MoVie: Visual Model-Based Policy Adaptation for View Generalization

Sizhe Yang[12*], **Yanjie Ze**[13*], **Huazhe Xu**[415]
[1]Shanghai Qi Zhi Institute, [2]University of Electronic Science and Technology of China,
[3]Shanghai Jiao Tong University, [4]Tsinghua University, [5]Shanghai AI Lab
[*] Equal contribution
yangsizhe.github.io/MoVie

Abstract

Visual Reinforcement Learning (RL) agents trained on limited views face significant challenges in generalizing their learned abilities to unseen views. This inherent difficulty is known as the problem of *view generalization*. In this work, we systematically categorize this fundamental problem into four distinct and highly challenging scenarios that closely resemble real-world situations. Subsequently, we propose a straightforward yet effective approach to enable successful adaptation of visual **Mo**del-based policies for **Vie**w generalization (**MoVie**) during test time, without any need for explicit reward signals and any modification during training time. Our method demonstrates substantial advancements across all four scenarios encompassing a total of **18** tasks sourced from DMControl, xArm, and Adroit, with a relative improvement of $\mathbf{33}\%$, $\mathbf{86}\%$, and $\mathbf{152}\%$ respectively. The superior results highlight the immense potential of our approach for real-world robotics applications. Code and videos are available at yangsizhe.github.io/MoVie.

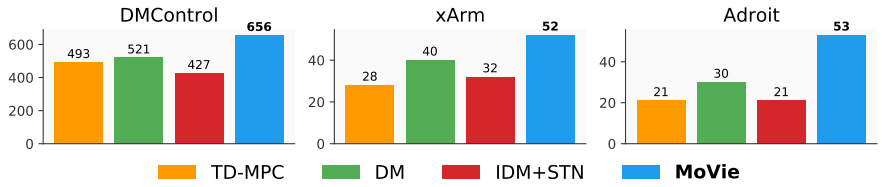

Figure 1: **View generalization results across 3 domains.** Our method MoVie largely improves the test-time view generalization ability, especially in robotic manipulation domains.

1 Introduction

Visual Reinforcement Learning (RL) has achieved great success in various applications such as video games [17, 18], robotic manipulation [22], and robotic locomotion [34]. However, one significant challenge for real-world deployment of visual RL agents remains: a policy trained with very limited views (commonly one single fixed view) might not generalize to unseen views. This challenge is especially pronounced in robotics, where a few fixed views may not adequately capture the variability of the environment. For instance, the RoboNet dataset [3] provides diverse views across a range of manipulation tasks, but training on such large-scale data only yields a moderate success rate ($10\% \sim 20\%$) for unseen views [3].

Recent efforts have focused on improving the visual generalization of RL agents [1, 7, 9, 36]. However, these efforts have mainly concentrated on generalizing to different appearances and backgrounds. In contrast, view generalization presents a unique challenge as the deployment view is unknown and may move freely in the 3D space. This might weaken the weapons such as data augmentations [12, 14, 15, 33] that are widely used in appearance generalization methods. Meanwhile, scaling up domain

37th Conference on Neural Information Processing Systems (NeurIPS 2023), New Orleans.

https://doi.org/10.52202/075280-0940

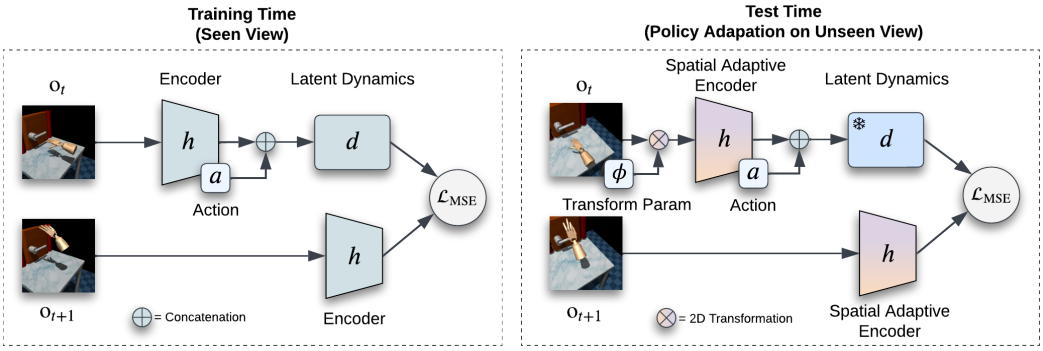

Figure 2: **Overview of MoVie**. During training (left), the agent is trained with the latent dynamics loss. At test time (right), we freeze the dynamics model and modify the encoder as a spatial adaptive encoder to adapt the agents to novel views.

randomization [19, 21, 31] to all possible views is usually unrealistic because of the large cost and the offline nature of existing robot data. With these perspectives combined, it is difficult to apply those common approaches to address view generalization.

In this work, we commence by explicitly formulating the test-time view generalization problem into four challenging settings: *a)* **novel view**, where the camera changes to a fixed novel position and orientation; *b)* **moving view**, where the camera moves continuously around the scene, *c)* **shaking view**, where the camera experiences constant shaking, and *d)* **novel FOV**, where the field of view (FOV) of the camera is altered initially. These settings cover a wide spectrum of scenarios when visual RL agents are deployed to the real world. By introducing these formulations, we aim to advance research in addressing view generalization challenges and facilitate the utilization of robot datasets [3] and deployment on physical robotic platforms.

To address the view generalization problem, we argue that *adaptation* to novel views during test time is crucial, rather than aiming for view-invariant policies. We propose MoVie, a simple yet effective method for adapting visual **Mo**del-based policies to generalize to unseen **Vie**ws. MoVie leverages collected transitions from interactions and incorporates spatial transformer networks (STN [13]) in shallow layers, using the learning objective of the dynamics model (DM). Notably, MoVie requires no modifications during training and is compatible with various visual model-based RL algorithms. It only necessitates small-scale interactions for adaptation to the deployment view.

We perform extensive experiments on 7 robotic manipulation tasks (Adroit hand [20] and xArm [7]) and 11 locomotion tasks (DMControl suite [30])), across the proposed 4 view generalization settings, totaling $\mathbf{18} \times \mathbf{4}$ configurations. MoVie improves the view generalization ability substantially, compared to strong baselines including the inverse dynamics model (IDM [7]) and the dynamics model (DM). Remarkably, MoVie attains a relative improvement of 86% in xArm and 152% in Adroit, underscoring the potential of our method in robotics. **We are committed to releasing our code and testing platforms.** To conclude, our contributions are three-fold:

- We formulate the problem of view generalization in visual reinforcement learning with a wide range of tasks and settings that mimic real-world scenarios.
- We propose a simple model-based policy adaptation method for view generalization (**MoVie**), which incorporates STN into the shallow layers of the visual representation with a self-supervised dynamics prediction objective.
- We successfully showcase the effectiveness of our method through extensive experiments. The results serve as a testament to its capability and underscore its potential for practical deployment in robotic systems, particularly with complex camera views.

2 Related Work

Visual generalization in reinforcement learning. Agents trained by reinforcement learning (RL) from visual observations are prone to overfitting the training scenes, making it hard to generalize to

https://doi.org/10.52202/075280-0940

unseen environments with appearance differences. A large corpus of recent works has focused on addressing this issue [7, 9, 10, 14, 15, 19, 21, 31, 33, 35, 36]. Notably, SODA [10] provides a visual generalization benchmark to better evaluate the generalizability of policies, while they only consider appearance changes of agents and backgrounds. Distracting control suite [26] adds both appearance changes and camera view changes into DMControl [30], where the task diversity is limited.

