IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 18 February 2024, accepted 9 March 2024, date of publication 13 March 2024, date of current version 21 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3376739

RESEARCH ARTICLE

Learning to Generate All Feasible Actions

MIRCO THEILE[1,2], (Graduate Student Member, IEEE), DANIELE BERNARDINI[1,3], (Member, IEEE), RAPHAEL TRUMPP[1], (Graduate Student Member, IEEE), CRISTINA PIAZZA[3], (Senior Member, IEEE), MARCO CACCAMO[1], (Fellow, IEEE), AND ALBERTO L. SANGIOVANNI-VINCENTELLI[2], (Life Fellow, IEEE)

[1]TUM School of Engineering and Design, Technical University of Munich, 80333 Munich, Germany
[2]Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, USA
[3]TUM School of Computation, Information and Technology, Technical University of Munich, 80333 Munich, Germany

Corresponding author: Mirco Theile (mirco.theile@tum.de)

The work of Marco Caccamo was supported by the Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

ABSTRACT Modern cyber-physical systems are becoming increasingly complex to model, thus motivating data-driven techniques such as reinforcement learning (RL) to find appropriate control agents. However, most systems are subject to hard constraints such as safety or operational bounds. Typically, to learn to satisfy these constraints, the agent must violate them systematically, which is computationally prohibitive in most systems. Recent efforts aim to utilize feasibility models that assess whether a proposed action is feasible to avoid applying the agent's infeasible action proposals to the system. However, these efforts focus on guaranteeing constraint satisfaction rather than the agent's learning efficiency. To improve the learning process, we introduce *action mapping*, a novel approach that divides the learning process into two steps: first learn feasibility and subsequently, the objective by mapping actions into the sets of feasible actions. This paper focuses on the feasibility part by *learning to generate all feasible actions* through self-supervised querying of the feasibility model. We train the agent by formulating the problem as a distribution matching problem and deriving gradient estimators for different divergences. Through an illustrative example, a robotic path planning scenario, and a robotic grasping simulation, we demonstrate the agent's proficiency in generating actions across disconnected feasible action sets. By addressing the feasibility step, this paper makes it possible to focus future work on the objective part of action mapping, paving the way for an RL framework that is both safe and efficient.

INDEX TERMS Action mapping, feasibility, generative neural network, self-supervised learning.

I. INTRODUCTION

Cyber-physical systems are becoming increasingly complex, with applications ranging from autonomous vehicles in chaotic urban environments to robotic assistants for support in everyday tasks. Most of these applications require the development of complex control systems. Traditionally, these systems were modeled in detail, and control strategies were derived using model-based techniques. However, the increasing complexity of these systems limits the applicability of model-based techniques, thus making data-driven techniques appealing. While data-driven techniques such as reinforcement learning (RL) improved significantly in recent years, they still lack guarantees that they meet all system constraints, i.e., only providing *feasible* control commands.

A popular idea is deriving only the feasibility-relevant part of the system to ensure feasibility while using learning techniques to optimize the underlying objective. The feasibility model only delineates whether a suggested control command in a given situation is feasible, i.e., the control command does not violate any constraints and does not lead to a state from which a future constraint violation is inevitable. Given this feasibility model, the subsequent challenge is integrating it within a learning framework in which a policy aims to optimize an objective function

The associate editor coordinating the review of this manuscript and approving it for publication was Shafiqul Islam.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

subject to feasibility constraints. The commonly applied techniques are *action rejection*, *resampling*, and *action projection*.

Action rejection is a traditional approach, e.g., applied in the Simplex architecture [1], which can be summarized as follows. If the policy's proposed action is feasible according to the feasibility model, it is applied to the system. Otherwise, a backup policy is used, which generates a feasible action, usually independent of the objective. While this is the simplest method to implement, and the timing requirements are predictable, the drawback is that the policy needs to learn the feasibility model explicitly to avoid its action being rejected and replaced with the sub-optimal backup action.

As a straightforward augmentation of the action rejection scheme, action resampling can be applied when training a stochastic policy. Instead of directly switching to the safe action, if the proposed action is infeasible, the policy can be resampled, and the newly generated action can be tested [2]. This process can be repeated until either a feasible action is proposed or a timeout is reached, at which point the safe action of the feasibility controller is applied to the system. While this method may decrease the rejection rate of the policy's actions, it adds computational costs. Additionally, most learning methods train agents that output a reparameterization of a single Gaussian. Resampling from this Gaussian may not offer a feasible action if it is too narrow or poorly aligned with the set of feasible actions. Moreover, the learning agent must still explicitly learn to avoid proposing infeasible actions.

A more nuanced method is action projection [3], which replaces a proposed infeasible action with a feasible action closest to the proposed action. This projection is typically formulated as an optimization problem that must be solved online. The supposed advantage of this method over action rejection is that the replacement action is better than the safe action, which was derived independently of the objective. However, only because the projected action is *close in the action space* does not mean it is also *close in performance*. Additionally, the online optimization requirement may not be computationally feasible, especially for complex systems. From a learning perspective, the projection can either be penalized or ignored. If penalized, the agent again needs to learn explicitly to avoid infeasible actions, but it could receive guidance from the projection distance. If the agent does not penalize infeasible actions, the agent is not required to learn the feasibility model. However, the projection to the closest feasible action will map all infeasible actions to the borders of the feasible action sets. Learning algorithms that require action densities or policy gradients must be adapted to handle the resulting high action density on the borders.

In all three approaches, the learning agent that aims to find an optimum of the objective subject to the feasibility constraints is not aided by the feasibility model; it is solely made safe. The agent must still violate the constraints systematically during interactions with the environment, albeit without actually applying infeasible actions to the system, to learn to satisfy them in the future. We introduce a different approach that allows the learning agent to benefit explicitly from the model-based feasibility model. We call the approach *action mapping*. The idea is to learn the feasibility and the objective consecutively. First, a *feasibility policy* is trained to generate all feasible actions for a given state. Using this feasibility policy, an *objective policy* can learn to choose the optimal action from the feasible ones, given an objective. Note that the optimization problem in the feasible actions could be solved with various methods, including, but not limited to, learning, which can all benefit from the guarantee of constraint satisfaction.

This methodology promises multiple potential advantages. First, the feasibility policy can be trained directly on the feasibility model, requiring no interactions with the environment. Afterward, the objective policy learns to choose among feasible actions, which could significantly reduce the number of interactions with the environment. The combined agent, i.e., feasibility plus objective policy, still needs to exhaustively violate constraints. However, it can learn constraint satisfaction offline from the feasibility model without interactions with the environment. Second, the feasibility policy can be reused if multiple objectives are subject to the same constraints. Third, any knowledge of the environment that can be extracted from the feasibility model can potentially be utilized in the objective policy through parameter sharing between both policies. Lastly, once deployed, it requires precisely one pass of the feasibility policy and the objective policy per step if the feasibility policy has no support in the infeasible action space.

Given these potential advantages, the pivotal question is: How do we train the feasibility policy? This paper endeavors to answer this very question. To this end, we derive the objective of the feasibility policy as a distribution matching problem in which the target is a uniform distribution over the feasible action space. The uniform distribution is chosen since the feasibility policy is agnostic to the objective and should thus not be biased toward specific actions. We further present a methodology for estimating the gradient of different divergence measures to train a feasibility policy toward the target distribution. To evaluate our proposed methodology, we perform three experiments. The first is an illustrative example with an analytical and highly parallelizable feasibility function that shows the input and output of the feasibility policy. The second example illustrates how the feasibility policy can learn to generate feasible trajectory segments for robotic path planning problems, providing a closer tie to reinforcement learning. The third experiment showcases a simple robotic grasping example where feasibility is defined as grasping poses that lead to a successful grasp. This experiment shows how a feasibility policy can be learned for systems without a feasibility model that can be efficiently parallelized.

The contributions of this work are the following:

- Conceptualization of *action mapping* as a framework for safe and efficient reinforcement learning;
- Formulation of a distribution matching problem to train the feasibility policy towards generating all feasible actions;
- Derivation of gradient estimators for different divergence measures utilizing kernel density estimates, resampling, and importance sampling;
- Evaluation of the proposed approach in an illustrative 2D example, a qualitative example for spline-based path planning, and a quantitative planar robotic grasping example.

The remainder of this paper is structured as follows. Section II discusses related work. Section III describes the action mapping motivation and the formulation as a distribution matching problem, followed by the gradient estimation in Section IV. Section V provides an illustrative example to visualize the feasibility policy and Section VI provides an additional example that showcases how action mapping could be used in robotic path planning problems. Sections VII and VIII introduce and discuss the robotic grasping experiments.

II. RELATED WORK

In discrete action spaces, the equivalent of action mapping is action masking, for which the feasibility of each action is evaluated, and the agent chooses the best action among the feasible ones. In [4], the action masking concept is termed *shielding*, in which the shield is based on linear temporal logic. The authors in [5] investigate the consequences of action masking for policy gradient deep reinforcement learning (DRL) algorithms. Applications in various domains show significant performance improvements, e.g., in autonomous driving [6], unmanned aerial vehicle (UAV) path planning [7], and vehicle routing [8].

