On the Role of the Action Space in Robot Manipulation Learning and Sim-to-Real Transfer

Elie Aljalbout*, Felix Frank*, Maximilian Karl, and Patrick van der Smagt[1]

***Abstract*— We study the choice of action space in robot manipulation learning and sim-to-real transfer. We define metrics that assess the performance, and examine the emerging properties in the different action spaces. We train over 250 reinforcement learning (RL) agents in simulated reaching and pushing tasks, using 13 different control spaces. The choice of spaces spans combinations of common action space design characteristics. We evaluate the training performance in simulation and the transfer to a real-world environment. We identify good and bad characteristics of robotic action spaces and make recommendations for future designs. Our findings have important implications for the design of RL algorithms for robot manipulation tasks, and highlight the need for careful consideration of action spaces when training and transferring RL agents for real-world robotics.**

I. Introduction

Robot reinforcement learning (RL) provides a way to acquire manipulation skills without requiring explicit programming of task plans or models of task objects. This property can enable robot manipulation to be feasible in a wide variety of tasks and applications. However, due to the complexity of real-world manipulation tasks, making RL work in these domains can be very challenging. During policy search, the robot acts almost randomly for a large period of time in order for the agent to explore its state and action spaces. This can be a very dangerous and expensive process and should be approached with care. Alternatively, such skills can be safely learned in simulation and transferred later to the real world [1], [2]. In recent years, many approaches have been proposed to overcome the sim-to-real gap [3], [4], [5]. These methods have been shown to enable the successful transfer of manipulation policies learned in simulation. Despite all efforts, the gap could not yet be fully bridged or even understood. This lack of understanding hinders the applicability of sim-to-real transfer to a wide range of applications.

The majority of previous studies focused on the state and observation spaces and perception aspects of robot policy learning and sim-to-real transfer. *Our* goal is to understand the *action* space and *control* aspects of this problem.

To provide more context, in recent years there has been a shift toward embedding well-established control principles into the action space, such as proportional-derivative-controlled joint positions [6] or impedance control in end-effector space [7]. This was in contrast to earlier interests in end-to-end policies, which directly output the lowest level possible of control commands such as joint torques [8]. In this recent trend in robot learning, the policy outputs a higher-level control command, such as desired joint velocities, which are then fed to a low-level hand-engineered controller that handles the low-level control. Such an approach reduces the complexity of the policy, effectively making learning simpler and more efficient. Nevertheless, the introduction of these low-level controllers can fundamentally change the nature of the problem, in some cases even violating some basic assumptions made in the design of RL algorithms, such as the Markov property. In practice, these engineered action spaces could consist of multiple control loops and can abstract some physical phenomena. There exist many options for action spaces that encapsulate robot control algorithms.

In this paper, our aim is to understand the role of the action space in manipulation learning. We are interested in:

(i) the exploration properties of different spaces,
(ii) the emerging properties of policies trained with different low-level controllers,
(iii) the gap created because of the choice of action space,
(iv) the transferability of policies trained in different control spaces to the real world.

We design a large-scale study to quantify these different aspects. We include 13 different action spaces that we evaluate using multiple metrics. measuring success rate transfer, the usability of resulting behaviors, and the gap introduced.

II. Related Work

Action Spaces for Robot Learning. Recent work has shown that the choice of action space can play an important role in the success and performance of manipulation [7], [9], flying [10], and locomotion policies [11], [12]. For manipulation, multiple action spaces for manipulation allow the policy to control its interaction with the environment, either by some direct means of force application [13], [14] or implicitly via impedance control [7], [15], [16]. Several methods explored the use of movement primitives in the action space [17], [18]. Another paradigm involves parameterized skills as discrete actions [19], [20], [21]. The majority of sim-to-real works use configuration space control [22], [4], [23], [24], [25]. The reason behind this is yet unclear. Ganapathi et al. [26] propose an approach to implicitly combine configuration and task-space actions using a differentiable forward kinematics model. Recent approaches propose learning latent action spaces, aiming to reduce control dimensionality to the task manifold [27], [28] or serve a different purpose such as coordinating multi-robot tasks [29].

Action Space Studies. There have been multiple studies concerning the action space in robotics [30], [10], [11]. These

[1]All authors are with the Volkswagen Group, Machine Learning Research Lab, Munich, Germany. *Shared first authorship.

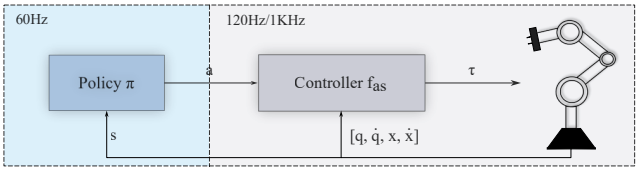

Fig. 1: The policy outputs an action a, which is then transformed into joint torques τ using a select controller f_{as}. The policy and controller receive feedback from the environment. Each action space is defined by the choice of the controller and the way the action is treated in the controller. The policy runs at a 60 Hz frequency and the controller runs at a frequency of 120 Hz and 1 kHz in the simulation and the real world, respectively.

studies span robotic applications including locomotion [11], manipulation [30], and flying robots [10]. However, the majority of these studies focus on a very limited set of action spaces or tasks. Multiple of them study this problem in simulation only [11], [30], [7]. For instance, Martín-Martín et al. [7] compare 4 action spaces for manipulation in simulation only, as part of the evaluation of their proposed action space. In contrast, Kaufmann et al. [10] study the transfer of flying policies from simulation to the real world with three different control spaces. In this work, we study the role of the action space for learning manipulation policies in simulation, and also their transfer to the real world. Our study includes 13 action spaces, spanning combinations of commonly used design characteristics in manipulation.