View generalization in robotics. The field of robot learning has long grappled with the challenge of training models on limited views and achieving generalization to unseen views. Previous studies, such as RoboNet [3], have collected extensive video data encompassing various manipulation tasks. However, even with pre-training on such large-scale datasets, success rates on unseen views have only reached approximately $10\% \sim 20\%$ [3]. In recent efforts to tackle this challenge, researchers have primarily focused on third-person imitation learning [23–25] and view-invariant visual representations [2, 4, 16, 32, 37], but these approaches are constrained by the number of available camera views. In contrast, our work addresses a more demanding scenario where agents trained on a single fixed view are expected to generalize to diverse unseen views and dynamic camera settings.

Test-time training. There is a line of works that train neural networks at test-time with self-supervised learning in computer vision [5, 28, 29], robotics [27], and visual RL [7]. Specifically, PAD is the closest to our work [7], which adds an inverse dynamics model (IDM) objective into model-free policies for both training time and test time and gains better appearance generalization. In contrast, we differ in a lot of aspects: *(i)* we focus on visual model-based policies, *(ii)* we require no modification in training time, and *(iii)* our method is designed for view generalization specifically.

3 Preliminaries

Formulation. We model the problem as a Partially Observable Markov Decision Process (POMDP) $\mathcal{M} = \langle \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathbf{o} \in \mathcal{O}$ are observations, $\mathbf{a} \in \mathcal{A}$ are actions, $\mathcal{F} : \mathcal{O} \times \mathcal{A} \mapsto \mathcal{O}$ is a transition function (called dynamics as well), $r \in \mathcal{R}$ are rewards, and $\gamma \in [0, 1)$ is a discount factor. During training time, the agent's goal is to learn a policy π that maximizes discounted cumulative rewards on $\mathcal{M}$, *i.e.*, $\max \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$. During test time, the reward signal from the environment is not accessible to agents and only observations are available, which are possible to experience subtle changes such as appearance changes and camera view changes.

Model-based reinforcement learning. TD-MPC [11] is a model-based RL algorithm that combines model predictive control and temporal difference learning. TD-MPC learns a visual representation $\mathbf{z} = h(\mathbf{o})$ that maps the high-dimensional observation $\mathbf{o} \in \mathcal{O}$ into a latent state $\mathbf{z} \in \mathcal{Z}$ and a latent dynamics model $d : \mathcal{Z} \times \mathcal{A} \mapsto \mathcal{Z}$ that predicts the future latent state $\mathbf{z}' = d(\mathbf{z}, \mathbf{a})$ based on the current latent state $\mathbf{z}$ and the action $\mathbf{a}$. **MoDem** [8] accelerates TD-MPC with efficient utilization of expert demonstrations $\mathcal{D} = \{D_1, D_2, \cdots, D_N\}$ to solve challenging tasks such as dexterous manipulation [20]. In this work, we select TD-MPC and MoDem as the backbone algorithm to train model-based agents, while our algorithm could be easily extended to most model-based RL algorithms such as Dreamer [6].

4 Method

We propose a simple yet effective method, visual **Mo**del-based policy adaptation for **Vie**w generalization (**MoVie**), which can accommodate visual RL agents to novel camera views at test time.

Learning objective for the test time. Given a tuple $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$, the original latent state dynamics prediction objective can be written as

$$\mathcal{L}_{\text{dynamics}} = \|d(h(\mathbf{o}_t), \mathbf{a}_t) - h(\mathbf{o_{t+1}})\|_2, \tag{1}$$

where h is an image encoder that projects a high-dimensional observation from space $\mathcal{O}$ into a latent space $\mathcal{Z}$ and d is a latent dynamics model $d : \mathcal{Z} \times \mathcal{A} \mapsto \mathcal{Z}$.

In test time, the observations under unseen views lie in a different space $\mathcal{O}'$, so that their corresponding latent space also changes to $\mathcal{Z}'$. However, the projection h learned in training time can only map $\mathcal{O} \mapsto \mathcal{Z}$ while the policy π only learns the mapping $\mathcal{Z} \mapsto \mathcal{A}$, thus making the policy hard to generalize to the correct mapping function $\mathcal{Z}' \mapsto \mathcal{A}$. Our proposal is thus to adapt the projection h from a mapping function $h : \mathcal{O} \mapsto \mathcal{Z}$ to a more useful mapping function $h' : \mathcal{O}' \mapsto \mathcal{Z}$ so that the

21509 https://doi.org/10.52202/075280-0940

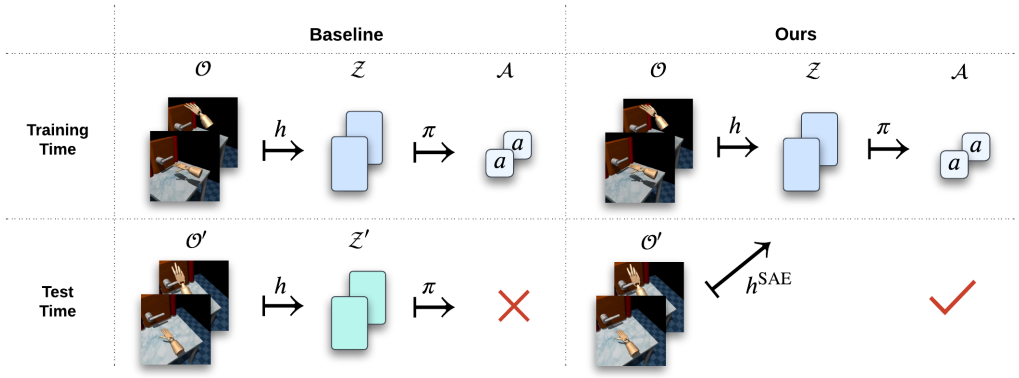

Figure 3: **An illustration of the reason why MoVie is effective.** We treat the frozen dynamics model as the source of supervision to adapt the latent space of h^{SAE} to that of the training views.

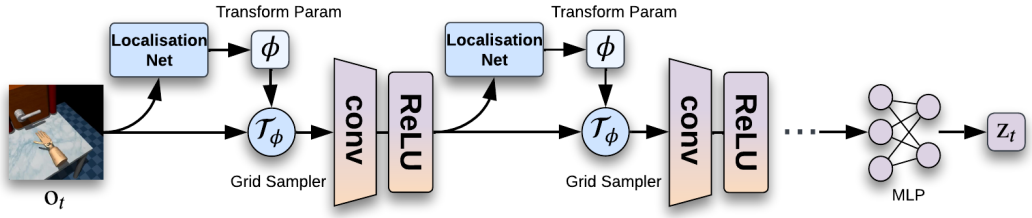

Figure 4: **The architecture of our spatial adaptive encoder (SAE).** We incorporate STNs before the first two convolutional layers of the encoder for better adaptation of the representation h.

policy would execute the correct mapping $\mathcal{Z} \mapsto \mathcal{A}$ without training. A vivid illustration is provided in Figure 3.

