in the recurrent architectures, in Flat Terrain. The average return over 5 seeds with 95% CIs is shown on the left. On the right, individual seeds are shown for MetaMorph+ and ModuMorph+, which demonstrates that none of the stable runs perform better than the average of the baselines.

C Ablations

C.1 ModuMorph with Fixed History Window

Here we show the performance of versions of ModuMorph that receive a fixed (small) window of previous observations and actions to investigate to what extent the RNN is necessary to capture (long range, generalizable) temporal dependencies or whether a recurrent architecture is not required to find the observed performance differences. Note that stacking multiple observations significantly increases the number of trainable parameters, especially in ModuMorph. Instead of 6,784 parameters to encode a single limb's observation, we need 13,952 parameters for two observations and previous actions. This, in turn, leads to 1,785,856 trainable parameters in the hypernetwork instead of 868,352 (that map a 128-size context embedding to encoder parameters). This quickly explodes for larger windows of observations.
Figure 13 shows performance during training on the left and final test performance on the 70 unseen robot topologies on the right in the Flat Terrain environment. We experiment with default ModuMorph (i.e., window size 1), a history window size of 2, 5, and 10. The results show that performance degrades with larger history windows. It is plausible that this is due to the combination of the larger and more varied input spaces and the substantial increase in parameters, which can require the network to train longer before converging to a good solution, and could make the network more susceptible to overfitting.

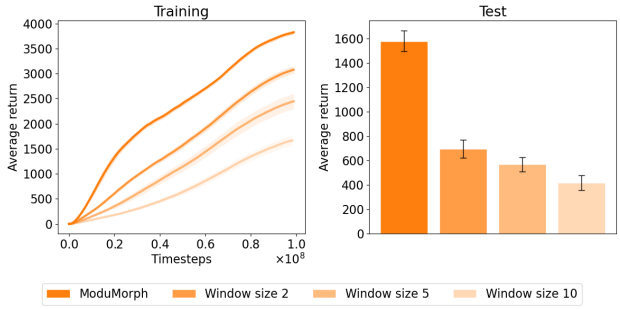

Figure 13: Training (left) and test (right) performance of ModuMorph with different window sizes of past observations (original 1, 2, 5, or 10) in Flat Terrain. Average return with a 95% CI over 10 seeds is shown.

C.2 Modular Recurrence in Hypernetwork

Instead of inserting the RNN (here: LSTM) in the base controller, it can also be introduced in the hypernetwork to learn to infer latent contextual information. In this case, we add the AOH representation to the mapped observable context, to form the input for the blocks that generate the embedding layer, the query and key, and the decoder layer. The original RNN in the base controller (i.e. before the Transformer) is removed. We refer to this variation as HyperR-MoMo and illustrate it in Figure 14. Note that the original hypernetwork only required a forward pass at the start of an episode, since the context does not change during an episode. HyperR-MoMo's hypernetwork requires a forward pass at every timestep and can, thus, generate different parameters at every timestep.

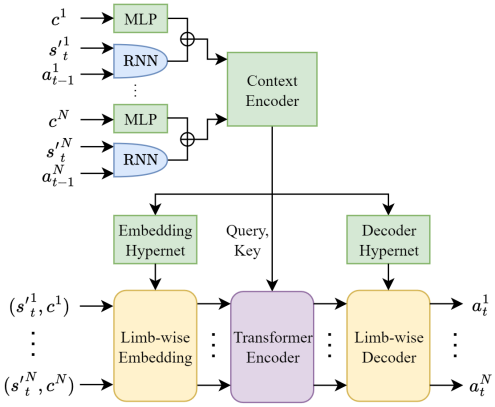

Figure 14: The HyperR-MoMo architecture variation, in which the AOH representation is learned in the hypernetwork instead of in the base network.

The training and zero-shot generalization performance of this architecture variation in Flat Terrain and Incline is shown in Figures 15 and 16, respectively. In Flat Terrain, notably, HyperR-MoMo obtains lower training performance, even though its average test performance is between ModuMorph and R-MoMo. In Incline, HyperR-MoMo shows the lowest training performance, and a similar test performance as ModuMorph. The lower performance and large variance across runs is due to a few runs in which the agent did not learn at all. This could be caused by larger instability due to the fact that the base network can contain a different set of parameters at every timestep (instead of the same set of parameters for each individual robot, as previously discussed). These results indicate that this architecture variation might require additional hyperparameter optimization to reduce instability, but could be promising for universal morphology control. Specifically because its generalization performance is still better than ModuMorph.