III. Methods

In robot learning, the choice of action space involves the choice of a controller and the frequency and limits of the policy outputs. The frequency directly defines the reactive capabilities of the policy, and the limits are typically given by the choice of controller and robot. In this paper, we aim to study low-level control policies. We fix the frequency of all action spaces to 60 Hz. We use the limits that are given by the robot for the corresponding control variables.

A. Reinforcement Learning

RL automates policy learning by maximizing the cumulative reward in a given environment [31]. Tasks are usually formulated as Markov Decision Processes (MDP). A finite-horizon, discounted MDP is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition dynamics, and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward. We have an initial state distribution ρ_0, a discount factor $\gamma \in [0, 1]$, and a horizon length T. The optimal policy $\pi : \mathcal{S} \to P(\mathcal{A})$ maximizes the expected discounted reward:

$$J(\pi) = E_\pi \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t) \right]. \tag{1}$$

In this study, we use the proximal policy optimization (PPO) algorithm [32] for learning manipulation policies. This choice is based on the popularity of this algorithm in recent works on sim-to-real transfer [33], [23], [34], [24], [25].

B. Action Spaces

For arm manipulation, one of the most native RL action spaces is one that expects **joint torque (JT)** commands. This corresponds to $\tau = a$, where τ is the vector of joint torques. Such an action space gives the policy full control over the robot. When given the right observations, such a policy can internally learn to control the motion and forces exerted by the robot. This action space was popular in early works using deep RL [8], [35]. However, learning a policy with this action space can be very complicated since the policy would need to either understand or implicitly handle both the kinematics and dynamics of the robot to fulfill the task. This is due to the fact that the reward is typically expressed using task space properties.

Alternatively, a low-level controller $f_{\text{as}} : \mathcal{A} \longmapsto T$ can be integrated in the action space to convert higher-level policy actions into the space of joint torques T of the robot. This concept is illustrated in Figure 1.

Configuration action spaces consist of all action spaces that expect a configuration-space action from the policy. At the center of all these action spaces is the same controller, namely a joint impedance controller (JIC) [36]. JIC regulates the behavior of a robot manipulator's joints and allows specifying desired stiffness, damping, and inertia characteristics for each joint. This allows the robot to be compliant with its environment. The JIC control law is the following:

$$\tau = f_{\text{JIC}}(a) \tag{2}$$
$$= K(q_d - q) + D(\dot{q_d} - \dot{q}), \tag{3}$$

where τ denotes the commanded joint torque, q_d and $\dot{q_d}$ denote the desired joint positions and velocities respectively, and q and $\dot{q}$ denote the actual joint positions and velocities of the robot given as feedback. K and D are the stiffness and damping matrices. Note that we omit the gravity vector from our controller equations for clarity. In practice, we use isotropic gains, i.e., these matrices are diagonal. Given this control law, we can define two different base configuration action spaces:

- **Joint Velocities (JV)**: sets $f_{\text{JV}} \leftarrow f_{\text{JIC}}$, and $\dot{q_d} = s(a)$ and q_d is computed as a first-order integration of $\dot{q_d}$;
- **Joint Position (JP)**: sets $f_{\text{JP}} \leftarrow f_{\text{JIC}}$, and $q_d = s(a)$ and $\dot{q_d}$ is computed as a first-order differentiation of q_d.

$s(a)$ is a function that scales the action vector to the limits of the corresponding output vector.

Task action spaces are defined using variables that are in the task/Cartesian space of the robot. In the absence of accurate system identification, we can use the following Cartesian impedance controller as described in [37]:

$$\tau = J(q)^T \big(K(x_d - x) + D(\dot{x_d} - \dot{x})\big), \tag{4}$$

where $J(q)$ is the Jacobian matrix for the current robot configuration q, relating joint velocities to Cartesian velocities, x_d and $\dot{x_d}$ are the desired Cartesian poses and velocities, x and $\dot{x}$ are the current Cartesian poses and velocities respectively. However, this formulation can be tricky to use when the x_d and $\dot{x_d}$ are generated by an RL policy. This is due to

the non-smooth nature of RL action trajectories. Of course, this problem can be handled by introducing interpolators or cubic spline fitting [7], [38]. But such solutions involve multiple design choices and hyperparameters, meaning that an ideal solution is task-specific. Instead, we transform the Cartesian actions into joint velocities, and then use a joint impedance controller. We found this approach to be very good at handling the non-smooth policy actions, without introducing any additional sim-to-real gap. We have two base task action spaces:

- **Cartesian Velocities (CV)**: sets $\dot{x_d} \leftarrow s(a)$, transforms $\dot{x_d}$ into $\dot{q_d}$ using inverse kinematics (IK), and then uses f_{JIC} in the same fashion as in the JV action space;
- **Cartesian Position (CP)**: sets $x_d \leftarrow s(a)$, and then uses a proportional control to obtain $\dot{x_d}$ from x_d. This step naturally results in smooth $\dot{x_d}$. Given $\dot{x_d}$, this action space proceeds as in the CV action space.

We use the pseudoinverse IK method with a null-space controller that pushes the joints toward their default positions. We represent the orientation actions differently for these two spaces. In CP, we use the 6D representation presented in [39]. In CV, we use the Euler representation.