For continuous action spaces, a straightforward masking approach is not yet available. As discussed before, the approaches can be grouped into *action rejection*, *resampling* [9], and *action projection* [10], [11], [12]. The safety model can be based on control barrier functions [13], Lyapunov functions [14], or variants thereof. Cheng et al. [3] use action projection and train a second model on the previous interventions to reduce the need for future interventions. Zhong et al. [15] derive a *safe-visor* that rejects infeasible actions proposed by the agent and replaces it with a safe action.

The distribution matching problem is similar to posterior sampling, a long-standing problem in statistics. State-of-the-art methods in Bayesian statistics rely on Markov Chain Monte Carlo (MCMC) algorithms [16], [17], eliminating the need to normalize the distribution, which is often an intractable problem [18]. Variational Inference (VI) relies instead on fitting the posterior with a family of parametric probability distributions that can be sampled [19], [20]. Neural samplers offer another alternative by approximating the posterior with a generative neural network [21], [22].

Normalizing Flows (NFs) infer the probability density function (pdf) for each sample using invertible mappings [23], [24], [25]. While NFs do not require density estimates, they have been shown to require a prohibitive number of layers to effectively match a target distribution in more than one dimension [26]. However, the depth of such models can lead to challenges like vanishing or exploding gradients, which are even exacerbated by the inherent conditioning difficulties of NFs [27].

For robotic grasping, the authors in [28] propose using DRL to find optimal grasps through interaction with multiple real-world robots. If the goal is to find grasping poses explicitly to be used as the target of a classical controller, supervised learning techniques are often utilized [29]. To support various downstream tasks, it would be necessary to find all feasible grasps. To this end, the action space is typically discretized, and grasping success is estimated for each discrete action through heat-maps. This can be learned using supervised [30], [31] or self-supervised [32] methods. Reference [32] explicitly utilizes the structure given by spatial equivariances. We aim to find a solution that needs neither discretization nor the use of the structure, as these requirements are specific to grasping and also restrict applicability to planar picking in carefully crafted environments.

III. OPTIMIZATION PROBLEM

A. ACTION MAPPING

For a state space $\mathcal{S}$ and an action space $\mathcal{A}$, the feasibility model can be expressed through the function

$$g : \mathcal{S} \times \mathcal{A} \to \mathbb{B}, \tag{1}$$

which delineates if a suggested action is feasible in a given state. Given g, the state-dependent set of feasible actions $\mathcal{A}_s^+ \subseteq \mathcal{A}$ contains all actions that are feasible for the state s, i.e., all actions for which $g(s, a) = 1$.

For action mapping, the feasibility policy is defined as

$$\pi_{\text{feasibility}} : \mathcal{S} \times \mathcal{Z} \to \mathcal{A}_s^+. \tag{2}$$

It learns a state-conditioned surjective map from a bounded latent space $\mathcal{Z} \subset \mathbb{R}^m$, with appropriate dimensionality m, into the set of feasible actions for that state. The latent space $\mathcal{Z}$ can be thought of as an infinite set of *indices*. For each *index*, the feasibility policy has to output a different feasible action.

Given the task specifics, an objective policy can be defined that learns the optimal latent value as

$$\pi_{\text{objective}} : \mathcal{S} \to \mathcal{Z}. \tag{3}$$

This optimal *index* in the latent space can then be mapped to a feasible action using $\pi_{\text{feasiblity}}$. Convolving the functions as $(\pi_{\text{feasibility}} \circ \pi_{\text{objective}}) : \mathcal{S} \to \mathcal{A}_s^+$, yields the action mapping policy

$$\pi(s) = \pi_{\text{feasibility}}(s, \pi_{\text{objective}}(s)). \tag{4}$$

TABLE 1. Non-exhaustive list of f-divergences and the corresponding first derivative for gradient estimators.

	$f(t)$	$f'(t)$
JS	$\frac{1}{2}\left[(t+1)\log\left(\frac{2}{t+1}\right)+t\ \log(t)\right]$	$\frac{1}{2}\log\left(\frac{2t}{t+1}\right)$
FKL	$-\log(t)$	$-\frac{1}{t}$
RKL	$t\ \log(t)$	$\log(t)+1$

The f-divergences are obtained by substituting the f functions above in (7) and setting $t = q_\theta/p$. The conventions for p, q, FKL and RKL assume that p is the target distribution, q is the model, and the FKL divergence is $\int p\log(p/q)$.

In this work, we derive how to train the feasibility policy. Since this work only concerns the feasibility policy, the subscript is dropped in the following.

B. FEASIBILITY POLICY

To train the feasibility policy π_θ, we parameterize it with parameters θ and formulate a distribution matching problem. The goal is that π_θ maps every $z \in \mathcal{Z}$ to an $a \in \mathcal{A}_s^+$, without any bias toward any specific feasible actions. Therefore, by sampling uniformly in $\mathcal{Z}$, π_θ should generate a uniform distribution in $\mathcal{A}_s^+$.

When sampling uniformly in $\mathcal{Z}$, π_θ becomes a generator with a conditional probability density function (pdf) $q_\theta(a|s)$. The target distribution is the uniform distribution in the feasible action space given as

$$p(a|s) = \frac{g(s,a)}{\int_{\mathcal{A}} g(s,a')da'}. \tag{5}$$

Given a divergence measure $\mathcal{D}$, the optimal parameters are the solution to the optimization problem

$$\text{argmin}_{\theta\in\Theta} \int_{\mathcal{S}} \mathcal{D}\big(p(\cdot|s)\,||\,q_\theta(\cdot|s)\big)ds, \tag{6}$$

with Θ being the set of possible parameters. The following section details how to iteratively minimize the divergence.

IV. METHODOLOGY

The following derives the gradient w.r.t. θ to iteratively minimize the divergence for a given state. For simplicity of notation, we omit the state and action dependence of q_θ and p.

A. F-DIVERGENCE

As the divergence measure, we choose the f-divergence, a generalization of the Kullback-Leibler (KL) divergence [33]. The f-divergence between two pdfs p and q_θ has the form

$$\mathcal{D}_f(p\,||\,q_\theta) = \int_{\mathcal{A}} p\, f\left(\frac{q_\theta}{p}\right) da, \tag{7}$$

where $f : (0,\infty) \to \mathbb{R}$ is a convex function. Different choices of f lead to well-known divergences as summarized in Table 1. The gradients of the f-divergence w.r.t. θ can be estimated commuting the derivative with the integral [34] and using the score function gradient estimator [35] as

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathcal{D}_f &= \frac{\partial}{\partial\theta}\int_{\mathcal{A}} p\, f\left(\frac{q_\theta}{p}\right) da \\ &= \int_{\mathcal{A}} p\, f'\left(\frac{q_\theta}{p}\right)\frac{1}{p}\frac{\partial}{\partial\theta}q_\theta\, da \\ &= \int_{\mathcal{A}} q_\theta\, f'\left(\frac{q_\theta}{p}\right)\frac{\partial}{\partial\theta}\log q_\theta\, da,\end{aligned} \tag{8}$$

considering that p does not depend on θ. Since q_θ is normalized to 1 and thus $\partial_\theta \int_{\mathcal{A}} q\, da = \int_{\mathcal{A}} q\, \partial_\theta \log q\, da = 0$, a Lagrangian term λ can be added to the gradient:

$$\frac{\partial}{\partial\theta}\mathcal{D}_f = \int_{\mathcal{A}} q_\theta \left(f'\left(\frac{q_\theta}{p}\right)+\lambda\right)\frac{\partial}{\partial\theta}\log q_\theta\, da. \tag{9}$$

If the support of q_θ includes all of $\mathcal{A}$ the above formula in (9) can be rewritten as the expectation on q_θ as

$$\frac{\partial}{\partial\theta}\mathcal{D}_f = \mathbb{E}_{q_\theta}\left[\left(f'\left(\frac{q_\theta}{p}\right)+\lambda\right)\frac{\partial}{\partial\theta}\log q_\theta\right]. \tag{10}$$

Alternatively, using a proposal distribution q' with full support in $\mathcal{A}$, the expectation in (10) can be reformulated as

$$\frac{\partial}{\partial\theta}\mathcal{D}_f = \mathbb{E}_{q'}\left[\frac{q_\theta}{q'}\left(f'\left(\frac{q_\theta}{p}\right)+\lambda\right)\frac{\partial}{\partial\theta}\log q_\theta\right]. \tag{11}$$

B. GRADIENT ESTIMATION

Given a sample $a \sim q_\theta$, it is not possible to directly evaluate $q_\theta(a)$ as it is not available in closed form. Therefore, q_θ needs to be estimated to compute the gradients of the f-divergence. Given N sampled actions $a_i \sim q_\theta$, q_θ can be approximated with a Kernel Density Estimation (KDE) by

$$q_\theta(a) \approx \hat{q}_{\theta,\sigma}(a) = \frac{1}{N}\sum_{a_i\sim q_\theta} k_\sigma(a-a_i), \tag{12}$$

where k_σ is a Gaussian kernel with a diagonal bandwidth matrix σ. The KDE enables the estimation of the expectation. Using (10), computing the expectation value as the average over the samples yields

$$\frac{\partial}{\partial\theta}\mathcal{D}_f \approx \frac{1}{N}\sum_{a_i\sim q_\theta}\left(f'\left(\frac{\hat{q}_{\theta,\sigma}}{p}\right)+\lambda\right)\frac{\partial}{\partial\theta}\log\hat{q}_{\theta,\sigma}. \tag{13}$$

The gradient estimator in (13) did not converge in our experiments. While a systematic investigation of the convergence issue was not completed, we suspect two primary reasons. First, the support q_θ usually does not cover the whole action space $\mathcal{A}$, which is necessary for the expectation formulation in (10). Second, evaluating $q_\theta(a_i)$ based on a KDE, which uses a_j as supports, has a bias for $j = i$.