We freeze the latent dynamics model d, denoted as $d^\star$, so that the latent dynamics model is not a training target but a supervision. We also insert STN blocks [13] into the shallow layers of h to better adapt the projection h, so that we write h as h^{SAE} (SAE denotes spatial adaptive encoder). Though the objective is still the latent state dynamics prediction loss, the supervision here is **superficially identical but fundamentally different** from training time. The formal objective is written as

$$\mathcal{L}_{\text{view}} = \|d^\star(h^{\text{SAE}}(\mathbf{o}), \mathbf{a}) - h^{\text{SAE}}(\mathbf{o_{t+1}})\|_2. \tag{2}$$

Spatial adaptive encoder. We now describe more details about our modified encoder architecture during test time, referred to as spatial adaptive encoder (SAE). To keep our method simple and fast to adapt, we only insert two different STNs into the original encoder, as shown in Figure 4. We observe in our experiments that transforming the low-level features (*i.e.*, RGB features and shallow layer features) is most critical for adaptation, while the benefit of adding more STNs is limited (see Table 6). An STN block consists of two parts: *(i)* a localisation net that predicts an affine transformation with 6 parameters and *(ii)* a grid sampler that generates an affined grid and samples features from the original feature map. The point-wise affine transformation is written as

$$\begin{pmatrix} x^s \\ y^s \end{pmatrix} = \mathcal{T}_\phi(G) = \mathrm{A}_\phi \begin{pmatrix} x^t \\ y^t \\ 1 \end{pmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \end{bmatrix} \begin{pmatrix} x^t \\ y^t \\ 1 \end{pmatrix} \tag{3}$$

where G is the sampling grid, (x_i^t, y_i^t) are the target coordinates of the regular grid in the output feature map, (x_i^s, y_i^s) are the source coordinates in the input feature map that define the sample points, and A_ϕ is the affine transformation matrix.

Training strategy for SAE. We use a learning rate 1×10^{-5} for STN layers and 1×10^{-7} for the encoder. We utilize a replay buffer with size 256 to store history observations and update 32 times for each time step to heavily utilize the online data. Implementation details remain in Appendix A.

https://doi.org/10.52202/075280-0940

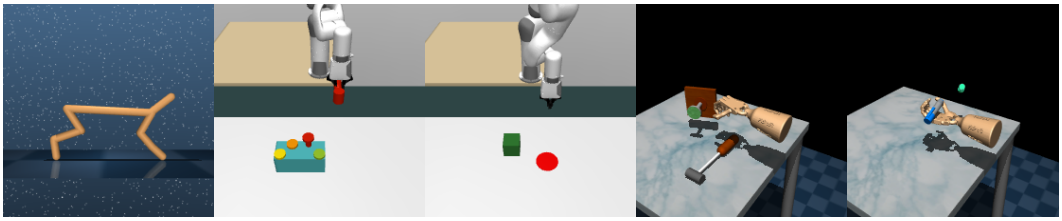

Figure 5: **Visualization of three task domains**, including DMControl [30], xArm [7], Adroit [20].

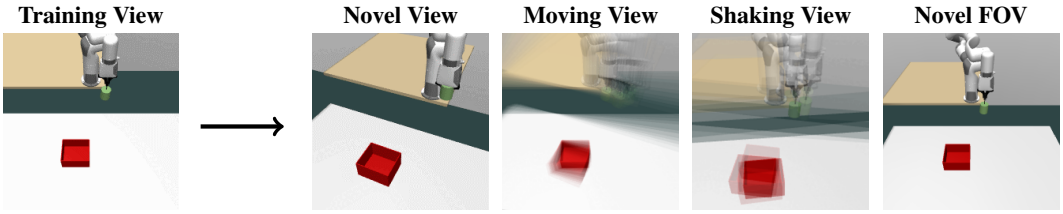

Figure 6: **Visualization of the training view and four view generalization settings.** Trajectory visualization for all tasks is available on our website yangsizhe.github.io/MoVie.

5 Experiments

In this section, we investigate how well an agent trained on a single fixed view generalizes to unseen views during test time. During the evaluation, agents have no access to reward signals, presenting a significant challenge for agents to self-supervise using online data.

5.1 Experiment Setup

Formulation of camera view variations: *a)* **novel view**, where we maintain a fixed camera target while adjusting the camera position in both horizontal and vertical directions by a certain margin, *b)* **moving view**, where we establish a predefined trajectory encompassing the scene and the camera follows this trajectory, moving back and forth for each time step while focusing on the center of the scene, *c)* **shaking view**, where we add Gaussian noise onto the original camera position at each time step, and *d)* **novel FOV**, where the FOV of the camera is altered once, different from the training phase. A visualization of four settings is provided in Figure 6 and videos are available in yangsizhe.github.io/MoVie for a better understanding of our configurations. Details remain in Appendix B.

Tasks. Our test platform consists of **18** tasks from **3** domains: 11 tasks from DMControl [30], 3 tasks from Adroit [20], and 4 tasks from xArm [7]. A visualization of the three domains is in Figure 5. Although the primary motivation for this study is for addressing view generalization in real-world robot learning, which has not yet been conducted, we contend that the extensive range of tasks tackled in our research effectively illustrates the potential of MoVie for real-world application. We run 3 seeds per experiment with seed numbers $0, 1, 2$ and run 20 episodes per seed. During these 20 episodes, the models could store the history transitions. We report cumulative rewards for DMControl tasks and success rates for robotic manipulation tasks, averaging over episodes.

Baselines. We first train visual model-based policies with TD-MPC [11] on DMControl and xArm environments and MoDem [8] on Adroit environments under the default configurations and then test these policies in the view generalization setting. Since MoVie consists of two components mainly (*i.e.*, DM and STN), we build two test-time adaptation algorithms by replacing each module: *a)* **DM**, which removes STN from MoVie and *b)* **IDM+STN**, which replaces DM in MoVie with IDM [7]. IDM+STN is very close to PAD [7], and we add STN for fair comparison. We keep all training settings the same.

21511 https://doi.org/10.52202/075280-0940

Table 1: **Experiment conclusion across all domains and all settings.** The best method on each setting is in **bold** and the relative improvement over TD-MPC is also reported.

Setting	Domain	TD-MPC	DM	IDM+STN	MoVie
Novel view	DMControl	395.61	508.85 (↑29%)	378.80 (↓4%)	**623.19** (↑57%)
	xArm	16	36 (↑125%)	18 (↑13%)	**46** (↑188%)
	Adroit	8	14 (↑75%)	12 (↑50%)	**34** (↑325%)
Moving view	DMControl	605.98	611.87 (↑1%)	502.34 (↓17%)	**673.74** (↑11%)
	xArm	20	31 (↑55%)	24 (↑20%)	**42** (↑110%)
	Adroit	15	20 (↑33%)	15 (↑0%)	**45** (↑200%)
Shaking view	DMControl	441.79	348.26 (↓21%)	291.45 (↓26%)	**558.23** (↑26%)
	xArm	42	44 (↑5%)	44 (↑5%)	**45** (↑7%)
	Adroit	30	35 (↑17%)	26 (↓13%)	**63** (↑110%)
Novel FOV	DMControl	527.47	613.74 (↑16%)	542.43 (↑3%)	**770.56** (↑46%)
	xArm	34	47 (↑38%)	40 (↑18%)	**75** (↑121%)
	Adroit	31	51 (↑65%)	31 (↑0%)	**68** (↑119%)
All settings	DMControl	492.71	520.68 (↑6%)	426.76 (↓13%)	**656.43** (↑33%)
	xArm	28	40 (↑43%)	32 (↑14%)	**52** (↑86%)
	Adroit	21	30 (↑43%)	21 (↑0%)	**53** (↑152%)

Table 2: **Results in novel view.** The best method on each task is in **bold**.