Delta action spaces are based on the base action spaces defined previously. In contrast to the base action spaces, delta action spaces set the control targets v_d relative to the current system feedback or to the control targets of the previous policy step. This distinction creates two classes of delta action spaces:

- **One-step Integrator (OIΔ)**: uses the robot feedback v to set $v_d \leftarrow v + c \cdot a \cdot dt$;
- **Multi-step Integrator (MIΔ)**: recurrently sets $v_d \leftarrow v_d + c \cdot a \cdot dt$,

where OIΔ is quite common [40], [41] and an action space similar to MIΔ was proposed in [25]. Depending on the choice of base action space, v_d is a control target vector, and can correspond to q_d, $\dot{q_d}$, x_d, or $\dot{x_d}$. Similarly, v is a control feedback vector, and can correspond to q, $\dot{q}$, x, or $\dot{x}$. dt is the step duration and c is a positive constant hyperparameter. Instead of the scaling performed in non-delta base action spaces, we clip the target v_d to the limits of the corresponding output space after updating it. This means that each base (configuration or task) action space has two additional variants, resulting in 12 action spaces, not including the joint torques action space. Despite relying on the base action spaces, the delta variants have their unique properties. For instance, due to the relative changes in the control targets, the magnitude of the target change in one step is bound by $c \cdot dt$ given that $a \in [-1, 1]$. This property can be helpful in imposing smoothness constraints on control target trajectories, even if policy output is unconstrained. If we set c to be the positive bound of the derivative of the corresponding control feedback variable v, each delta action space would approximately be equivalent to an action space of a higher-order derivative than its base action space. For instance, a delta joint velocity action space would be approximately similar to a joint acceleration action space. However, when sampling actions a from a uniform distribution, the resulting distribution of control targets can differ between a delta action space and the base action space it approximates (e.g. ΔJP and JV). The clipping can also lead to more probability mass on the borders of the v_d space.

C. Metrics

For each action space, we aim to assess its training performance, resulting sample efficiency, emerging properties such as the usability in the real-world environment, and the sim-to-real gap it creates. Therefore, we propose multiple metrics to quantify these different aspects. For assessing training performance, we look at the **episodic rewards (ER)** in simulation. This metric can also show us the sample efficiency of the different action spaces.

To assess the emerging properties of each policy, we look at the number of times it violates robot constraints, such as the joint acceleration and jerk constraints. This is especially important since simulated environments rarely have any mechanisms for enforcing these constraints. In contrast, real-world robot control implementations can include rate limiters, which ensure that the control targets would not result in violations of these constraints. A policy trained in simulation without such mechanisms could learn to violate them for the sake of exploration. Implementing these mechanisms in simulation, on the other hand, can also hinder the policy training since these mechanisms typically break the Markov assumption. To quantify this property, we use the **expected constraints violations (ECV)** defined as

$$\text{ECV}(\pi) = E_{s,a\sim\pi}\left[\mathbb{1}\left(\sum_{c\in\mathcal{C}} c(s,a) > 0\right)\right], \tag{5}$$

where $\mathbb{1}$ is an indicator function, $\mathcal{C}$ is the set of all constraints, and each constraint function $c(s,a)$ returns 1 for violated constrained and 0 otherwise. Furthermore, we report the **normalized tracking error (NTE)** as a measure of the feasibility of the policy's actions in the environment,

$$\text{NTE}(\pi) = E_{s,a\sim\pi}\frac{|v_{d,t} - v_{t+1}|}{v_{\max} - v_{\min}}, \tag{6}$$

where $v_{\min}$ and $v_{\max}$ are respectively the lower and upper bounds of the control variable v. NTE is useful for analysing why certain action spaces result in better transfer. A high value means that the policy outputs actions that are hard to achieve in one control step. This could create an additional sim-to-real gap since control targets that are not fulfilled in one step can be tracked differently in simulated and real environments due to the gap in dynamics. We also report the **task accuracy (ACC)** measured by the Euclidean distance to the goal. This metric gives a more detailed view of the performance of a policy than just the success rate.

To assess the sim-to-real gap of each action space we report the **offline trajectory error (OTE)** in configuration and task spaces. This metric measures the joint trajectory error when replaying, in simulation, the actions produced by the policy when queried in the real world,

$$\text{OTE}(\pi) = E_{a,q_{\text{real}}\sim\mathcal{D}_{\text{real}}}|q_{\text{sim}} - q_{\text{real}}|, \tag{7}$$

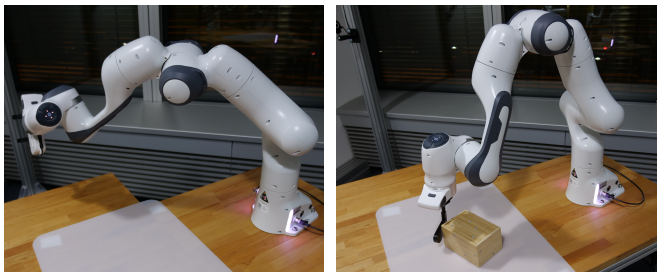

Fig. 2: We show our real-world robot setup for the reaching (left) and the pushing (right) tasks.

where a and q_{real} are actions and joint configurations that are sampled from the dataset $\mathcal{D}_{\text{real}}$. The latter is collected by playing a policy π in the real-world, and q_{sim} is the joint configuration obtained when executing a in simulation in an open-loop fashion, i.e. without a policy.