Adding Gaussian noise to the samples gives full support in $\mathcal{A}$ and reduces the bias at the support points of the KDE, which led to convergence in the experiments. The new samples are given by $a_j^* = a_i + \epsilon$ for $mi \le j < m(i+1)$ and $\epsilon \sim \mathcal{N}(0,\sigma')$, where m indicates the number of samples drawn for each original sample. This is equivalent to sampling from a KDE with a_i as supports and σ' as bandwidth. Using importance

TABLE 2. Gradient estimators of various losses and choice of Lagrangian multiplier λ.

Loss	Actor Gradient Estimator	λ
JS	$\frac{1}{2M}\sum_{a_j^*}\frac{\hat{q}_{\theta,\sigma}}{\hat{q}_{\theta,\sigma'}}\log\left(\frac{2\hat{q}_{\theta,\sigma}}{p+\hat{q}_{\theta,\sigma}}\right)\frac{\partial}{\partial\theta}\log\hat{q}_{\theta,\sigma}$	0
FKL	$-\frac{1}{M}\sum_{a_j^*}\frac{p}{\hat{q}_{\theta,\sigma'}}\frac{\partial}{\partial\theta}\log\hat{q}_{\theta,\sigma}$	0
RKL	$\frac{1}{M}\sum_{a_j^*}\frac{\hat{q}_{\theta,\sigma}}{\hat{q}_{\theta,\sigma'}}\log\left(\frac{\hat{q}_{\theta,\sigma}}{p}\right)\frac{\partial}{\partial\theta}\log\hat{q}_{\theta,\sigma}$	-1
GAN	$\frac{1}{N}\sum_{a_i}\frac{\partial}{\partial a}\log(1-\xi_\phi)\frac{\partial}{\partial\theta}a_i$	-
ME	$\frac{1}{N}\sum_{a_i}\frac{\partial}{\partial\theta}\log\hat{q}_{\theta,\sigma}-\frac{\partial}{\partial a}\log\xi_\phi\frac{\partial}{\partial\theta}a_i$	-

sampling in (11), the gradient in (13) after resampling can be rewritten as follows

$$\frac{\partial}{\partial\theta}\mathcal{D}_f \approx \frac{1}{M}\sum_{a_j^*\sim\hat{q}_{\theta,\sigma'}}\frac{\hat{q}_{\theta,\sigma}}{\hat{q}_{\theta,\sigma'}}\left(f'\left(\frac{\hat{q}_{\theta,\sigma}}{p}\right)+\lambda\right)\frac{\partial}{\partial\theta}\log\hat{q}_{\theta,\sigma}, \quad (14)$$

with $M = mN$. Additionally, align (14) requires an estimate of p, which in turn requires an estimate of the volume in (5)

$$\int_{\mathcal{A}} g(a)\,da \approx \frac{1}{M}\sum_{a_j^*}\frac{g(a_j^*)}{\hat{q}_{\theta,\sigma'}(a_j^*)}. \quad (15)$$

This volume estimation in (15) is similar to self-normalized importance sampling [36] but uses the proposal distribution. The bandwidth σ' of the proposal distribution is a hyper-parameter. Setting $\sigma' = c\,\sigma$, experiments show that in most cases $c > 1$ helps convergence. Intuitively, a larger bandwidth enables the exploration of nearby modes in the action space. Specific estimators for the different f-divergences can be obtained by substituting f' from Table 1 into (14). A summary of the gradient estimators used in this work is given in Table 2.

C. TRAINING PROCESS

Algorithm 1 shows a training loop when training a feasibility policy directly on the feasibility model using a Jensen-Shannon (JS) loss. The training iterates as follows: A batch of random states is sampled, and the actor generates N actions a_i per state. For each action a_i, m values are sampled from a normal distribution $\mathcal{N}(0,\sigma')$ and added to the action values to create the M action samples a_j^*. Using the actions a_i as support of the KDE in (12), the densities $\hat{q}_{\theta,\sigma}(a_j^*)$ and $\hat{q}_{\theta,\sigma'}(a_j^*)$ are computed. Then the feasibility model g is evaluated on all samples a_j^* and the estimate of $p(a_j^*)$ is computed using (5) and importance sampling in (15). Finally, the gradient of θ can be computed according to (14). For a better understanding of the gradient, the trace of the gradient is highlighted in red throughout the algorithm.

Intuitively, the gradient in (14) attracts support actions a_i towards sample actions a_j^* where $p(a_j^*) > \hat{q}_{\theta,\sigma}(a_j^*)$ and repulses support actions from samples where $p(a_j^*) < \hat{q}_{\theta,\sigma}(a_j^*)$. The different f-divergences place different weights on attraction and repulsion. FKL only attracts support actions towards samples with high p, while RKL repulses strongly from samples with $p = 0$, and JS attracts and repulses with lower magnitude.

Algorithm 1 Jensen-Shannon Training Loop

Initialize θ
for 1 **to** *Training Steps* **do**
  for $k = 1$ **to** K **do**
    $s_k \leftarrow$ Sample from $\mathcal{S}$
    $z_i \leftarrow$ Sample uniformly in $\mathcal{Z}$, $\quad \forall i \in [1, N]$
    $a_i \leftarrow \pi_\theta(s_k, z_i)$, $\quad \forall i \in [1, N]$
    $\epsilon_j \sim \mathcal{N}(0, \sigma')$, $\quad \forall j \in [1, M]$
    $a_j^* \leftarrow \text{stop\_gradient}(a_{\lfloor j/m \rfloor}) + \epsilon_j$, $\quad \forall j \in [1, M]$
    // Resample from KDE
    $\hat{q}_j \leftarrow \frac{1}{N}\sum_{i=1}^{N} k_\sigma(a_j^* - a_i)$, $\quad \forall j \in [1, M]$
    // Evaluate KDE on samples
    $\hat{q}'_j \leftarrow \frac{1}{N}\sum_{i=1}^{N} k_{\sigma'}(a_j^* - a_i)$, $\quad \forall j \in [1, M]$
    // Evaluate proposal pdf
    $\hat{r}_j \leftarrow g(s_k, a_j^*)$, $\quad \forall j \in [1, M]$ // Evaluate feasibility model on samples
    $\hat{V} \leftarrow \frac{1}{M}\sum_{j=1}^{M}\frac{\hat{r}_j}{\hat{q}'_j}$ // MC integration with importance sampling
    $\hat{p}_j \leftarrow \frac{\hat{r}_j}{\hat{V}}$, $\quad \forall j \in [1, M]$
    $g_k \leftarrow \frac{1}{2M}\sum_{j=1}^{M}\frac{\hat{q}_j}{\hat{q}'_j}\log\left(\frac{2\hat{q}_j}{\hat{q}_j+\hat{p}_j}\right)\nabla_\theta\log(\hat{q}_j)$
    // gradient trace
  end
  $\theta \leftarrow \theta - \alpha_\theta\frac{1}{K}\sum_{k=1}^{K} g_k$
end

D. ACTOR-CRITIC

Algorithm 1 assumes that the training can be performed directly on the feasibility model. However, multiple actions must be evaluated for the same state to train the actor. This is possible if g is available in closed form or effectively simulated. In some scenarios, g can be a real experiment that does not allow reproducibility of states. To mitigate this problem, an auxiliary neural network $\xi_\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with parameters ϕ can be trained to imitate the environment g. The policy can then be trained to match the distribution of feasible actions according to this auxiliary neural network. We refer to π_θ and ξ_ϕ as actor and critic, respectively.

The actor and critic can be trained simultaneously. The critic is trained on data from a replay memory collected through interactions between the actor and the environment, with each training batch containing half feasible actions and half infeasible actions to stabilize training. To further improve the training efficiency of the critic, when the actor interacts with the environment, it suggests multiple actions, which the

critic evaluates. The action with the highest uncertainty, i.e., the action with $\xi \approx 0.5$ is selected as it contains the most information for the critic. We call this process *maximum uncertainty sampling*. During evaluation, to improve the precision of the actor, the critic can be evaluated on proposed actions, and actions with low values can be rejected. This action optimization can increase precision but may reduce recall or the ability to find all the disconnected sets of feasible actions.

V. ILLUSTRATIVE EXAMPLE

This section provides illustrative examples to elucidate the feasibility policy and demonstrates the potential for direct training on a parallelizable feasibility model across multiple actions for a given state. Hyperparameters, their ranges, and training and inference times are summarized in Table 3.