C.3 Gated Recurrent Unit

To assess the robustness of the performance gains with shared modular recurrence and to evaluate whether they are specific for LSTMs, we experimented with a GRU. Leaving the rest of the architecture unchanged, the LSTM is swapped with a GRU in R-MoMo. The training performance in Figure 15 and test performance in Figure 16 in Flat Terrain and Incline show that this *GRU-MoMo* is competitive with R-MoMo that uses an LSTM. These results demonstrate that the performance gains are not dependent on the usage of LSTM and provide additional evidence of the necessity of inferring latent contextual features from the history.

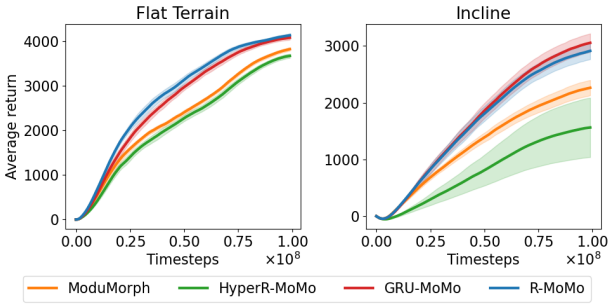

Figure 15: Training performance on the 100 training robots in Flat Terrain and Incline for ModuMorph, HyperR-MoMo, GRU-MoMo (R-MoMo with a GRU), and R-MoMo (with an LSTM). Average return with a 95% confidence interval over 10 seeds is shown.

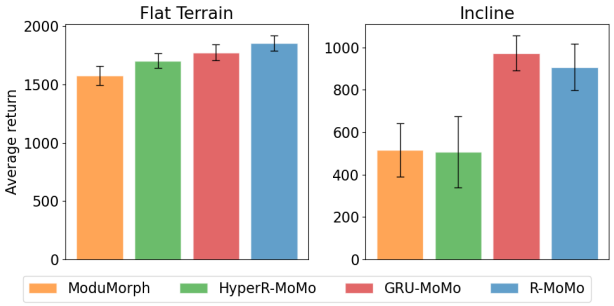

Figure 16: Test performance on the 70 unseen test robots in Flat Terrain and Incline for ModuMorph, HyperR-MoMo, GRU-MoMo (R-MoMo with a GRU), and R-MoMo (with an LSTM). Average return with a 95% confidence interval over 10 seeds is shown.

D Comparisons

D.1 Reward Per Timestep

To qualitatively ensure that the observed performance gains are not the result of a limited number of transitions with a large reward (for example at the very beginning of the episode), we analyzed the reward obtained by the agent at every timestep during an episode. In Figure 17, the obtained average reward per timestep for all test robots is shown for the first 500 timesteps of the episode. In general, the reward slowly increases at the start of the episode when the agent has to apply torques to induce locomotion. In Variable Terrain and Obstacles, the average reward per timestep decreases as soon as the terrain changes or obstacles arise, as these slow down the robot. The differences between the methods demonstrate that the performance improvement with shared modular recurrence over the baselines is consistent over timesteps during the episode.

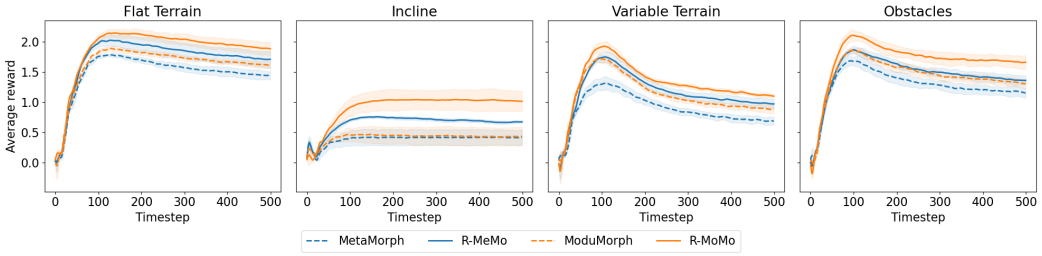

Figure 17: The reward obtained at every timestep in the first half of the episode in the different environments. The obtained rewards are averaged over all test robots and 5 seeds and the standard deviation is shown.