IV. Experiments

We design experiments to understand the role of different characteristics of the action space on learning efficiency, sim-to-real gap, emerging properties, and sim-to-real transfer. Throughout this section, we pose different questions related to these topics, and answer them to the best of our capability based on empirical analysis.

A. Experimental Setup

To answer these questions we evaluate all action spaces on two arm manipulation tasks, using the 7-degree-of-freedom Franka Emika Panda robot. The first task is goal reaching. At the beginning of each episode, a Cartesian goal is sampled in the workspace of the robot, and the policy needs to move the end-effector towards that goal. This task is ideal for studying the behavior of policies from all action spaces in the lack of force interactions with the robot's environment. The second task is object pushing. At the beginning of each episode, a goal position is sampled in a predefined area on the table. The policy needs to push a wooden box towards that goal. This task involves moving an external object. Unlike the reaching task, pushing requires physical interaction with the environment. It allows us to understand the reactive capabilities of each action space and whether it creates any sim-to-real gap that hinders the transfer of the interaction behavior. During policy training for pushing in simulation, we perform domain randomization on the box's friction and mass parameters. For each task, we train 5 different policies for each action space variant, using random seeds. We use a fixed number of training epochs for all policies and we evaluate the best 3 policies on the real-world setup. The real-world setup for both tasks is shown in Figure 2. In both tasks, the observation space of the policy consists of joint positions, joint velocities, end-effector Cartesian position, and the goal position. Additionally, in the pushing task the policy has access to the object's position and orientation. We train PPO policies in a simulated environment, using NVIDIA's Isaac-sim simulator and the PhysX engine. We leverage the robot model developed for the cabinet opening task in [42], and scan the woodbox object used in the real-world environment to use the meshes in our simulation. Self-collisions are not enabled during training.

The results discussed in this paper are based on 250 agents with different action spaces and different tasks. However, prior to the final training runs (of the 250 agents), we optimized hyperparameters, such as learning rates or different reward scales and terms, to guarantee consistency and a fair comparison over all action spaces. We matched the simulation to the real environment to the best of our knowledge, eliminating sim-to-real gaps where possible. This includes, for example, matching the stiffness of the impedance controllers, sharing implementations for each action space across both environments, and finding good velocity limits that allow for safe transfer of policies. This tuning process required training more than 20,000 agents in simulation and testing more than a 1500 agents in the real world. After training, we evaluate the learned policies in both tasks in simulation and in the real world. For reaching, we use a fixed grid of target goals that span the feasible workspace of the robot. For pushing, we randomly sample goal positions and use the object position from the previous episode as the initial one from the new run. We only manually reset the object's position whenever it gets outside of the predefined area it was trained to operate in or when the policy fails to push the object away from its starting location once. We introduced the latter intervention to avoid heavily skewing the data by randomly sampling a difficult-to-handle initial object position. In both training and evaluation, we run the simulation at 120 Hz and we use action repeat to work with the policy output at 60 Hz. We originally experimented with higher frequency joint torques policies, but found for our setup no benefit in going beyond 60 Hz. On the real robot, we need an additional low-level safety controller running at 1 kHz, which also repeats the actions from the policy. In all action spaces except torque we additionally use a 5 Hz low-pass filter and a rate limiter to conform to motion limits given by the robot. We tried implementing the rate limiter in simulation as well, however, the policy learning deteriorated massively, which we attribute to the rate limiter breaking the Markov property. For the reaching policy, we use a 4-layer feed-forward neural network with (512, 256, 128, 64) units for all action spaces. For pushing, we use 5-layer networks with (1024, 1024, 512, 256, 128) units. Furthermore, for both tasks, torque policies use 2 additional layers with 1024 neurons.

B. Results

We study different properties of action space selection in robotic manipulation and sim-to-real transfer.

Does the choice of action space affect the exploration behavior during training in simulation?

We first examine the training performance in simulation. The results can be seen in the Figure 3. The episodic rewards are not directly representative of the success rate of the policies. This can be seen when comparing the rewards from Figure 3 to the success rates shown in Table I. Therefore, we mostly focus on sample efficiency in the current analysis. Since