A. PROBLEM

Consider three circles with given radii and center points as the state s. The feasibility model g deems any point $\mathbf{a}$ a feasible action if it falls within at least one circle and lies inside a unit square, described as follows: $s = (\mathbf{c_k}, r_k)_{k\in 1,2,3}$ where $(\mathbf{c_k}, r_k)$ are the center points and radii of the circles, and $\mathbf{a} \in \mathbb{R}^2$ represents a coordinate. The feasibility model is thus expressed by

$$g(s, \mathbf{a}) = (\mathbf{0} \leq \mathbf{a} \leq \mathbf{1}) \wedge \bigvee_{k=1}^{3} (|\mathbf{a} - \mathbf{c_k}| < r_k). \qquad (16)$$

In the extended example, each circle includes an inner radius, forming annular regions.

B. RESULTS

Figure 1 illustrates the outcomes of applying three distinct divergences, JS, FKL, and RKL, to the circle and annulus scenarios, depicted in subfigures (b) and (c), respectively. Actions are generated from a grid of 256^2 latent values shown in subfigure (a), where each color corresponds to a specific latent value. Three states, marked as (1), (2), and (3), represent various configurations: disconnected shapes, partially connected shapes, and fully connected shapes. The figure visually underscores the different outcomes using the divergences: the RKL approach tends to focus on singular modes, even failing to span overlapping regions, as seen in the third row of both (b) and (c). On the contrary, both FKL and JS exhibit a more expansive coverage, approaching the borders of the feasible space, indicated by the white regions, with the JS divergence showing a reduced density within the infeasible space, represented by the black regions, as compared to FKL. This phenomenon is particularly evident in the first and second states for the circle and annulus examples, which can be attributed to the repulsive gradient present in JS divergence that is absent in the FKL divergence.

These visualizations show that a feasibility policy can be trained to navigate complex distributions beyond the Gaussian reparameterization commonly found in the literature. They further elucidate the importance of enabling the FKL and JS divergences to address disconnected feasible sets effectively. Ultimately, these examples offer an intuitive comprehension of the aim: for the feasibility policy to generate all feasible actions by learning to map the latent space into diverse shapes conditioned on the state.

VI. FEASIBLE TRAJECTORY SEGMENTS EXAMPLE

When solving problems in robotic path planning with reinforcement learning, a standard action space is the direction and velocity target of the robot. However, in tasks that span a long time horizon, it can be beneficial to reduce the number of actions by bundling multiple actions in parametric trajectory segments, often splines, to be followed. Another benefit of generating splines is that these can be checked for collisions with obstacles and other system constraints, such as maximum curvature. This application example shows how learning all feasible actions could be used in this context.

A. PROBLEM

Consider a stationary agent at the center of an environment with known obstacles. In this example, the objective is to find all quadratic splines that fulfill the following conditions

1) does not intersect with any obstacle;
2) longer than a minimum length;
3) shorter than a maximum length;
4) its maximal curvature is less than a threshold.

Figure 2 shows an example scenario with randomly generated obstacles (gray) and example splines. For each constraint, the figure shows an example that violates it, additionally providing examples of feasible splines. The splines are parameterized through the endpoint and an intermediary point that bends the spline, yielding a 4D action space. The feasibility model checks for any constraint violation numerically along the spline. The agent observes the obstacles as a black and white image with size 31×31. It is trained with the JS loss on randomly generated obstacle maps and evaluated on maps not seen during training. The parameters for training, and training and inference times are given in Table 3.

B. RESULTS

Figure 2 shows three example obstacle maps and action samples from the agent. In Figure 2b , the agent provides 256 splines for the randomly generated map, among which 254 are feasible. On the right side of the map, with only two smaller obstacles, the agent produces a wide range of splines that avoid the two obstacles, with a larger margin toward the bottom obstacle. The left side of the map, with larger obstacles and only a smaller gap for feasible paths, shows that the agent also produced a group of splines. Given

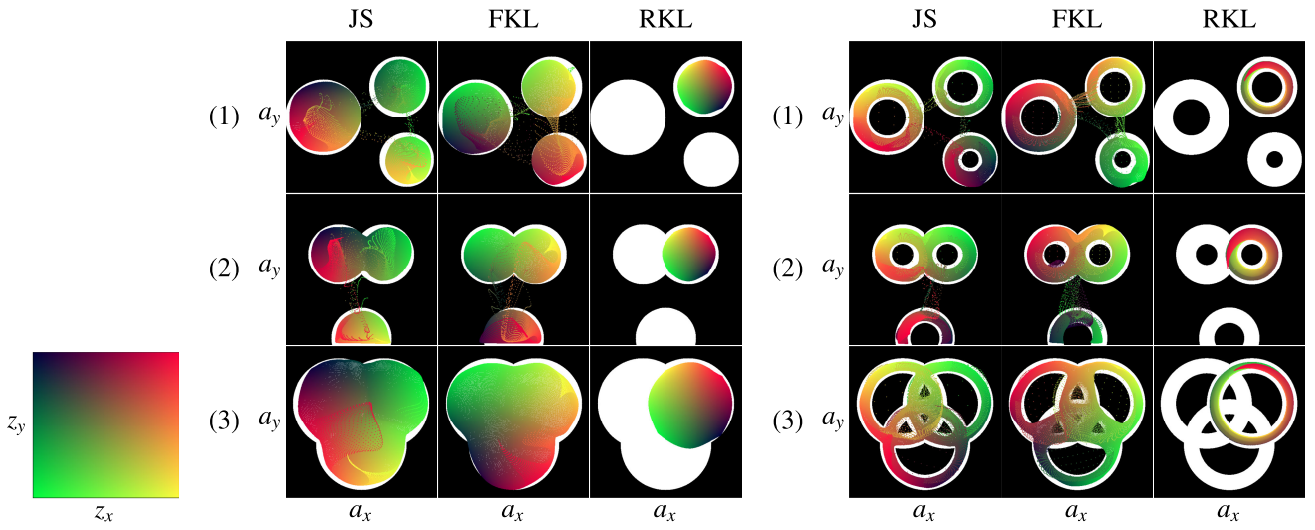


FIGURE 1. Illustrative example showing two feasibility models, which specify feasible regions as the union of three random circles (b) or annuli (c). Three states (1)-(3) are shown for each example, solved with the JS, FKL, and RKL divergence, with feasible and infeasible action space in white and black, respectively. The colored points are actions generated by the feasibility policy when using the corresponding latent space value $(z_x, z_y) \in \mathcal{Z}$ in (a).

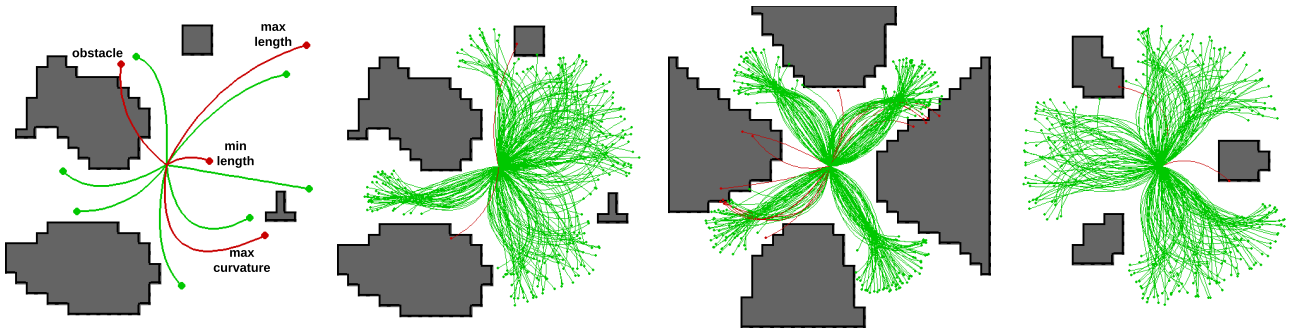


FIGURE 2. Quadratic spline action space application showing three different maps: a randomly generated map in (a) and (b) and two handcrafted maps in (c) and (d). In (a), example splines are shown with green indicating a feasible spline and red indicating an infeasible one. An example for each constraint violation is given. In (b)-(d), the agent generates 256 actions that are displayed with the color depending on the feasibility of each proposed action..

the minimum length constraint on the splines, the splines going to the left are disconnected from the splines on the right, considering the parameter space. The two infeasible splines generated by the agent are likely to be on the transition boundary between these disconnected sets of feasible splines.

Map 2 in Figure 2c shows a situation that contains four disconnected sets of feasible splines, one in each diagonal direction. The agent produces feasible splines in each direction, though generating more infeasible splines. This is likely due to the difficulty of generating four relatively small disconnected sets separated by large volumes of infeasible action space. Map 3 in Figure 2d shows a simpler problem with three small obstacles resulting in three disconnected sets of feasible splines. In this example, the agent again generated 254 feasible splines in all three sets with only two splines when transitioning between sets. Overall, the agent can generate splines in all disconnected sets, largely avoiding generating infeasible splines.