Novel view	TD-MPC	DM	IDM+STN	MoVie
Cheetah, run	90.13±20.38	254.43±13.89	127.76±16.16	**342.39±54.95**
Walker, walk	249.34±14.78	262.65±62.86	215.90±29.30	**512.71±404.25**
Walker, stand	568.01±16.81	635.64±13.70	508.82±15.77	**679.90±23.03**
Walker, run	127.07±8.06	**131.65±2.20**	94.49±18.72	94.49±18.87
Cup, catch	922.83±30.74	949.36±9.20	932.75±23.34	**961.98±2.68**
Finger, spin	137.30±4.47	750.65±9.90	141.03±37.52	**892.01±1.20**
Finger, turn-easy	380.80±138.54	528.91±160.25	348.00±123.84	**705.36±71.98**
Finger, turn-hard	261.43±72.83	312.36±134.28	306.76±181.13	**331.83±18.98**
Pendulum, swingup	40.65±3.72	64.51±2.17	82.46±8.19	**528.76±77.51**
Reacher, easy	981.25±4.80	982.30±0.83	875.31±150.91	**984.66±2.78**
Reacher, hard	592.91±59.96	724.91±125.84	533.50±96.90	**821.03±67.89**
DMControl	395.61	508.85	378.80	**623.19** (↑57%)
xArm, reach	3±2	85±13	1±2	**95±0**
xArm, push	46±5	45±8	**60±5**	48±7
xArm, peg in box	5±0	5±15	3±2	**6±7**
xArm, hammer	10±10	10±10	8±5	**36±12**
xArm	16	36	18	**46** (↑188%)
Adroit, door	0±0	10±5	0±0	**66±7**
Adroit, hammer	6±2	11±10	**21±2**	11±5
Adroit, pen	18±5	20±8	15±5	**25±18**
Adroit	8	14	12	**34** (↑325%)

5.2 Main Experiment Results

Considering the extensive scale of our conducted experiments, we present an overview of our findings in Table 1 and Figure 1. The detailed results for four configurations are provided in Table 2, Table 3, Table 4, and Table 5 respectively. We then detail our findings below.

Superiority of MoVie across domains, especially in robotic manipulation. It is observed that irrespective of the domain or configuration, MoVie consistently outperforms TD-MPC without any adaptation and other methods that apply DM and IDM. This large improvement highlights the fundamental role of our straightforward approach in addressing view generalization. Additionally, we observe that MoVie exhibits greater suitability for robotic manipulation tasks. We observe a significant relative improvement of **86**% in xArm tasks and **152**% in Adroit tasks, as opposed to a comparatively modest improvement of only 33% in DMControl tasks; see Table 1. We attribute this

https://doi.org/10.52202/075280-0940

Table 3: **Results in moving view.** The best method on each task is in **bold**.

Moving view	TD-MPC	DM	IDM+STN	MoVie
Cheetah, run	235.37±63.39	344.70±7.54	199.48±25.78	**365.22±42.66**
Walker, walk	632.77±15.62	707.60±26.72	373.44±30.85	**810.19±7.77**
Walker, stand	**803.97±15.75**	706.05±5.75	576.87±44.69	712.48±11.67
Walker, run	**295.54±4.39**	250.84±7.02	190.02±2.61	281.43±33.34
Cup, catch	887.20±3.67	910.51±4.12	915.16±7.31	**951.26±10.68**
Finger, spin	636.91±3.91	697.11±8.67	532.90±3.25	**896.00±21.65**
Finger, turn-easy	715.45±19.23	728.93±180.26	683.85±112.64	**744.45±16.54**
Finger, turn-hard	**593.46±48.22**	454.58±88.50	559.20±121.35	558.26±25.64
Pendulum, swingup	26.23±1.85	83.88±10.55	47.20±4.39	**236.81±50.23**
Reacher, easy	**984.10±4.38**	981.78±0.60	695.03±236.91	982.58±1.39
Reacher, hard	854.76±31.64	864.55±556.37	752.55±83.55	**872.46±51.30**
`DMControl`	605.98	611.87	502.34	**673.74** (↑11%)
xArm, reach	15±5	71±2	21±2	**80±21**
xArm, push	58±5	52±10	55±5	**73±5**
xArm, peg in box	0±0	0±0	1±2	**5±8**
xArm, hammer	8±7	0±0	**10±5**	8±7
`xArm`	20	31	24	**42** (↑110%)
Adroit, door	0±0	10±5	0±0	**66±7**
Adroit, hammer	25±8	31±12	28±02	**43±18**
Adroit, pen	20±0	20±10	18±14	**26±2**
`Adroit`	15	20	15	**45** (↑200%)

Table 4: **Results in shaking view.** The best method on each task is in **bold**.

Shaking view	TD-MPC	DM	IDM+STN	MoVie
Cheetah, run	381.04±39.31	317.66±9.57	212.31±19.74	**493.54±56.80**
Walker, walk	662.13±57.71	627.77±21.47	340.48±36.92	**835.99±14.98**
Walker, stand	**834.70±1.71**	604.55±16.20	471.30±69.19	687.96±9.11
Walker, run	251.58±13.69	186.83±3.75	128.31±9.19	**291.39±7.94**
Cup, catch	752.86±41.81	835.16±32.91	648.75±12.77	**951.20±21.01**
Finger, spin	89.55±4.72	88.56±5.86	145.68±2.93	**284.05±473.73**
Finger, turn-easy	**717.60±40.93**	449.10±20.48	656.23±50.35	694.20±91.02
Finger, turn-hard	411.41±40.28	383.96±32.94	132.40±34.25	**629.20±110.57**
Pendulum, swingup	23.73±9.24	**57.41±4.50**	27.30±2.38	48.23±30.25
Reacher, easy	529.58±65.05	225.08±31.02	299.11±59.00	**771.38±150.60**
Reacher, hard	205.53±25.16	54.80±32.45	144.11±5.60	**453.48±381.60**
`DMControl`	441.79	348.26	291.45	**558.23** (↑26%)
xArm, reach	76±2	76±7	83±2	**88±20**
xArm, push	**64±3**	55±10	61±5	57±13
xArm, peg in box	3±2	**15±13**	3±2	5±5
xArm, hammer	25±5	28±7	28±12	**30±13**
`xArm`	42	44	44	**45** (↑7%)
Adroit, door	1±2	10±0	5±5	**83±2**
Adroit, hammer	58±12	65±18	55±20	**70±27**
Adroit, pen	30±5	30±8	18±5	**35±0**
`Adroit`	30	35	26	**63** (↑110%)

disparity to two factors: *a)* the inherent complexity of robotic manipulation tasks and *b)* the pressing need for effective view generalization in this domain.

Challenges in handling shaking view. Despite improvements of MoVie in various settings, we have identified a relative weakness in addressing the *shaking view* scenario. For instance, in xArm tasks, the success rate of MoVie is only 45%, which is close to the 42% success rate of TD-MPC without adaptation. Other baselines such as DM and IDM+STN also experience performance drops. We acknowledge the inherent difficulty of the shaking view scenario, while it is worth noting that in real-world robotic manipulation applications, cameras often exhibit smoother movements or are positioned in fixed views, partially mitigating the impact of shaking view.

https://doi.org/10.52202/075280-0940

Table 5: **Novel FOV.** The best method on each task is in **bold**.