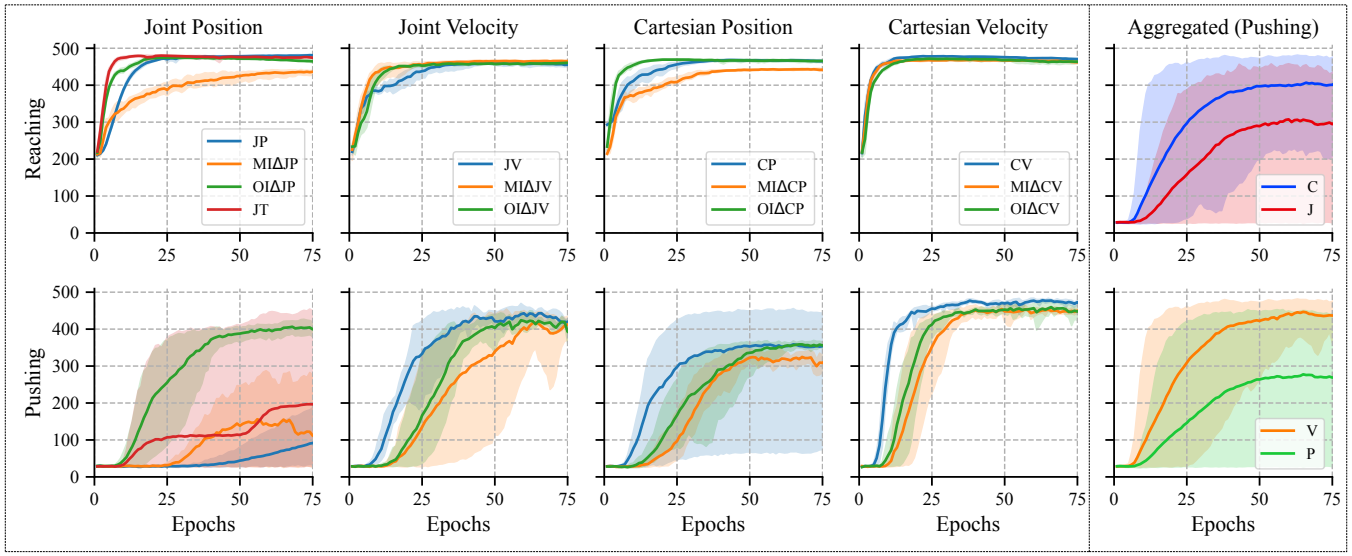

Fig. 3: We show the episodic reward (ER) obtained during training in simulation for all the studied action spaces. The first four columns show the learning curves grouped by action spaces, whereas the last column shows aggregated results comparing joint (J) and Cartesian (C) action spaces as well as position based (P) and velocity based (V) action spaces for the pushing task. Delta (Δ) action spaces are labeled as either one-step (OI) or multi-step (MI) target integration. We added joint torque (JT) control to the first column. The shaded region represents the range between the 5th and 95th percentiles.

different action spaces have different characteristics, we aim to understand the global effect of these characteristics on the sample efficiency during training. First, we compare Cartesian and joint action spaces. In the reaching task, both action space groups behave almost identically. However, in the pushing task, Cartesian action spaces seem to have an advantage in terms of sample efficiency and maximum reached reward, as can be seen in the top right plot in Figure 3. This result is most likely due to the spatial nature of the pushing task, which gives Cartesian action spaces a natural advantage in exploration.

We next compare the order of derivative of the action space, i.e. positions vs. velocities vs. torques. Among joint action spaces, joint velocity and its derivatives have the overall best performance in both tasks. These action spaces show better sample efficiency and converge to higher reward regions than their counterparts in the joint action space group as can be seen in the bottom right plot in Figure 3. JP reaches the highest reward in reaching but struggles massively in pushing. The same tendency can be observed for Cartesian action spaces, i.e. velocity action spaces perform better in terms of sample efficiency and final rewards. The joint torque action space is the fastest one to converge in the reaching task, however, it fails to solve the pushing task reliably within the same data budget. Training pushing policies in this action space for a longer time does converge, but is very expensive.

We also compare base action spaces with the two different kinds of delta action spaces. We observe that multi-step integration delta action spaces consistently perform the worst, while one-step methods seem to have a slight advantage. This tendency is less pronounced in the velocity action spaces, with MIΔJV being consistently one of the best-performing joint action spaces in both tasks in simulation. Hence, the current simulation data does not conclusively favor any of these characteristics (non-delta, OI and MI). Overall, we note that Cartesian velocity (CV) is the best-performing action space in simulation. The worst one is multi-step-integration joint position action space.

What properties naturally emerge due to the choice of action space?

After training the policies in simulation we evaluate them on a real robotic setup. For the pushing task, we were not able to obtain joint torque policies that are safe to run in the real world without damaging the robot. We made multiple attempts to produce safely deployable joint torque policies, for instance, by introducing different penalties or increasing the policy's control frequency. Despite all efforts, all joint torque policies were very jerky or aggressive when deployed on the real robot. The main difficulty was to obtain policies that are safe to deploy in a task where the end effector needs to remain in close proximity to a rigid surface, e.g. the table. Therefore, we exclude the joint toque action space from our real-world pushing experiments. For all other action spaces, we look at their sim-to-real transfer capabilities. The reaching task was less safety-critical. Hence, we managed to evaluate torque policies on real-world reaching. In Table I, we report the metrics introduced in Section III-C to quantify the sim-to-real gap and the performance in the real world. One common challenge when learning manipulation skills is to obtain smooth policies that do not violate the velocity, acceleration, and jerk constraints of the robot. Based on the results in Table I, we observe that different action spaces yield different ECV metrics. JV and its derivatives (OIΔJV and MIΔJV) have on average the lowest ECV score in both tasks. Nominal JP results in the highest possible ECV. This

TABLE I: Sim-to-real transfer evaluation. We evaluate the success rate (SR) in simulation and the real-world environment, the task accuracy (ACC), the expected constraints violations (ECV), and the offline trajectory error (OTE).