This example shows how action mapping could be applied to motion or path planning problems when they are solved with reinforcement learning. It can be clearly seen that the feasibility policy learned to generate splines representative of all feasible options with only a few infeasible splines. Therefore, an objective policy should be greatly aided if it only needs to choose among the splines that the feasibility policy can generate. In our future work, we plan to investigate action mapping using splines as action space in a reinforcement learning-based path planning problem.

VII. ROBOTIC GRASPING SETUP

Besides the illustrative examples, the proposed method was tested in a simplified robotic grasping simulation, where we compare different f-divergences with other approaches and

investigate how the proposed approach reacts to distortions in the observation.

A. GRASPING SIMULATION

Our grasping simulator generates four shapes (H, 8, Spoon, T) for training and a Box shape for testing. The shape position, orientation, color, and geometry parameters are randomly sampled, producing various observations. The observation space is a 128×128 pixel RGB image. We assume a vertical configuration of a parallel gripper with three degrees of freedom x, y, and α and assume that the object is an extrusion of the 2D shape in the observation. The action space is constrained to the center 78×78 pixel region to avoid undefined behavior at the border of the RGB image. The angle of the grasp is in $[0, \pi)$ as the gripper is symmetrical; thus, a complete revolution is unnecessary.

The success of a grasp is only determined by the relative position and alignment of the gripper to the outline of the object, as seen from a camera positioned above the experiment. Given the alignment of the gripper, i.e., x, y, and α and a simulated picture of the object from a fixed camera, we developed an algorithm that provides a success/failure outcome in a deterministic and reproducible manner. Given the maximum aperture of the parallel gripper l and the width of the gripper claws w, the simulation analyzes the cropped image content of dimensions $l \times w$ between the gripper claws before the claws close on the object. The simulation checks if the object is sufficiently present, equidistant from the claws, and aligned within parameterized margins. Figure 3 shows successful grasping poses and the respective gripper content for the objects that are trained on.

In the primary experiment, we test the algorithm under aligned observation and action spaces. In a second study, we investigate if distortions of the observation affect the performance. The distortions investigated are a rotation, projection, and rotation + projection as shown in Figure 4. These distortions correspond to different camera perspectives. The applied distortion is only on the observation and does not change the mechanics of the experiment.

B. NEURAL NETWORK DESIGN

The neural network that was used for the actor and critic in the robotic experiment is illustrated in Figure 5. The neural network design was guided by simplicity and inspired by Generative Adversarial Networks (GANs). Features that rely on domain-specific knowledge are avoided to evaluate better the learning method presented in the paper. The actor and critic share the residual feature extraction network [37]. The hyperparameters for training and training and inference times are summarized in Table 3.

As a peculiarity of the network and the loss, the actor's inferred action has four components, $[x, y, r\sin\alpha, r\cos\alpha]$, with $r \in [0, \sqrt{2}]$. The angle can be extracted trivially with the arctan of the ratio of the third and fourth action components. As the scale factor r does not change the angle, the critic receives the normalized action $[x, y, \sin\alpha, \cos\alpha]$ as input. To avoid the actor from reaching the singularity at $r = 0$ and the distribution q being spread along the radius, $g(s, a)$ and $\xi(s, a)$ are scaled with an unnormalized Gaussian on the radius, centered at 0.5 with the standard deviation of σ_{sc}.

TABLE 3. List of parameters for all experiments..

	Sec. V	Sec. VI	Sec. VII	Description
N	128	256	128	Support size
M	256	256	256	Resampling size
σ_{xy}	0.01	0.1	0.025	KDE bw. x, y
σ_{sc}	-	-	0.4 (RKL: 0.1)	KDE bw. $\sin\alpha, \cos\alpha$
c	2.0	2.0	3.0	Sampling bw. $\sigma' = c\sigma$
U	-	-	64	Max Uncertainty Proposals
$\lvert\mathcal{M}\rvert$	-	-	320,000	Replay memory size
K	16	16	16	Actor batch size
L	-	-	32	Critic batch size
lr_a	$5 * 10^{-5}$	$5 * 10^{-5}$	$5 * 10^{-5}$	Learning rate actor
lr_c	-	-	$5 * 10^{-5}$	Learning rate critic
$\mathbf{c}$	$[0.0, 1.0]^2$	-	-	Center point range
r	$[0.1, 0.3]$	-	-	Radius range circles
r_o	$[0.2, 0.3]$	-	-	Outer radius range annuli
r_i	$[0.3, 0.7] * r_o$	-	-	Inner radius range annuli
l_{min}	-	0.5	-	Minimum spline length
l_{max}	-	1.0	-	Maximum spline length
c_{max}	-	8.0	-	Maximum curvature
T_t	8 h	4 h	48 h	Training time
I_1	2.2 ms	1.5 ms	4.0 ms	Inference time 1 action
I_{256}	2.2 ms	1.5 ms	8.3 ms	Inference time 256 actions
I_{4096}	9.0 ms	4.0 ms	92.0 ms	Inference time 4096 actions

The training and inference times were measured on an NVIDIA A100 GPU. Inference times for multiple actions measure generating multiple actions for one problem. RKL is sensitive to large KDE bandwidths and benefits from a smaller bandwidth for $\sin\alpha, \cos\alpha$.

C. COMPARISON

In the primary experiment, we are comparing different f-divergences with each other and with two other approaches. The first is a maximum entropy (ME) RL algorithm similar to Soft Actor-Critic (SAC) in [38], which trains the actor to minimize

$$\min_{\theta} \mathbb{E}_{s\sim\mathcal{M}, z\sim\mathcal{Z}} \left[\log q_\theta(\pi_\theta(s, z)|s) - \xi_\phi(s, \pi_\theta(s, z))\right], \quad (17)$$

with $\mathcal{M}$ being the replay memory. The critic is trained as described in Section IV-D. Instead of using the reparameterization trick with a known distribution to estimate the entropy, we use the KDE. The other approach is an implementation of a conditional GAN [39] with a growing dataset. The min-max optimization problem is given through

$$\min_{\theta} \max_{\phi} \mathbb{E}_{\substack{s,a\sim\mathcal{M}_p \\ z\sim\mathcal{Z}}} \left[\log(\xi_\phi(s, a)) - \log(1 - \xi_\phi(s, \pi_\theta(s, z)))\right], \quad (18)$$

with a positive replay memory $\mathcal{M}_p$ only containing feasible actions. An asterisk is added (e.g., JS*) when using action optimization, rejecting 10% of the proposed actions with the lowest critic value.

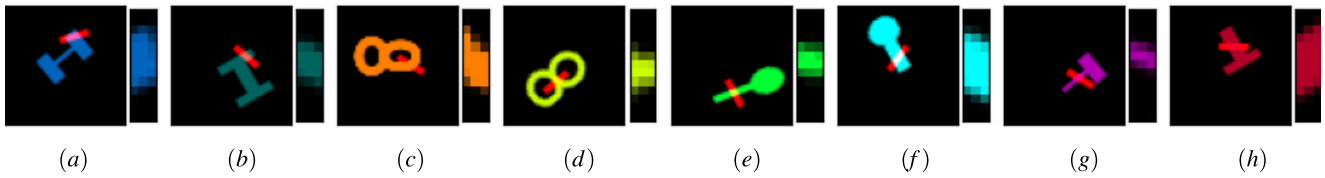


FIGURE 3. Feasible gripper positions (red) for different variations of the shapes (*H-shape* (a+b), *8-shape* (c+d), *Spoon* (e+f), and *T-shape* (g+h)) used in training, with a detailed view of the area between the gripper to the right of each figure.

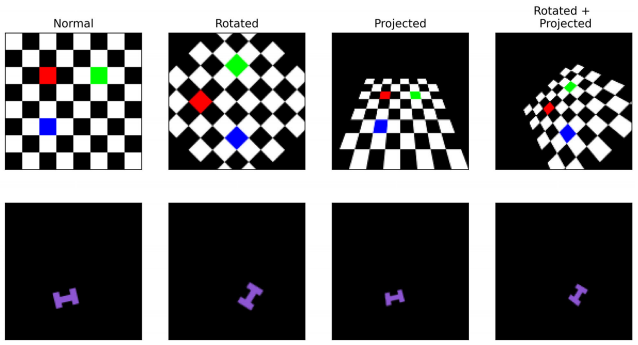


FIGURE 4. Different distortions are applied, showing a colored chess board for illustration and an example shape under all distortions.

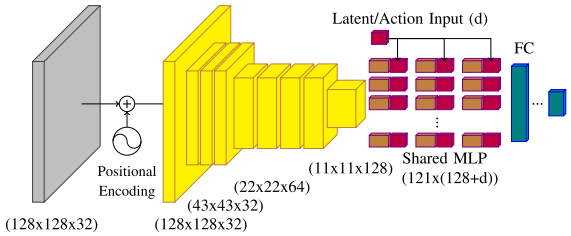


FIGURE 5. Before processing, the image is embedded (in gray) and augmented with positional encoding, resulting in 32 total channels. After positional encoding, a convolutional layer with stride 3, followed by 7 residual blocks (in yellow) with a bottleneck, preprocesses the state. The output is processed by 3 layers of "pixel-wise" shared MLPs (in brown), with the features being concatenated with a latent input (in purple) of length d. The latent input is a random sample from $\mathcal{Z}$ for the actor and the action to be evaluated for the critic. Four (for the actor) or three (for the critic) fully connected layers (in blue) output the action and the feasibility estimate, respectively.