Novel FOV	TD-MPC	DM	IDM+STN	MoVie
Cheetah, run	128.55±6.57	379.01±10.90	299.02±88.47	**532.94±19.74**
Walker, walk	239.54±129.47	882.21±12.75	579.58±47.87	**920.18±8.76**
Walker, stand	679.70±21.21	732.07±14.77	678.46±55.32	**785.52±24.45**
Walker, run	184.28±1.39	337.28±11.40	295.99±10.00	**482.03±13.84**
Cup, catch	940.20±28.21	912.70±34.54	964.83±10.45	**973.60±1.94**
Finger, spin	675.81±4.99	886.63±4.54	414.83±79.20	**917.21±1.71**
Finger, turn-easy	792.26±145.24	690.61±140.88	805.63±54.95	**845.35±35.52**
Finger, turn-hard	591.43±109.84	393.18±118.38	449.88±121.99	**640.21±183.23**
Pendulum, swingup	216.90±78.56	184.33±83.99	253.36±78.26	**699.90±50.64**
Reacher, easy	929.40±12.35	931.56±12.69	937.20±45.97	**971.48±11.07**
Reacher, hard	424.11±83.74	421.56±94.93	287.96±6.33	**707.75±86.05**
`DMControl`	527.47	613.74	542.43	**770.56** (↑46%)
xArm, reach	53±12	98±2	85±5	**100±0**
xArm, push	60±13	71±5	60±5	**81±2**
xArm, peg in box	20±5	13±10	6±11	**63±11**
xArm, hammer	3±2	6±5	8±2	**55±5**
`xArm`	34	47	40	**75** (↑121%)
Adroit, door	1±2	58±10	3±5	**81±12**
Adroit, hammer	66±7	75±18	76±10	**81±10**
Adroit, pen	26±10	20±8	15±5	**41±7**
`Adroit`	31	51	31	**68** (↑119%)

Effective adaptation in novel view, moving view, and novel FOV scenarios. In addition to the shaking view setting, MoVie consistently outperforms TD-MPC without adaptation by $2\times \sim 4\times$ in robotic manipulation tasks across the other three settings. It is worth noting that these three settings are among the most common scenarios encountered in real-world applications.

Real-world implications. Our findings have important implications for real-world deployment of robots. Previous methods, relying on domain randomization and extensive data augmentation during training, often hinder the learning process. Our proposed method enables direct deployment of offline or simulation-trained agents, improving success rates with minimal interactions.

5.3 Ablations

To validate the rationale behind several choices of MoVie and our test platform, we perform a comprehensive set of ablation experiments.

Integration of STN with low-level features. To enhance view generalization, we incorporate two STN blocks [13], following the image observation and the initial convolutional layer of the feature encoder. This integration is intended to align the low-level features with the training view, thereby preserving the similarity of deep semantic features to the training view. As shown in Table 6, by progressively adding more layers to the feature encoder, we observe that deeper layers do not provide significant additional benefits, supporting our intuition for view generalization.

Table 6: **Ablation on applying different numbers of STNs from shallow to deep.** The best method on each setting is in **bold**.

Cheetah-run	0	1	2 (MoVie)	3	4
Novel View	254.42±13.89	259.64±32.01	**342.39±54.95**	309.25±33.19	327.39±10.44
Moving View	344.70±7.54	360.29±13.64	**365.22±42.66**	361.28±6.78	357.29±31.24
Shaking View	317.66±9.57	491.96±20.07	**493.54±56.80**	454.82±19.57	472.57±8.49
Novel FOV	379.01±10.90	512.69±17.67	**532.94±19.74**	505.65±12.02	492.79±21.66

Different novel views. We classify novel views into three levels of difficulty, with our main experiments employing the *medium* difficulty level by default. Table 7 presents additional results for

https://doi.org/10.52202/075280-0940

the *easy* and *hard* difficulty levels. As the difficulty level increases, we observe a consistent decrease in the performance of all the methods.

Table 7: **Ablation on different novel views.** The best method on each setting is in **bold**.

Cheetah-run	TD-MPC	DM	IDM+STN	MoVie
Easy	265.90±7.94	441.61±14.00	349.86±55.82	**466.67±58.94**
Medium	90.13±20.38	254.43±13.89	127.76±16.16	**342.39±54.95**
Hard	48.64±13.20	112.39±8.36	59.64±5.35	**215.36±62.03**

Table 8: **Ablation on IDM with and without STN.** The best method is in **bold**.

Task	Setting	IDM	IDM+STN
xArm, push	Novel view	40±5	**60±5**
	Moving view	46±2	**55±5**
	Shaking view	60±10	**61±5**
	Novel FOV	58±2	**60±5**
	`All settings`	51	**59**
xArm, hammer	Novel view	6±5	**8±5**
	Moving view	5±0	**10±5**
	Shaking view	**35±8**	28±12
	Novel FOV	6±7	**8±2**
	`All settings`	13	**14**
Cup, catch	Novel view	867.87±3.14	**932.75±23.34**
	Moving view	912.85±10.22	**915.16±7.31**
	Shaking view	435.40±87.32	**648.75±12.77**
	Novel FOV	815.27±34.61	**964.83±10.45**
	`All settings`	757.84	**865.37**
Finger, spin	Novel view	**177.26±2.49**	141.03±37.52
	Moving view	332.90±0.28	**532.90±3.25**
	Shaking view	106.72±7.11	**145.68±2.93**
	Novel FOV	311.66±8.55	**414.83±79.20**
	`All settings`	232.13	**301.86**
Cheetah, run	Novel view	111.73±18.98	**127.76±16.16**
	Moving view	**205.19±25.45**	199.48±25.78
	Shaking view	210.06±27.99	**212.32±19.74**
	Novel FOV	262.76±62.17	**299.03±88.47**
	`All settings`	197.43	**209.43**

The efficacy of STN in conjunction with IDM. Curious readers might be concerned about our direct utilization of IDM+STN in the main experiments, suggesting that STN could potentially be detrimental to IDM. However, Table 8 shows that STN not only benefits our method but also improves the performance of IDM, thereby demonstrating effectiveness of SAE for adaptation of the representation and validating our baseline selection.

Finetune or freeze the DM. In our approach, we employ the frozen DM as a form of supervision to guide the adaptation process of the encoder. However, it remains unverified for the readers whether end-to-end finetuning of both the DM and the encoder would yield similar benefits. The results presented in Table 9 demonstrate that simplistic end-to-end finetuning does not outperform MoVie, thereby reinforcing the positive results obtained by MoVie.

https://doi.org/10.52202/075280-0940

Table 9: **Ablation on whether to finetune the DM.** The best method is in **bold**.