Action space	Reaching					Pushing			
	SR (sim) ↑	**SR** (real) ↑	**ACC** [cm] ↓	**ECV** ↓	**OTE** [rad] ↓	**SR** (sim) ↑	**SR** (real) ↑	**ACC** [cm] ↓	**ECV** ↓
JP	**100 ± 0**	6 ± 6	1.76 ± 0.14	100 ± 0	0.22 ± 0.01	87 ± 12	4 ± 0	4.23 ± 0.66	100 ± 0
OIΔJP	**100 ± 0**	86 ± 9	1.59 ± 0.16	14 ± 1	0.45 ± 0.01	99 ± 3	48 ± 12	3.16 ± 0.46	98 ± 2
MIΔJP	**100 ± 0**	69 ± 2	2.14 ± 0.13	**0 ± 0**	**0.03 ± 0.00**	90 ± 7	36 ± 3	3.52 ± 0.36	**3 ± 1**
JV	**100 ± 0**	**100 ± 0**	**1.17 ± 0.11**	**0 ± 0**	**0.03 ± 0.00**	97 ± 5	90 ± 5	2.04 ± 0.32	66 ± 9
OIΔJV	**100 ± 0**	97 ± 5	1.35 ± 0.13	13 ± 11	0.18 ± 0.01	99 ± 3	88 ± 13	**1.56 ± 0.13**	75 ± 4
MIΔJV	**100 ± 0**	93 ± 9	1.98 ± 0.08	37 ± 37	**0.03 ± 0.00**	**100 ± 0**	**96 ± 4**	1.66 ± 0.11	55 ± 25
JT	**100 ± 0**	64 ± 27	1.97 ± 0.25	22 ± 6	0.52 ± 0.03	40 ± 49	- ± -	- ± -	89 ± 21
CP	99 ± 2	25 ± 4	1.71 ± 0.29	26 ± 9	0.16 ± 0.01	**100 ± 0**	57 ± 9	2.34 ± 0.39	92 ± 3
OIΔCP	89 ± 2	44 ± 17	2.02 ± 0.32	21 ± 1	0.27 ± 0.01	99 ± 3	55 ± 17	2.86 ± 0.49	35 ± 16
MIΔCP	**100 ± 0**	68 ± 2	1.85 ± 0.08	13 ± 4	0.28 ± 0.01	**100 ± 0**	73 ± 7	2.36 ± 0.14	99 ± 0
CV	**100 ± 0**	43 ± 13	1.76 ± 0.41	12 ± 2	0.35 ± 0.02	99 ± 3	73 ± 9	2.69 ± 0.58	93 ± 10
OIΔCV	97 ± 2	24 ± 2	2.13 ± 0.13	17 ± 3	0.40 ± 0.01	99 ± 3	61 ± 19	2.67 ± 0.30	99 ± 0
MIΔCV	99 ± 2	58 ± 8	1.82 ± 0.08	4 ± 0	0.13 ± 0.01	86 ± 5	74 ± 4	2.84 ± 0.45	6 ± 2

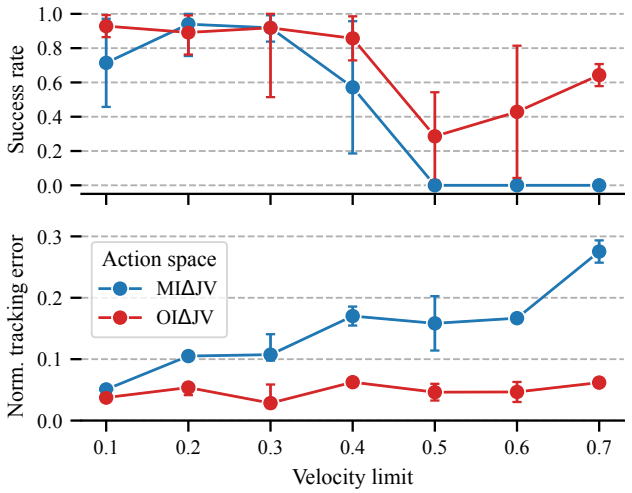

Fig. 4: Robustness of delta action spaces to the velocity limit hyperparameter in the pushing task on our real setup. We compare one-step and multi-step integration. The error bars indicate the 5th and 95th percentiles.

means that deploying this action space in the real world would always result in some form of constraint violation. In the absence of safety mechanisms (such as rate limiters and low-pass filters) these violations can be very harmful. If such mechanisms are implemented, this behavior would increase the sim-to-real gap further because of how these violations trigger the safety mechanisms. In contrast, MIΔJP, seems to have the lowest ECV, which was also evident when evaluating this action space in the real world.

Does the action space affect the sim-to-real gap?
First, we look at the offline trajectory error in the reaching task, which gives us a proxy measure of the sim-to-real gap. We observe that the OTE varies tremendously from one action space to another, despite the fact that the data used to compute this measure are based on the same starting and goal positions in all action spaces. This confirms that *the choice of action space does indeed contribute to the sim-to-real gap*. Looking closer at the results, we observe that JT exhibits the highest OTE. This is due to the behavior of this action space being dictated solely by the dynamics of the robot, which is different in simulation and in the real world. Unlike all the other action spaces, JT does not include any feedback loops outside of the policy. Based on this result, one would expect that more feedback loops should help reduce the sim-to-real gap. However, the opposite can be seen in the data. For instance, Cartesian action spaces have on average one additional feedback loop compared to the joint action spaces. Their OTE is, however, higher on average. This is due to the fact that feedback loops have different effects in simulation than in the real world. In turn, this means that a good, highly-reactive feedback loop is beneficial to overcome the dynamics gap, but adding more could potentially contribute further to the sim-to-real gap. Comparing OIΔ and MIΔ action spaces we notice that the latter consistently have a smaller OTE. Their OTE is even smaller than the corresponding base action spaces. This effect is potentially due to the integral term embedded in these action spaces. Executing the same actions results in the same final goal given to the lower-level feedback loops, which is unique compared to all other action spaces.