In the secondary evaluation, we compare with a common approach in the literature [32] that uses spatial equivariance. The domain-specific approach utilizes fully convolutional networks to output a probability of success for each action of a *discretized* action space. As in [32], the observation is fed into the neural network multiple times with different rotations. The neural network then only needs to output a one-channel image containing the probability of success of each discretized x, y action for the given rotation of the image. This approach thus uses translation equivariance by using a convolutional neural network (CNN) and rotation equivariance. In the experiments, we denote it as the heat-map approach (H).

The approach is implemented using fully convolutional networks with an hourglass structure, adopting the beginning of the Resnet in Figure 5 and adding the same structure in reverse order with nearest-neighbor upsampling. The approach predicts grasping success for 78×78 pixels with 16 rotation angles, trained on a cross-entropy loss on the grasping outcome sampled from the replay buffer. The replay buffer is also filled with imitation learning examples, and maximum uncertainty sampling is applied. For evaluation, the success estimate of each discretized action is used as its probability to be sampled. To increase accuracy, an inverted temperature factor increases the difference between higher and lower score actions. Specifically, the actions are sampled according to

$$q(a|s) = \frac{\exp(\beta \log \xi(s,a))}{\sum_{\forall a \in \mathcal{A}_d} \exp(\beta \log \xi(s,a))}, \tag{19}$$

with ξ being the fully convolutional network with s as input and as output shape the discretized action space $\mathcal{A}_d$. The inverted temperature was set to $\beta = 100$ for H and $\beta = 1000$ for H^*.

VIII. ROBOTIC GRASPING RESULTS

A. TOP-DOWN OBSERVATION

For each configuration, three agents were trained for 1 million interaction steps with the environment, taking approximately 48 hours per agent on a single NVIDIA 40GB A100 GPU. At the start of the training, 80k examples, including positives and negatives, for randomly generated shapes were added to the replay memory to bootstrap the critic and discriminator learning. The training architecture is implemented in TensorFlow [40] with the parameters in Table 3.

Figure 6 shows the problem, the ground truth feasible picking positions, the critic estimate, and a heat-map of the actor's proposed actions. All figures are projections taking the maximum over the dimension that is not shown. In the problem visualization in Figure 6a, five feasible picks are shown in different colors, which correspond to the markers in Figure 6b. These markers highlight the complex multimodality of the problem. While it appears that, e.g., red and purple are in the same mode in the x-y projection, it is visible in the x-α projection that they are not directly connected. Figure 6c shows that the critic has an approximate understanding of the feasible regions of the action space, showing five modes clearly in the x-y projection. The actor distribution in Figure 6d also shows all five modes, while the output is significantly sharper in the centers of the modes. This is due to the use of the KDEs and the choice of bandwidth σ.

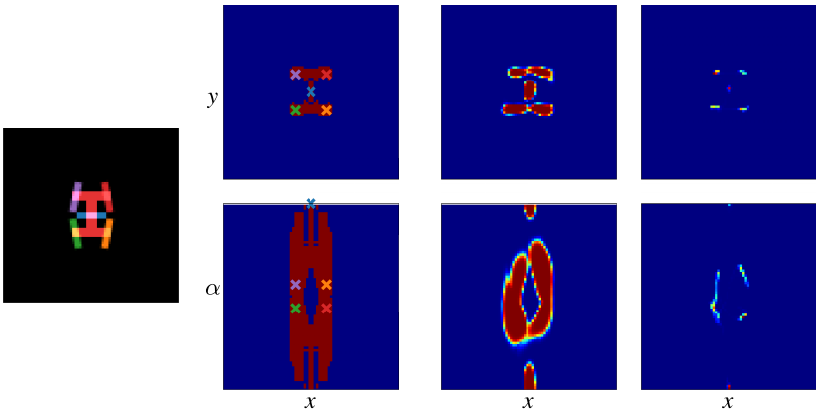


FIGURE 6. Critic classification and actor distribution trained with JS compared with the ground truth. Five example grasps are shown in the problem and their associated locations in the ground truth. The figures show projections onto the x-y plane (top row) and the x-α plane (bottom row).

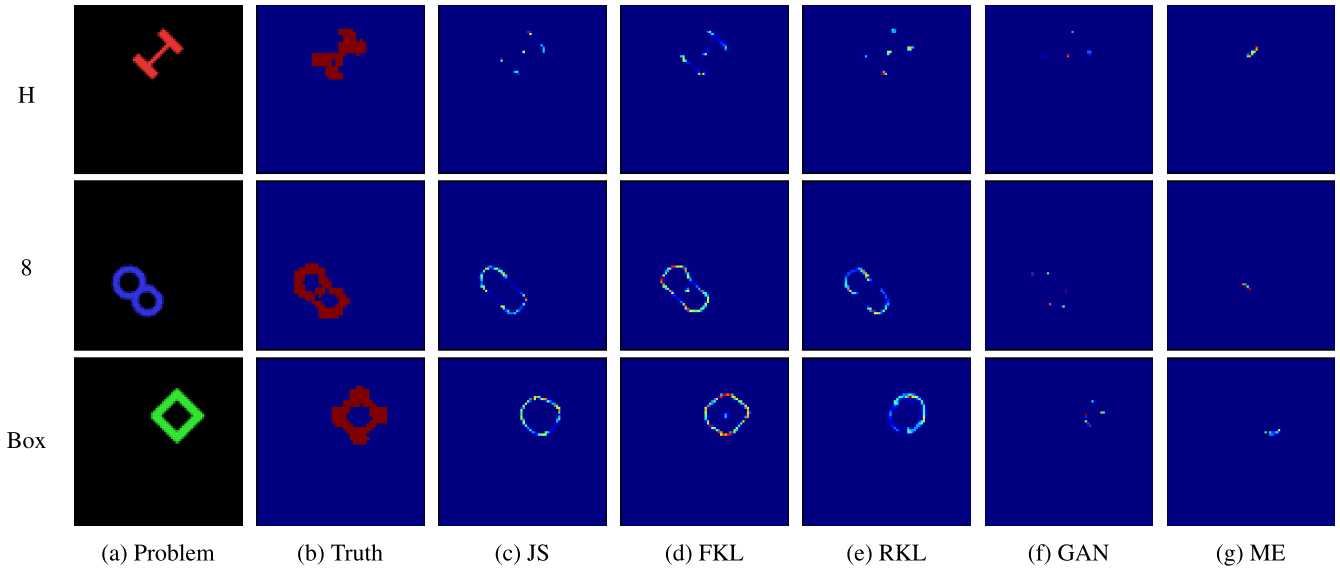
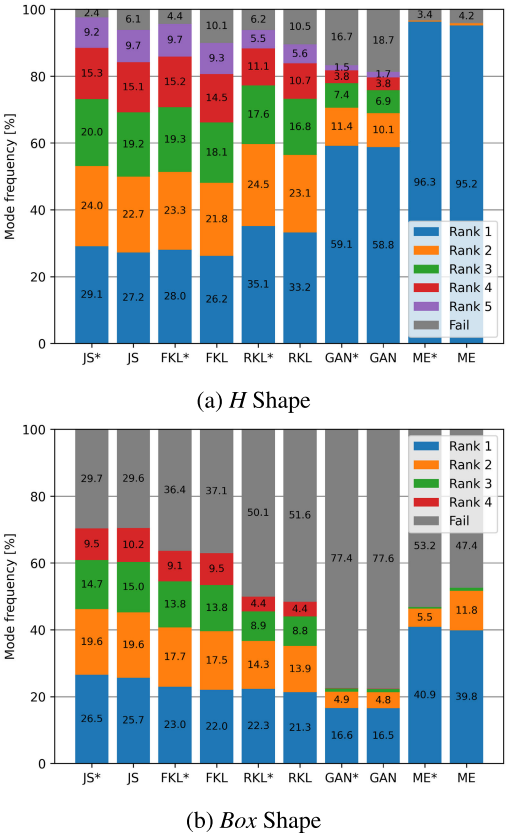


FIGURE 7. Qualitative comparison of the implemented algorithms, showing action heat-maps on three different states, with the last state never been observed during training.

In the qualitative comparison in Figure 7, the actor distributions of the different algorithms are shown for three different shapes. While the *H* and *8* shapes were trained on, the *Box* shape has not been seen during training. The different subfigures show the action heat-maps of all implemented algorithms, showing only the x-y projections. The *H*-row shows that Jensen-Shannon (JS) and Forward Kullback-Leibler (FKL) learned all five modes, with JS having the fewest samples in the connecting area. Against the expectation from the illustrative examples, Reverse Kullback-Leibler (RKL) also learned all modes. The most probable reason is that the actor learns to match the critic's distribution, changing simultaneously from a rough estimate of one feasibility region to the refined shape of individual modes. If the actor learns the entire distribution of the critic early on, when the critic learns to distinguish different modes, the actor's distribution has support in all modes and is thus trapped in each mode. The GAN implementation shows four very unbalanced modes. Additionally, the modes are single points, which correspond to the automatically generated imitation examples, showing that the GAN approach can only imitate but cannot find other feasible actions. The ME implementation collapses in a single mode. The *8*-row and the *Box*-row show a similar pattern with the most pronounced spread of the action distributions in JS, FKL, and RKL and mostly collapsed action regions in the other approaches.