Additionally, to investigate the agent's usage of the RNN hidden states, we performed the same analysis for recurrent agents that receive hidden states consisting of only zeros during the first 100

timesteps of each episode. In Figure 18 we can now see that the recurrent agents execute a weak policy during these timesteps, but are able to recover up to the original reward per timestep observed in Figure 17. This demonstrates that the hidden states are encoding crucial information.

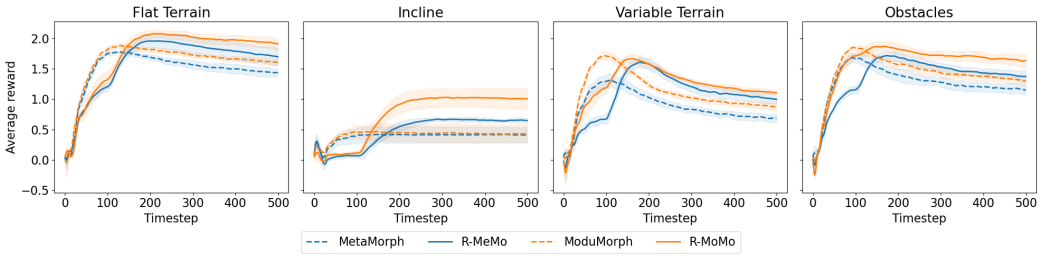

Figure 18: The reward obtained at every timestep in the first half of the episode in the different environments, where the recurrent agents receive zeros as hidden states during the first 100 timesteps. The obtained rewards are averaged over all test robots and 5 seeds and the standard deviation is shown.

D.2 Wall-Clock Training Time

In Table 4 the wall-clock time of the different methods for 100M environmental steps and training in Flat Terrain is reported (note that this includes logging to different destinations and storing model checkpoints). Experiments were performed on NVIDIA A40 GPUs of the Delft AI Cluster (DAIC) [2024]. Chunking the collected episodes before training, and applying the burn-in strategy, introduced by Kapturowski et al. [2018] for DRQN, reduces the extra overhead of sequential processing by the RNNs. Still, training time is increased significantly. However, this difference disappears during inference as data is then processed sequentially for all methods. The trade-off between performance and the reduction of the chunk and/or burn-in size could be further exploited to reduce training time, which we leave for future work.

Table 4: The wall-clock time of the different methods for 100M timesteps of training on Flat Terrain, averaged over 10 seeds.

	Wall-clock training time
MetaMorph	16.3 ± 0.5 hours
R-MeMo	21.8 ± 0.9 hours
ModuMorph	21.9 ± 0.6 hours
R-MoMo	26.0 ± 1.1 hours

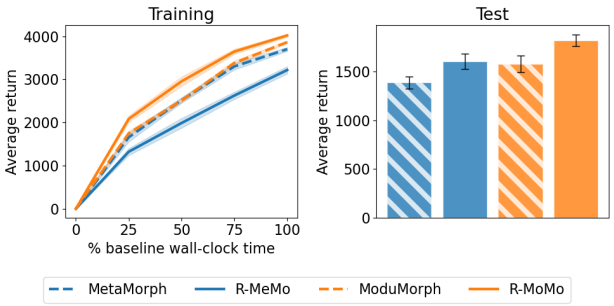

Figure 19: Training (left) and final test (right) performance in Flat Terrain when the recurrent methods receive the same wall-clock training time budget as their baselines (16.3 hours for R-MeMo and 21.9 hours for R-MoMo). Results are averaged over 10 seeds and 95% CIs are shown.

As an additional comparison between the baselines, MetaMorph and ModuMorph, and their recurrent counterparts, R-MeMo and R-MoMo, we report results where each recurrent method is given the same training time (wall-clock time) as its baseline. Thus, R-MeMo is trained for 16.3 hours and R-MoMo is trained for 21.9 hours. Figure 19 shows training performance over these wall-clock times

on the left, and final test performance, as measured with the unseen robot topologies, on the right, in the Flat Terrain environment. Comparing the training results with the original results in Figure 5, it can be observed that MetaMorph now reaches higher training performance than R-MeMo, and the difference between ModuMorph and R-MoMo is smaller. However, even though the training times of R-MeMo and R-MoMo are cut short, the differences in final test performance remain similar to those observed under the full budget of 100M timesteps (in Figure 7).

D.3 Zero-Shot Generalization Performance Improvement Per Environment

The difference in performance on test robots between R-MoMo and ModuMorph is shown in Figures 20, 21, 22 and 23 for the Flat Terrain, Incline, Variable Terrain and Obstacles environments, respectively. These results show increased test performance for R-MoMo on a majority of the test robots across all environments. Moreover, R-MoMo generally struggles with fewer robots (e.g. return below 500) than ModuMorph.