Task	Setting	Finetune DM	MoVie
xArm, push	Novel view	**51**$\pm$**2**	48$\pm$7
	Moving view	48$\pm$7	**73**$\pm$**5**
	Shaking view	45$\pm$5	**57**$\pm$**13**
	Novel FOV	68$\pm$5	**81**$\pm$**2**
	`All settings`	53	**64**
xArm, hammer	Novel view	10$\pm$5	**36**$\pm$**12**
	Moving view	0$\pm$0	**8**$\pm$**7**
	Shaking view	5$\pm$5	**30**$\pm$**13**
	Novel FOV	11$\pm$2	**55**$\pm$**5**
	`All settings`	6	**32**
Cup, catch	Novel view	648.90$\pm$25.668	**961.98**$\pm$**2.68**
	Moving view	676.05$\pm$79.69	**951.26**$\pm$**10.68**
	Shaking view	228.20$\pm$7.84	**951.20**$\pm$**21.01**
	Novel FOV	658.50$\pm$8.48	**973.60**$\pm$**1.94**
	`All settings`	552.91	**959.51**
Finger, spin	Novel view	278.57$\pm$26.34	**892.01**$\pm$**1.20**
	Moving view	192.95$\pm$72.76	**896.00**$\pm$**21.65**
	Shaking view	1.50$\pm$0.70	**284.05**$\pm$**473.73**
	Novel FOV	372.82$\pm$51.63	**917.21**$\pm$**1.71**
	`All settings`	211.46	**747.31**
Cheetah, run	Novel view	273.01$\pm$11.29	**342.39**$\pm$**54.95**
	Moving view	331.55$\pm$22.22	**365.22**$\pm$**42.66**
	Shaking view	476.25$\pm$30.25	**493.54**$\pm$**56.80**
	Novel FOV	**561.94**$\pm$**37.73**	532.94$\pm$19.74
	`All settings`	410.68	**433.52**

6 Conclusion

In this study, we present MoVie, a method for adapting visual model-based policies to achieve view generalization. MoVie mainly finetunes a spatial adaptive image encoder using the objective of the latent state dynamics model during test time. Notably, we maintain the dynamics model in a frozen state, allowing it to function as a form of self-supervision. Furthermore, we categorize the view generalization problem into four distinct settings: novel view, moving view, shaking view, and novel FOV. Through a systematic evaluation of MoVie on 18 tasks across these four settings, totaling 64 different configurations, we demonstrate its general and remarkable effectiveness.

One limitation of our work is the lack of real robot experiments, while our focus is on addressing view generalization in robot datasets and deploying visual reinforcement learning agents in real-world scenarios. In our future work, we would evaluate MoVie on real-world robotic tasks.

Acknowledgment

This work is supported by National Key R&D Program of China (2022ZD0161700).

References

[1] David Bertoin, Adil Zouitine, Mehdi Zouitine, and Emmanuel Rachelson. Look where you look! saliency-guided q-networks for generalization in visual reinforcement learning. *NeurIPS*, 2022. 1, 17

[2] Boyuan Chen, Pieter Abbeel, and Deepak Pathak. Unsupervised learning of visual 3d keypoints for control. In *ICML*, 2021. 3

[3] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. *CoRL*, 2019. 1, 2, 3

[4] Danny Driess, Ingmar Schubert, Pete Florence, Yunzhu Li, and Marc Toussaint. Reinforcement learning with neural radiance fields. *arXiv*, 2022. 3

[5] Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei Efros. Test-time training with masked autoencoders. *NeurIPS*, 2022. 3

[6] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv*, 2020. 3

[7] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *ICLR*, 2021. 1, 2, 3, 5, 13, 17

[8] Nicklas Hansen, Yixin Lin, Hao Su, Xiaolong Wang, Vikash Kumar, and Aravind Rajeswaran. Modem: Accelerating visual model-based reinforcement learning with demonstrations. *ICLR*, 2023. 3, 5, 13

[9] Nicklas Hansen, Hao Su, and Xiaolong Wang. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. *NeurIPS*, 2021. 1, 3, 17

[10] Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *International Conference on Robotics and Automation*, 2021. 3

[11] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *ICML*, 2022. 3, 5, 13

[12] Nicklas Hansen, Zhecheng Yuan, Yanjie Ze, Tongzhou Mu, Aravind Rajeswaran, Hao Su, Huazhe Xu, and Xiaolong Wang. On pre-training for visuo-motor control: Revisiting a learning-from-scratch baseline. *ICML*, 2023. 1

[13] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *NeurIPS*, 2015. 2, 4, 8

[14] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *NeurIPS*, 2020. 1, 3

[15] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv preprint arXiv:1910.05396*, 2019. 1, 3

[16] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control. In *CoRL*, 2022. 3

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv*, 2013. 1

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015. 1

[19] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *ICRA*, 2018. 2, 3

[20] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *RSS*, 2018. 2, 3, 5, 13

[21] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv*, 2019. 2, 3

21517 https://doi.org/10.52202/075280-0940

[22] Rutav Shah and Vikash Kumar. Rrl: Resnet as representation for reinforcement learning. *ICML*, 2021. 1

[23] Jinghuan Shang and Michael S Ryoo. Self-supervised disentangled representation learning for third-person imitation learning. In *IROS*, 2021. 3

[24] Pratyusha Sharma, Deepak Pathak, and Abhinav Gupta. Third-person visual imitation learning via decoupled hierarchical controller. *NeurIPS*, 2019. 3

[25] Bradly C Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *ICLR*, 2017. 3

[26] Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite–a challenging benchmark for reinforcement learning from pixels. *arXiv*, 2021. 3

[27] Yu Sun, Wyatt L Ubellacker, Wen-Loong Ma, Xiang Zhang, Changhao Wang, Noel V Csomay-Shanklin, Masayoshi Tomizuka, Koushil Sreenath, and Aaron D Ames. Online learning of unknown dynamics for model-based controllers in legged locomotion. *RA-L*, 2021. 3

[28] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *ICML*, 2020. 3

[29] Yu Sun, Xiaolong Wang, Liu Zhuang, John Miller, Moritz Hardt, and Alexei A. Efros. Test-time training with self-supervision for generalization under distribution shifts. In *ICML*, 2020. 3

[30] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv*, 2018. 2, 3, 5, 13

[31] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017. 2, 3

[32] Hsiao-Yu Fish Tung, Zhou Xian, Mihir Prabhudesai, Shamit Lal, and Katerina Fragkiadaki. 3d-oes: Viewpoint-invariant object-factorized environment simulators. *arXiv*, 2020. 3

[33] Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. *NeurIPS*, 2020. 1, 3

[34] Ruihan Yang, Minghao Zhang, Nicklas Hansen, Huazhe Xu, and Xiaolong Wang. Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers. *ICLR*, 2021. 1

[35] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv*, 2021. 3, 17

[36] Zhecheng Yuan, Zhengrong Xue, Bo Yuan, Xueqian Wang, Yi Wu, Yang Gao, and Huazhe Xu. Pre-trained image encoder for generalizable visual reinforcement learning. *NeurIPS*, 2022. 1, 3, 17

[37] Yanjie Ze, Nicklas Hansen, Yinbo Chen, Mohit Jain, and Xiaolong Wang. Visual reinforcement learning with self-supervised 3d representations. *RA-L*, 2023. 3

https://doi.org/10.52202/075280-0940

Appendix

A Implementation Details

In this section, we describe the implementation details of our algorithm for training on the training view and test time training in view generalization settings on the DMControl [30], xArm [7], and Adroit [20] environments. We utilize the official implementation of TD-MPC [11] and MoDem [8] which are available at github.com/nicklashansen/tdmpc and github.com/facebookresearch/modem as the model-based reinforcement learning codebase. During training time, we use the default hyperparameters in official implementation of TD-MPC and MoDem. We present relevant hyperparameters during both training and test time in Table 10 and Table 11. One seed of our experiments could be run on a single 3090 GPU with fewer than 2GB and it takes $\sim$ 1 hours for test-time training.