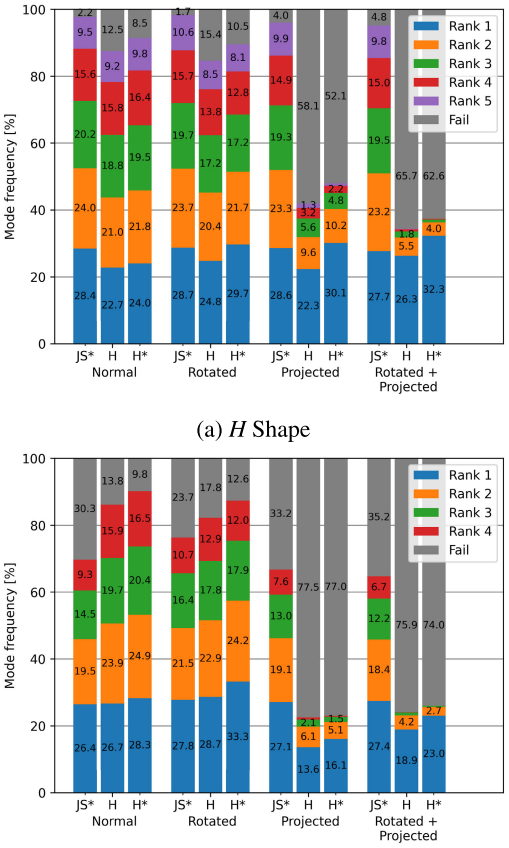
Each algorithm's accuracy and shares of modes on all shapes were evaluated to quantify the capability of generating actions



(a) *H* Shape

(b) *Box* Shape

FIGURE 8. Gripping rank comparison, with the ratio of picks for each ranked mode or failure in %.



(a) *H* Shape

(b) *Box* Shape

FIGURE 9. Gripping rank comparison, with the ratio of picks for each ranked mode or failure in %.

in all disconnected sets of feasible actions. 1024 random states were generated for each shape that differed in pose, color, and geometry. For each state, 1024 actions were sampled from the different actors. The actions were then evaluated, and the mode of each action was recorded. The modes were then ranked and averaged over all the states of that shape by frequency. By averaging the ranks instead of the modes, the last rank shows the average ratio of the least frequent mode for each state.

Figure 8 shows the shares of each rank for the *H* and *Box* shapes for all the algorithms. This figure presents the multimodal capabilities of the proposed approaches. For the *H* shape, JS and FKL have the most balanced distribution over the grasping modes. The GAN approach sometimes generates actions in all the modes but primarily focuses the actions in a primary mode. The ME approach almost exclusively generates actions in one mode. The comparison on the *Box* shape shows that the generalization capability of the JS and FKL algorithms outperform the other approaches, which could indicate that explicitly learning all feasible actions improves generalization. The generalization capability of the GAN implementation is significantly lower than the others, as seen on the *Box* shape, indicating that that approach overfitted on the imitation examples.

To quantify the overall performance, Table 4 shows the precision (feasible actions generated over total actions generated) for each shape and the last ranked mode for the *H*, *T*, and *Box* shapes. The table shows that ME has solid performance on all shapes trained on but has lower generalization performance and fails to find the different modes. The GAN algorithm shows some actions in the last ranked modes, but it is significantly weaker overall. The best approach is JS with the highest precision and similar shares in the last ranked mode as FKL. As discussed before, action optimization improves precision but reduces recall, slightly decreasing the least ranked mode for most approaches. The maximum deviations in the superscript show that all approaches learn reliably, with the GAN having the highest performance deviations among runs.

B. OBSERVATION VARIATION EXPERIMENTS

For each observation distortion, we trained one agent using the JS loss and one agent using the heat-map approach, each for 10^6 training steps. The results are shown in Figure 9 and Table 5, which compare the performance of the proposed Jensen-Shannon (JS) approach with the heat-map (H) approach. As expected, the domain-specific heat-map approach performs well on the original problem. In that scenario, no scene understanding is required, and only local features need to be considered to estimate grasping success.

TABLE 4. Grasping score and mode comparison..

		JS*	JS	FKL*	FKL	RKL*	RKL	GAN*	GAN	ME*	ME
Score	H	$\mathbf{97.6}^{0.0}$	$93.9^{0.1}$	$95.6^{0.5}$	$89.9^{0.6}$	$93.8^{1.6}$	$89.5^{1.3}$	$83.3^{5.9}$	$81.3^{6.1}$	$96.6^{0.3}$	$95.8^{0.5}$
	T	$\mathbf{98.2}^{0.6}$	$96.2^{0.6}$	$97.1^{0.4}$	$93.5^{0.1}$	$96.1^{1.4}$	$93.2^{1.3}$	$84.8^{5.7}$	$82.8^{5.3}$	$96.7^{0.1}$	$95.8^{0.7}$
	8	$\mathbf{93.0}^{1.2}$	$88.4^{1.4}$	$89.5^{1.0}$	$84.5^{0.7}$	$87.3^{4.3}$	$83.7^{4.6}$	$58.9^{7.8}$	$57.3^{7.9}$	$86.6^{1.1}$	$87.2^{2.6}$
	Spoon	$\mathbf{98.8}^{0.5}$	$98.5^{0.5}$	$97.4^{0.6}$	$94.2^{1.4}$	$98.2^{0.5}$	$96.9^{0.9}$	$86.5^{6.4}$	$86.2^{6.9}$	$96.7^{0.5}$	$96.6^{0.7}$
	Box	$70.3^{4.3}$	$\mathbf{70.4}^{4.0}$	$63.6^{2.4}$	$62.9^{2.1}$	$49.9^{13.5}$	$48.4^{12.7}$	$22.6^{3.3}$	$22.4^{4.0}$	$46.8^{1.0}$	$52.6^{16.0}$
	Avg	$\mathbf{91.6}^{0.8}$	$89.5^{0.7}$	$88.6^{0.3}$	$85.0^{0.1}$	$85.1^{3.9}$	$82.3^{3.7}$	$67.2^{5.2}$	$66.0^{5.4}$	$84.7^{0.1}$	$85.6^{3.6}$
Mode	H	$9.2^{0.2}$	$9.7^{0.3}$	$\mathbf{9.7}^{0.8}$	$9.3^{0.5}$	$5.5^{0.3}$	$5.6^{0.3}$	$1.5^{2.2}$	$1.7^{2.3}$	$0.0^{0.0}$	$0.0^{0.0}$
	T	$13.8^{0.8}$	$14.4^{1.1}$	$17.1^{0.6}$	$\mathbf{17.3}^{0.2}$	$9.1^{3.0}$	$9.4^{3.0}$	$2.0^{1.3}$	$2.0^{1.3}$	$0.0^{0.0}$	$0.0^{0.0}$
	Box	$9.5^{1.2}$	$\mathbf{10.2}^{0.9}$	$9.1^{0.7}$	$9.5^{0.7}$	$4.4^{2.2}$	$4.4^{2.0}$	$0.1^{0.1}$	$0.1^{0.1}$	$0.0^{0.0}$	$0.0^{0.0}$

Comparison on all shapes with the mean of the grasping success ratio in % on top and the least ranked mode in % on the bottom, with the maximum deviations over the three runs in superscript.

TABLE 5. Grasping score and mode comparison under perspective distortions..

		Normal			Rotated			Projected			Rotated + Projected		
		JS*	H	H*	JS*	H	H*	JS*	H	H*	JS*	H	H*
Score	H	**97.8**	87.5	91.5	**98.3**	84.6	89.5	**96.0**	41.9	47.9	**95.2**	34.3	37.4
	T	**98.9**	88.4	92.3	**98.9**	87.2	91.8	**97.4**	41.9	46.0	**96.2**	38.6	40.7
	8	**91.6**	84.8	89.4	**94.9**	80.9	86.4	**90.3**	24.5	28.0	**86.6**	17.2	19.0
	Spoon	**99.4**	89.0	93.0	**98.9**	88.0	92.2	**98.3**	43.5	46.1	**97.1**	38.0	40.9
	Box	69.7	86.2	**90.2**	76.3	82.2	**87.4**	**66.8**	22.5	23.0	**64.8**	24.1	26.0
	Avg	**91.5**	87.2	91.3	**93.5**	84.6	89.4	**89.8**	34.9	38.2	**88.0**	30.5	32.8
Mode	H	9.5	9.2	**9.8**	**10.6**	8.5	8.1	**9.9**	1.3	0.7	**9.8**	0.2	0.0
	T	13.8	17.9	**18.5**	12.8	**16.5**	15.5	**8.2**	1.6	0.9	**8.8**	0.4	0.2
	Box	9.3	15.9	**16.5**	10.7	**12.9**	12.0	**7.6**	0.7	0.3	**6.7**	0.1	0.0

Therefore, the approach is expected to generalize well to unseen shapes, as seen for the Box-Shape, since the grasping success depends only on gripper alignment. It only needs to learn to imitate the grasping success heuristic shown in Figure 3.