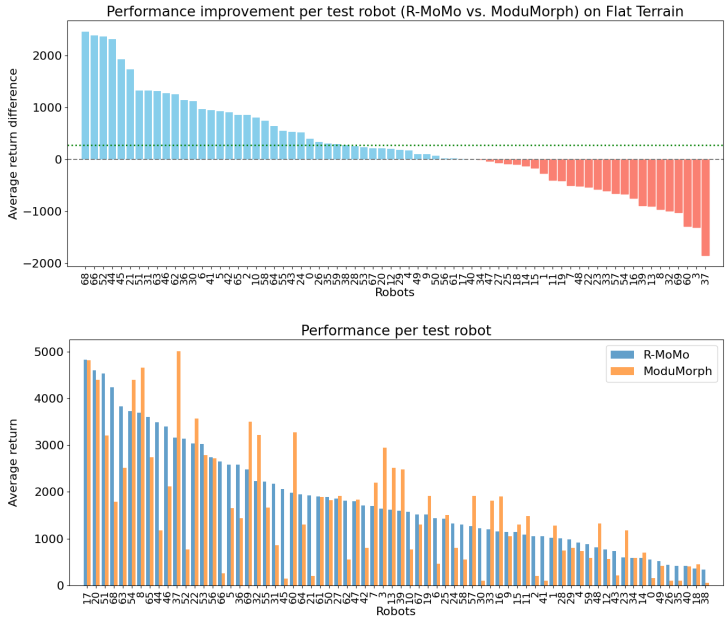

Figure 20: The difference in return between R-MoMo and ModuMorph (top) and the obtained returns (bottom) on each of the 70 unseen test robots in the **Flat Terrain** environment. Returns are averaged over 10 seeds.

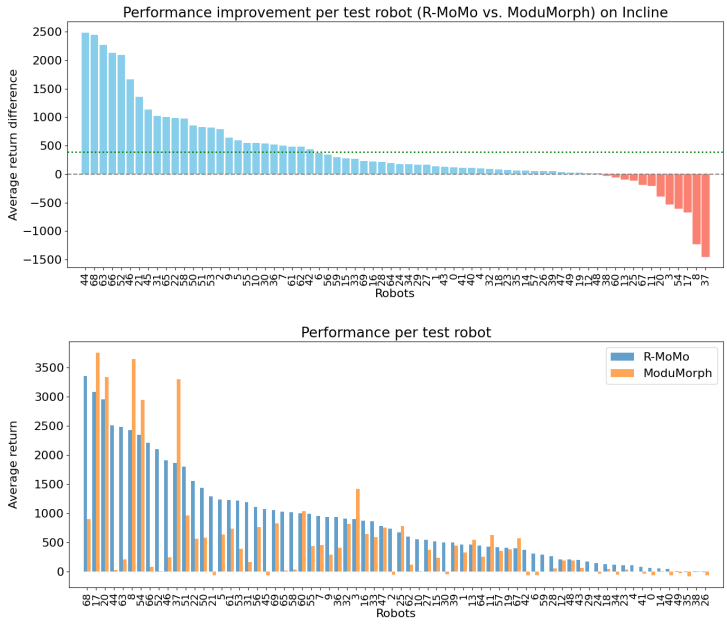

Figure 21: The difference in return between R-MoMo and ModuMorph (top) and the obtained returns (bottom) on each of the 70 unseen test robots in the **Incline** environment. Returns are averaged over 10 seeds.