Training time setup. We train visual model-based policies with TD-MPC on DMControl and xArm environments, and MoDem on Adroit environments, We employ identical network architecture and hyperparameters as original TD-MPC and MoDem during training time.

The network architecture of the encoder in original TD-MPC is composed of a stack of 4 convolutional layers, each with 32 filters, no padding, stride of 2, 7×7 kernels for the first one, 5×5 kernels for the second one and 3×3 kernels for all others, yielding a final feature map of dimension $3 \times 3 \times 32$ (inputs whose framestack is 3 have dimension $84 \times 84 \times 9$). After the convolutional layers, a fully connected layer with an input size of 288 performs a linear transformation on the input and generates a 50-dimensional vector as the final output.

The network architecture of the encoder in original Modem is composed of a stack of 6 convolutional layers, each with 32 filters, no padding, stride of 2, 7×7 kernels for the first one, 5×5 kernels for the second one and 3×3 kernels for all others, yielding a final feature map of dimension $2 \times 2 \times 32$ (inputs whose framestack is 2 have dimension $224 \times 224 \times 6$). After the convolutional layers, a fully connected layer with an input size of 128 performs a linear transformation on the input and generates a 50-dimensional vector as the final output.

Test time training setup. During test time, we train spatial adaptive encoder (SAE) to adapt to view changes. We insert STN blocks before and after the first convolutional layer of the original encoders in TD-MPC and MoDem. The original encoders are augmented by inserting STN blocks, resulting in the formation of SAE. Particularly, for the STN block inserted before the first convolutional layer, the input is a single frame. This means that when the frame stack size is N, N individual frames are fed into this STN block. This is done to apply different transformations to different frames in cases of moving and shaking view.

To update the SAE, we collect online data using a buffer with a size of 256. For each update, we randomly sample 32 (observation, action, next_observation) tuples from the buffer as a batch. The optimization objective is to minimize the loss in predicting the dynamics of the latent states, as defined in Equation 2.

During testing on each task, we run 20 consecutive episodes, although typically only a few or even less than one episode is needed for the test-time training to converge. To make efficient use of the data collected with minimal interactions, we employ a multi-update strategy. After each interaction with the environment, the SAE is updated 32 times.

The following is the network architecture of the first STN block inserted into the encoder of TD-MPC.

```
STN_Block_0_TDMPC(
  (localization): Sequential(
    # By default, each image consists of three channels. Each frame in the
        ↪ observation is treated as an independent input to the STN.
    (0): Conv2d(in_channels=3, out_channels=8, kernel_size=7, stride=1)
    (1): MaxPool2d(kernel_size=4, stride=4, padding=0)
    (2): ReLU()
    (3): Conv2d(in_channels=8, out_channels=10, kernel_size=5, stride=1)
    (4): MaxPool2d(kernel_size=4, stride=4, padding=0)
    (5): ReLU()
  )
  (fc_loc): Sequential(
    (0): Linear(in_dim=90, out_dim=32)
```


https://doi.org/10.52202/075280-0940

```
    (1): ReLU()
    (2): Linear(in_dim=32, out_dim=6)
  )
)
```

The following is the network architecture of the second STN block inserted into the encoder of TD-MPC.

```
STN_Block_1_TDMPC(
  (localization): Sequential(
    (0): Conv2d(in_channels=32, out_channels=8, kernel_size=7, stride=1)
    (1): MaxPool2d(kernel_size=3, stride=3, padding=0)
    (2): ReLU()
    (3): Conv2d(in_channels=8, out_channels=10, kernel_size=5, stride=1)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0)
    (5): ReLU()
  )
  (fc_loc): Sequential(
    (0): Linear(in_dim=90, out_dim=32)
    (1): ReLU()
    (2): Linear(in_dim=32, out_dim=6)
  )
)
```

The following is the network architecture of the first STN block inserted into the encoder of MoDem.

```
STN_Block_0_MoDem(
  (localization): Sequential(
    # By default, each image consists of three channels. Each frame in the
        ↪ observation is treated as an independent input to the STN.
    (0): Conv2d(in_channels=3, out_channels=5, kernel_size=7, stride=2)
    (1): MaxPool2d(kernel_size=4, stride=4, padding=0)
    (2): ReLU()
    (3): Conv2d(in_channels=5, out_channels=10, kernel_size=5, stride=2)
    (4): MaxPool2d(kernel_size=4, stride=4, padding=0)
    (5): ReLU()
  )
  (fc_loc): Sequential(
    (0): Linear(in_dim=90, out_dim=32)
    (1): ReLU()
    (2): Linear(in_dim=32, out_dim=6)
  )
)
```

The following is the network architecture of the second STN block inserted into the encoder of MoDem.

```
STN_Block_1_MoDem(
  (localization): Sequential(
    (0): Conv2d(in_channels=32, out_channels=8, kernel_size=7, stride=2)
    (1): MaxPool2d(kernel_size=3, stride=3, padding=0)
    (2): ReLU()
    (3): Conv2d(in_channels=8, out_channels=10, kernel_size=5, stride=2)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0)
    (5): ReLU()
  )
  (fc_loc): Sequential(
    (0): Linear(in_dim=90, out_dim=32)
    (1): ReLU()
    (2): Linear(in_dim=32, out_dim=6)
  )
)
```

B Environment Details

We categorize the view generalization problem into four distinct settings: novel view, moving view, shaking view, and novel FOV. In this section, we provide descriptions of the implementation details for

https://doi.org/10.52202/075280-0940

Table 10: **Hyperparameters for training time.**

Hyperparameter	Value
Discount factor	0.99
Image size	84×84 (TD-MPC) 224×224 (MoDem)
Frame stack	3 (TD-MPC) 2 (MoDem)
Action repeat	1 (xArm) 2 (Adroit, Finger, and Walker in DMControl) 4 (otherwise)
Data augmentation	± 4 pixel image shifts (TD-MPC) ± 10 pixel image shifts (MoDem)
Seed steps	5000
Replay buffer size	Unlimited
Sampling technique	PER ($\alpha = 0.6$, $\beta = 0.4$)
Planning horizon	5
Latent dimension	50
Learning rate	1e-3 (TD-MPC) 3e-4 (MoDem)
Optimizer (θ)	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$)
Batch size	256
Number of demos	5 (MoDem only)

Table 11: **Hyperparameters for test time training.**

Hyperparameter	Value
Buffer size	256
Batch size	32
Multi-update times	32
Learning rate for encoder	1e-6 (xArm) 1e-7 (otherwise)
Learning rate for STN blocks	1e-5

each setting. The detailed camera settings can be referred to in the code of the environments that we are committed to releasing or in the visualization available on our website yangsizhe.github.io/MoVie.

Novel view. In this setting, for locomotion tasks (cheetah-run, walker-stand, walker-walk, and walker-run), the camera always faces the moving agent, while for other tasks, the camera always faces a fixed point in the environment. Therefore, as we change the camera position, the camera orientation also changes accordingly.

Moving view. Similar to the previous setting, the camera also always faces the moving agent or a fixed point in the environment. The camera position varies continuously.