Which action space characteristics are good for sim-to-real transfer?
We compare the success rates in simulation and the real world. We observe that *the success rate in simulation is not directly reflected in the real world*. This is clear when comparing the ordering of success rates in both domains. Furthermore, we notice that certain action space characteristics are clearly advantageous for sim-to-real transfer. For instance, velocity-based action spaces tend to keep a high success rate when transferred to the real world. In contrast, position-based action spaces do not typically transfer well. This is especially the case for JP, which loses almost all of its performance when transferred. These results are consistent across both tasks. Velocity-based action spaces are on average more accurate in fulfilling the task as can be seen when comparing the ACC score in Table I.

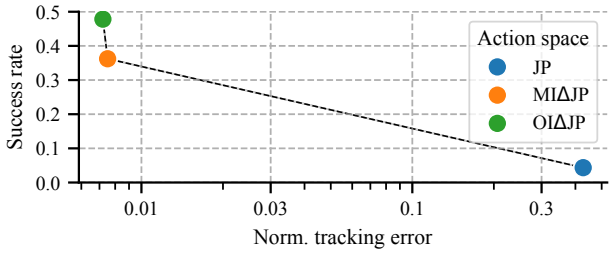

Fig. 5: We compare joint position-based action spaces in the pushing task and show their influence on the normalized tracking error and the resulting effect on success rate in the real robot setup.

Additionally, we can see that delta action spaces transfer better than their corresponding base spaces. The difference between OI and MI delta action spaces depends on the task and the variants of the action space. However, OIΔ action spaces required less tuning than MIΔ spaces and have shown to be more robust to the choice of hyperparameters. This can be seen in Figure 4. When varying the velocity limits for ΔJV action spaces, OIΔJV showed to be less sensitive to the chosen value. We attribute this to the lower tracking error of this action space as shown in the bottom plot in Figure 4. OIΔ action spaces naturally lead to a lower tracking error since they integrate the policy actions into control targets based on the current feedback of the system. In contrast, MIΔ action spaces integrate the policy actions into control targets based on the previous control target. This in turn leads to a bigger gap between the control target and the state of the system, and hence a larger tracking error. While a lower NTE helps make JV action spaces more robust to hyperparameters, it has an even larger effect on the best possible transfer performance in JP action spaces, as shown in Figure 5. This tendency is also evident when comparing the base spaces, as shown in Figure 6. Finally, joint velocity show better transfer than Cartesian velocity action spaces, while the opposite is true for position-based action spaces.

In summary, our data shows that two characteristics mostly influence the transfer capability of an action space. The first one is the order of the derivative of the control variables. *An action space that controls a higher order derivative transfers better.* A clear exception to this finding is the torque action space, which suffers from the highest sim-to-real gap (as shown by the OTE in Table I) due to its direct reliance on the mismatched environment dynamics and lack of external feedback loops in comparison to the other action spaces. The second characteristic is the emerging tracking error of the action space's control variable(s). Our data strongly indicates that *an action space which yields or naturally limits the tracking error transfers better.* This last property can be enforced by the action space design, for instance, by reducing the magnitude of jumps of the corresponding control targets. The latter can be controlled by the scaling of the actions in delta action space, or by increasing the stiffness if the task allows. For action spaces that do not naturally allow for controlling this property, one could attempt to enforce a smaller NTE by means of rewards/penalties on the actions' magnitude and smoothness. However, based on our experience, the tuning process for the resulting additional hyperparameters (of such reward terms) can be very difficult.

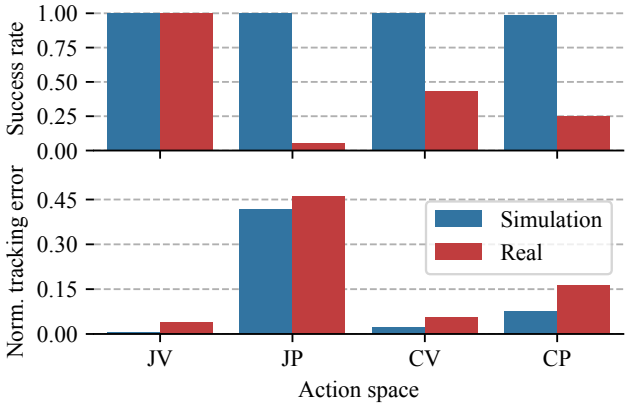

Fig. 6: We show the influence of the normalized tracking error on sim-to-real transfer in terms of success rate for non-delta action spaces in the reaching task.

Is there a consistently best-performing action space? Overall, we notice that *joint velocity action spaces seem to have the best performance in sim-to-real transfer*. These action spaces have on average the lowest OTE, ACC and ECV and transfer the best to the real world. They also required the least tuning to work, making them the most suited action spaces for manipulation learning (among the studied options). This result is consistent with our previous findings concerning favorable characteristics, i.e. JV-based action space benefits from a higher-quality feedback loop due to their configuration-space control, and can more easily generate high forces for interaction than position-based action space due to controlling a higher-order derivative.

V. Conclusion

We studied the role of the action space in learning robotic arm manipulation skills. We designed a study that includes 13 action spaces that we used for training RL policies in simulation. We looked at the exploration behavior under the different action spaces and observed that Cartesian spaces as well as ones that control a higher-order derivative have an exploration advantage. Furthermore, we observe that different action spaces lead to different emergent properties such as the safety of their transfer and their rate of constraints violations. However, we notice that the success rate of policies in simulation does not dictate their performance in the real world. Our study further shows that the characteristics that mostly affect the sim-to-real transfer are the order of derivative and the emerging tracking error of the action space. Joint velocity-based action spaces show very favorable properties, and overall the best transfer performance. Our results show that the choice of action space plays a central role in learning manipulation policies and the transfer of such policies to the real world. To validate the extent of our findings and recommendations to different tasks, observation spaces, robots, and hardware setups, more studies of this sort are required in the future.