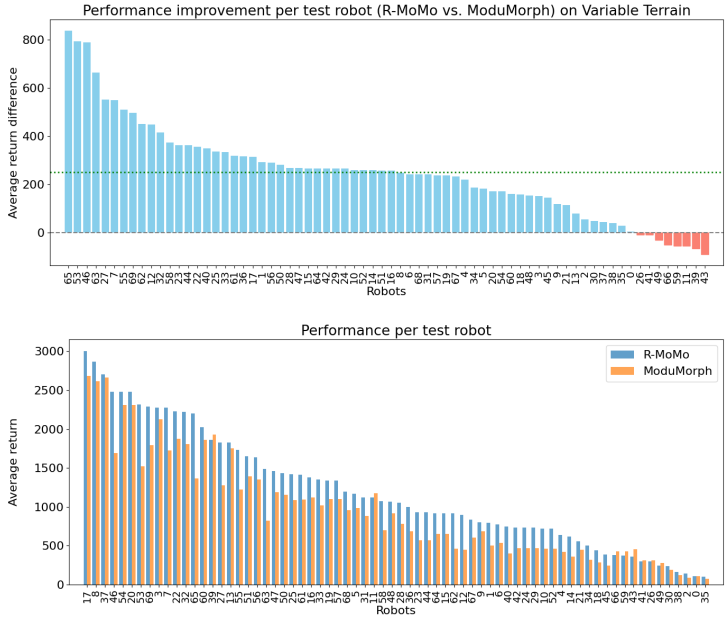

Figure 22: The difference in return between R-MoMo and ModuMorph (top) and the obtained returns (bottom) on each of the 70 unseen test robots in the **Variable Terrain** environment. Returns are averaged over 10 seeds.

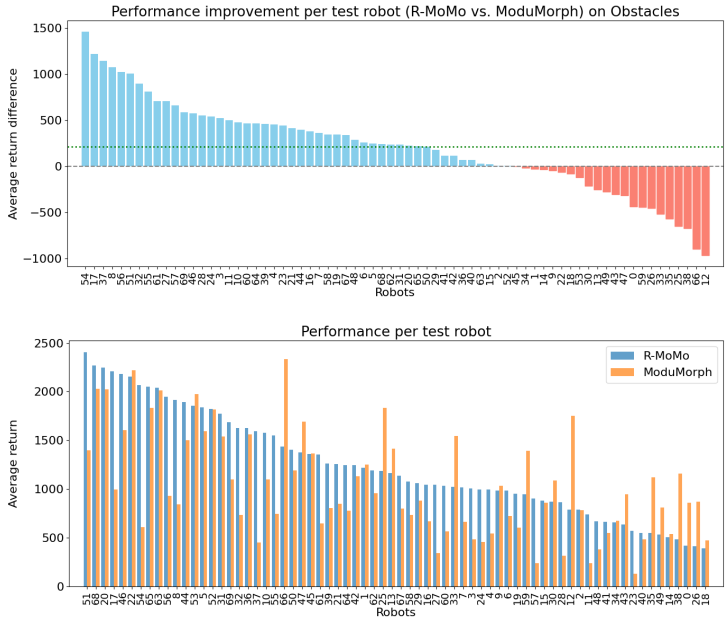

Figure 23: The difference in return between R-MoMo and ModuMorph (top) and the obtained returns (bottom) on each of the 70 unseen test robots in the **Obstacles** environment. Returns are averaged over 10 seeds.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The scope of multi-robot control and the contribution of empirically shown improved (zero-shot generalization) performance through shared modular recurrence by considering contextual features as partially observable are outlined in the abstract and introduction, and are further elaborated on in the rest of the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Sections 5.2, 5.5, 5.6, and 6 discuss the limitations of the proposed approach, such as its computational efficiency, its scaling with dataset size, and (extreme) settings in which it is less effective.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.

- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include theoretical results.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The proposed architectural contribution and its accompanying alterations in the training procedure are fully described in Section 4, and the empirical evaluation is described in Section 5, enabling reproducibility. In addition, code containing the implementation and experimentation will be made publicly available.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.

- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Upon acceptance, data and code to reproduce all the main experimental results will be made open access. At submission time, the link to data and code is not included to retain anonymity. Appendix B describes all hyperparameters and details to reproduce the experiments.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We use the same setting for our main experiments as Gupta et al. [2022], Xiong et al. [2023], and describe all deviations, additional experiments, and hyperparameters (Appendix B) sufficiently in the paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The main experimental results are averaged over 10 runs with different random seeds and 95% confidence intervals are reported. For additional analyses that required more compute, results were averaged over 5 seeds and standard deviations were reported. The paper always reports which of the two is shown in the dedicated descriptions.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Information on the required computer resources is provided in Appendix B and D, and should be sufficient to reproduce the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: There is no conflict between this work and the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no direct path to societal impact of the work, as the advancement of generalizable multi-robot control is done in simulation and still far from deployment in the real world.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The creators of the original data (training and testing sets of robots) and baseline models (MetaMorph and ModuMorph) are properly credited.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The code that will be made publicly available upon acceptance is well documented.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: There was no usage of LLMs for important, original, or non-standard components of the core methods in this research.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.