Shaking view. To simulate camera shake, we applied Gaussian noise to the camera position (XYZ coordinates in meters) at each time step. For DMControl and Adroit, the mean of the distribution is 0, the standard deviation is 0.04, and we constrain the noise within the range of -0.07 to +0.07. For xArm, the mean of the distribution is 0, the standard deviation is 0.4, and we constrain the noise within the range of -0.07 to +0.07.

Novel FOV. We experiment with a larger FOV. For DMControl, we modify the FOV from 45 to 53. For xArm, we modify the FOV from 50 to 60. For Adroit, we modify the FOV from 45 to 50. We also experiment with a smaller FOV and results are presented in Appendix F.

C Visualization of Feature Maps from Shallow to Deep Layers

We incorporate STN in the first two layers of the visual encoder for all tasks. After visualizing the features of different layers (as shown in Figure 7), we found that the features of shallow layers contain

https://doi.org/10.52202/075280-0940

more information about the spatial relationship. Therefore transforming the features of shallow layers for view generalization is reasonable.

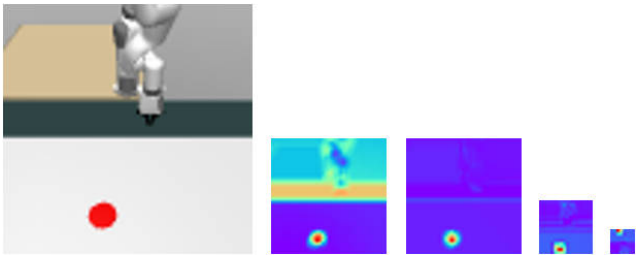

Figure 7: **Visualization of features from shallow to deep layers.** The features of shallow layers contain more information about the spatial relationship.

D Visualization of Feature Map Transformation

We visualize the first layer feature map of the image encoder from TD-MPC and MoVie in Figure 8. It is observed that the feature map from MoVie on the novel view exhibits a closer resemblance to that on the training view.

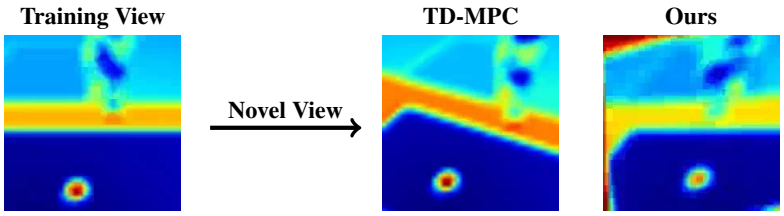

Figure 8: **Visualization of the first layer feature maps of the original encoder on the training view and on the novel view, and the learned SAE on the novel view.**

E Extended Description of Baselines

TD-MPC. We test the agent trained on training view without any adaptation in the view generalization settings.

DM. This is derived from MoVie by removing STN blocks, which just adapts encoder during test time.

IDM+STN. This is derived from MoVie by replacing the dynamics model with the inverse dynamics model which predicts the action in between based on the latent states before and after transition. The inverse dynamics model is finetuned together with the encoder and STN blocks during testing.

F Ablation on Different FOVs

In our main experiments, we consider the novel FOV as a FOV larger than the original. In Table 12, we present results for both smaller and larger FOV scenarios. Our method demonstrates the successful handling of both cases.

Table 12: **Ablation on different FOVs.** The best method on each setting is in **bold**.

Cheetah-run	TD-MPC	DM	IDM+STN	MoVie
Small FOV	104.85±4.59	398.75±17.52	75.92±8.76	**530.37±12.84**
Large FOV	128.55±6.57	379.01±10.90	299.02±88.47	**532.94±19.74**

https://doi.org/10.52202/075280-0940

G Comparison with Other Visual RL Generalization Algorithms

In addition to baselines in the main paper, we also compare MoVie with state-of-the-art methods that focus on visual generalization or data augmentation in visual RL, *i.e.*, DrQ-v2 [35], SVEA [9], PIE-G [36], SGQN [1] and PAD [7]. Results on DMControl environments are shown in Table 13. And the training performance is shown in Table 14. It is observed that MoVie still outperforms these two methods that do not adapt in test time, while PIE-G could also achieve reasonable returns under the view perturbation. This may indicate that fusing the pre-trained networks from PIE-G into MoVie might lead to better results, which we leave as future work. Note that unlike MoVie, other methods need strong data augmentation or modification during training time.

Table 13: **Comparison with other visual RL generalization algorithms.** The best method is in **bold**.

Task	Setting	DrQ-v2	SVEA	SGQN	PIE-G	PAD	MoVie
Cup, catch	Novel view	857.05±27.27	743.50±70.973	776.26±123.63	834.10±23.20	854.51±150.78	**961.98±2.68**
	Moving view	911.20±1.626	843.80±28.17	**971.08±9.30**	809.43±19.12	699.34±24.31	951.26±10.68
	Shaking view	638.51±39.98	501.56±66.76	943.08±7.23	869.86±14.93	870.06±21.72	**951.20±21.01**
	Novel FOV	919.83±12.76	534.36±61.09	803.00±55.15	927.26±15.36	928.87±28.75	**973.60±1.94**
	`All settings`	831.64	655.80	873.35	860.16	838.19	**959.51**
Finger, spin	Novel view	518.75±14.63	312.38±117.76	383.78±6.82	680.10±37.47	233.60±48.93	**892.01±1.20**
	Moving view	706.91±7.69	596.56±156.00	543.23±1.00	828.60±16.47	547.98±3.22	**896.00±21.65**
	Shaking view	39.13±6.37	101.56±67.03	168.60±4.26	**551.83±7.52**	209.30±7.30	284.05±473.73
	Novel FOV	793.70±0.65	505.03±278.99	553.26±1.38	755.86±48.25	82.08±23.4	**917.21±1.71**
	`All settings`	514.77	378.88	412.21	704.09	215.91	**747.31**

Table 14: **The performance of DrQ-v2, SVEA, SGQN, PIE-G, PAD and MoVie under the training view on 2 DMControl tasks.**

Task	DrQ-v2	SVEA	SGQN	PIE-G	PAD	MoVie
Cup, catch	953.61±1.57	971.55±0.06	970.58±3.72	959.98±8.92	974.75±5.33	980.56±4.33
Finger, spin	822.4±1.02	840.33±32.08	603.10±1.17	884.96±4.30	710.26±10.92	985.20±2.25

H Results of Original Models on Training View

The performance of the original agents without any adaptation under the training view is reported in Table 15, 16, and 17 for reference. In the context of view generalization, it is evident that the performance of agents without adaptation significantly deteriorates.

Table 15: **Training Result on DMControl.**

Task	Cheetah, run	Walker, walk	Walker, stand	Walker, run	Cup, catch	Finger, spin
Reward	658.10±9.98	944.99±21.71	983.53±5.34	697.75±11.35	980.56±4.33	985.20±2.25
Task	**Finger, turn-easy**	**Finger, turn-hard**	**Pendulum, swingup**	**Reacher, easy**	**Reacher, hard**	
Reward	756.16±150.76	616.96±149.44	827.26±61.72	983.8±0.34	937.43±54.59	

Table 16: **Training Result on xArm.**

Task	Reach	Push	Peg in box	Hammer
Success rate (%)	96±5	90±17	80±10	83±20

Table 17: **Training Result on Adroit.**

Task	Door	Hammer	Pen
Success rate (%)	96±3	78±7	48±15

https://doi.org/10.52202/075280-0940