ACKNOWLEDGMENT

The authors would like to thank Alexandros Paraschos and Philip Becker-Ehmck for their help and support.

REFERENCES

[1] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*, IEEE, 2020.

[2] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers in Robotics and AI*, 2022.

[3] E. Heiden, D. Millard, E. Coumans, and G. S. Sukhatme, "Augmenting differentiable simulators with neural networks to close the sim2real gap," *ArXiv*, vol. abs/2007.06045, 2020.

[4] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, 2019.

[5] J. Borrego, P. de Carvalho Borrego, P. M. López, J. A. Rosado, and L. M. M. Custódio, "Mind the gap! bridging the reality gap in visual perception and robotic grasping with domain randomisation," 2018.

[6] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," *ArXiv*, vol. abs/2109.11978, 2021.

[7] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019.

[8] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: Policy learning with deep dynamical models," *arXiv preprint arXiv:1502.02251*, 2015.

[9] M. Alles and E. Aljalbout, "Learning to centralize dual-arm assembly," *Frontiers in Robotics and AI*, vol. 9, 2022.

[10] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022.

[11] X. B. Peng and M. Van De Panne, "Learning locomotion skills using deeprl: Does the choice of action space matter?," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2017.

[12] H. Duan, J. Dao, K. Green, T. Apgar, A. Fern, and J. Hurst, "Learning task space actions for bipedal locomotion," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021.

[13] C. C. Beltran-Hernandez, D. Petit, I. G. Ramirez-Alpizar, T. Nishi, S. Kikuchi, T. Matsubara, and K. Harada, "Learning force control for contact-rich manipulation tasks with rigid position-controlled robots," *IEEE Robotics and Automation Letters*, 2020.

[14] M. Ulmer, E. Aljalbout, S. Schwarz, and S. Haddadin, "Learning robotic manipulation skills using an adaptive force-impedance action space," *arXiv preprint arXiv:2110.09904*, 2021.

[15] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel, "Reinforcement learning on variable impedance controller for high-precision robotic assembly," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019.

[16] M. Bogdanovic, M. Khadiv, and L. Righetti, "Learning variable impedance control for contact sensitive tasks," *IEEE Robotics and Automation Letters*, vol. 5, 2020.

[17] S. Bahl, M. Mukadam, A. Gupta, and D. Pathak, "Neural dynamic policies for end-to-end sensorimotor learning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[18] E. Aljalbout, J. Chen, K. Ritt, M. Ulmer, and S. Haddadin, "Learning vision-based reactive policies for obstacle avoidance," in *Conference on Robot Learning*, PMLR, 2021.

[19] M. Sharma, J. Liang, J. Zhao, A. LaGrassa, and O. Kroemer, "Learning to compose hierarchical object-centric controllers for robotic manipulation," *arXiv preprint arXiv:2011.04627*, 2020.

[20] R. Chitnis, S. Tulsiani, S. Gupta, and A. Gupta, "Efficient bimanual manipulation using learned task schemas," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020.

[21] M. Dalal, D. Pathak, and R. R. Salakhutdinov, "Accelerating robotic reinforcement learning via parameterized action primitives," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[22] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Conference on Robot Learning*, PMLR, 2020.

[23] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[24] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, *et al.*, "Dextreme: Transfer of agile in-hand manipulation from simulation to reality," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023.

[25] B. Tang, M. A. Lin, I. Akinola, A. Handa, G. S. Sukhatme, F. Ramos, D. Fox, and Y. Narang, "Industreal: Transferring contact-rich assembly tasks from simulation to reality," *arXiv preprint arXiv:2305.17110*, 2023.

[26] A. Ganapathi, P. Florence, J. Varley, K. Burns, K. Goldberg, and A. Zeng, "Implicit kinematic policies: Unifying joint and cartesian action spaces in end-to-end robot learning," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022.

[27] W. Zhou, S. Bajracharya, and D. Held, "Plas: Latent action space for offline reinforcement learning," in *Conference on Robot Learning*, 2020.

[28] A. Allshire, R. Martín-Martín, C. Lin, S. Manuel, S. Savarese, and A. Garg, "Laser: Learning a latent action space for efficient reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021.

[29] E. Aljalbout, M. Karl, and P. van der Smagt, "Clas: Coordinating multi-robot manipulation with central latent action spaces," in *Learning for Dynamics and Control Conference*, PMLR, 2023.

[30] P. Varin, L. Grossman, and S. Kuindersma, "A comparison of action spaces for learning manipulation tasks," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019.

[31] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

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

[33] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *Robotics: Science and Systems*, 2018.

[34] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019.

[35] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[36] N. HOGAN, "Impedance control: an approach to manipulation. ii: Implementation," *Journal of dynamic systems, measurement, and control*, 1985.

[37] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, Feb. 1987.

[38] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.

[39] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[40] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," 2023.

[41] M. A. Lee, Y. Zhu, P. Zachares, M. Tan, K. Srinivasan, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making Sense of Vision and Touch: Learning Multimodal Representations for Contact-Rich Tasks," *IEEE Transactions on Robotics*, June 2020.

[42] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.