Rotating the observation does not seem to impact its performance. However, under projection and projection + rotation, the heat-map approach fails to learn to grasp reliably. Our proposed approach learns well under all distortions. In general, the performance of our proposed approach does not depend on the distortion as it does not explicitly use the spatial structure. Its design does not depend on the specifics of the experiment at all. It can, therefore, learn independently of the distortion applied as long as the object is still fully observable.

IX. DISCUSSION

This paper introduced the concept of action mapping, in which an optimization process can be learned sequentially by first learning feasibility and then learning the objective. In this paper, we focused on the former part by learning to generate all feasible actions. We showed that by formulating a distribution matching problem and deriving a gradient estimator for general f-divergences, we train a feasibility policy that can function as a map between a latent space and the feasible action space. An illustrative example, a robotic path planning example, and experiments for robotic grasping show that our approach allows the feasibility policy to generate actions in all disconnected sets of feasible actions, a challenging task for state-of-the-art approaches. Enabling FKL and JS through our gradient estimator was instrumental.

Our experiments, detailed in Table 3, reveal that training time varies significantly across different setups, with no clear correlation to increases in dimensionality. Surprisingly, the 2D system described in Section V required more training time than the 4D system in Section VI. While our results do not show increased complexity with higher dimensions, we anticipate that scalability to higher-dimensional action spaces may still pose challenges. Nevertheless, adopting alternative non-parametric density estimators from existing literature could help mitigate these scalability concerns.

Given the proposed method for training the feasibility policy from a feasibility model, the following steps will focus on action mapping. We will test it in reinforcement learning scenarios for which a feasibility model is known. A potential problem could be that the rough transition between disconnected sets of feasible actions makes deterministic objective policies more challenging. An added regularizing loss on smoothness could improve the transition, all be it by likely reducing accuracy. Further, the approach is very sensitive to the KDE bandwidth. We may need to adapt it throughout training, learn it, or derive a better estimate based on the Jacobian of the network.

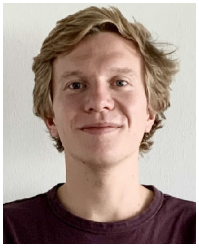
REFERENCES

[1] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *Proc. 15th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2009, pp. 99–107.

[2] H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, and A. Garg, "Conservative safety critics for exploration," in *Proc. Int. Conf. Learn. Represent.*, Oct. 2021, pp. 1–27.
[3] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 3387–3395.
[4] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–10.
[5] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," in *Int. FLAIRS Conf. Proc.*, vol. 35, 2022, pp. 1–14.
[6] H. Krasowski, X. Wang, and M. Althoff, "Safe reinforcement learning for autonomous lane changing using set-based prediction," in *Proc. IEEE 23rd Int. Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2020, pp. 1–7.
[7] M. Theile, H. Bayerlein, M. Caccamo, and A. L. Sangiovanni-Vincentelli, "Learning to recharge: UAV coverage path planning through deep reinforcement learning," 2023, *arXiv:2309.03157*.
[8] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–11.
[9] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1437–1480, 2015.
[10] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Trans. Autom. Control*, vol. 64, no. 7, pp. 2737–2752, Jul. 2019.
[11] Z. Li, U. Kalabic, and T. Chu, "Safe reinforcement learning: Learning with supervision using a constraint-admissible set," in *Proc. Annu. Amer. Control Conf. (ACC)*, Jun. 2018, pp. 6390–6395.
[12] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," 2018, *arXiv:1801.08757*.
[13] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *Proc. 18th Eur. Control Conf. (ECC)*, Jun. 2019, pp. 3420–3431.
[14] L. Sha, "Using simplicity to control complexity," *IEEE Softw.*, vol. 18, no. 4, pp. 20–28, Jul. 2001.
[15] B. Zhong, A. Lavaei, H. Cao, M. Zamani, and M. Caccamo, "Safe-visor architecture for sandboxing (AI-based) unverified controllers in stochastic cyber–physical systems," *Nonlinear Anal., Hybrid Syst.*, vol. 43, Dec. 2021, Art. no. 101110.
[16] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, p. 97, Apr. 1970.
[17] A. E. Gelfand and A. F. M. Smith, "Sampling-based approaches to calculating marginal densities," *J. Amer. Stat. Assoc.*, vol. 85, no. 410, p. 398, Jun. 1990.
[18] J. K. Kruschke, "Chapter 5-Bayes' rule," in *Doing Bayesian Data Analysis*, 2nd ed., J. K. Kruschke, Ed. Washington, DC, USA: National Academy Press, 2015, ch. 5, pp. 99–120.
[19] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Mach. Learn.*, vol. 37, no. 2, pp. 183–233, Nov. 1999.
[20] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Found. Trends Mach. Learn.*, vol. 1, nos. 1–2, pp. 1–305, 2007.
[21] S. Nowozin, B. Cseke, and R. Tomioka, "F-GAN: Training generative neural samplers using variational divergence minimization," in *Advances in Neural Information Processing Systems*, vol. 29, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2016.
[22] T. Hu, Z. Chen, H. Sun, J. Bai, M. Ye, and G. Cheng, "Stein neural sampler," 2018, *arXiv:1810.03545*.
[23] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proc. International Conference Machine Learning*, Jun. 2015, pp. 1530–1538.
[24] E. G. Tabak and C. V. Turner, "A family of nonparametric density estimation algorithms," *Commun. Pure Appl. Math.*, vol. 66, no. 2, pp. 145–164, Feb. 2013.
[25] E. G. Tabak and E. Vanden-Eijnden, "Density estimation by dual ascent of the log-likelihood," *Commun. Math. Sci.*, vol. 8, no. 1, pp. 217–233, 2010.
[26] Z. Kong and K. Chaudhuri, "The expressive power of a class of normalizing flow models," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 3599–3609.
[27] F. Koehler, V. Mehta, and A. Risteski, "Representational aspects of depth and conditioning in normalizing flows," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 5628–5636.
[28] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proc. Int. Conf. Robot. Learn.*, 2018, pp. 651–673.
[29] K. Kleeberger, R. Bormann, W. Kraus, and M. F. Huber, "A survey on learning-based robotic grasping," *Current Robot. Rep.*, vol. 1, no. 4, pp. 239–249, Dec. 2020.
[30] S. Kumra, S. Joshi, and F. Sahin, "Antipodal robotic grasping using generative residual convolutional neural network," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 9626–9633.
[31] D. Morrison, P. Corke, and J. Leitner, "Learning robust, real-time, reactive robotic grasping," *Int. J. Robot. Res.*, vol. 39, nos. 2–3, pp. 183–201, Mar. 2020.
[32] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "TossingBot: Learning to throw arbitrary objects with residual physics," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1307–1319, Aug. 2020.
[33] F. Liese and I. Vajda, "On divergences and informations in statistics and information theory," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4394–4412, Oct. 2006.
[34] P. L'Ecuyer, "On the interchange of derivative and expectation for likelihood ratio derivative estimators," *Manage. Sci.*, vol. 41, no. 4, pp. 738–748, 1995.
[35] J. Kleijnen and R. Rubinstein, "Optimization and sensitivity analysis of computer simulation models by the score function method," *Eur. J. Oper. Res.*, vol. 88, no. 3, pp. 413–427, 1996.
[36] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
[39] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014, *arXiv:1411.1784*.
[40] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, *arXiv:1603.04467*.

MIRCO THEILE (Graduate Student Member, IEEE) received the M.Sc. degree in electrical engineering and information technology from the Technical University of Munich, Germany, in 2018, where he is currently pursuing the Ph.D. degree. He is a Visiting Researcher with the University of California at Berkeley, USA. His research interests include reinforcement learning in applications of cyber-physical systems, including UAVs, robotics, and real-time systems.

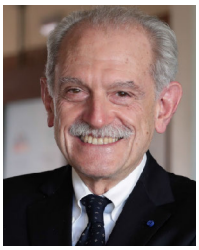
DANIELE BERNARDINI (Member, IEEE) received the M.Sc. degree in theoretical physics from the University of Florence, in 1997. He is pursuing the Ph.D. degree with the School of Computation, Information and Technology, Technical University of Munich (TUM), since 2022, beside his role with the School of Engineering and in the startup. After graduation, he spent two more years as a Researcher with Ludwig Maximilians University Munich, before transitioning to the industry, where he gained more than 20 years of experience in software development and data science. In 2021, he joined the School of Engineering and Design of the TUM as a Research Group Leader, where he focused on advancing perception for robotic manipulation. Since 2021, he has been the Co-Founder and the CEO of Cognivix, a startup specializing in automation solutions for industries requiring high-variability and low-volume production.



RAPHAEL TRUMPP (Graduate Student Member, IEEE) received the M.Sc. degree in mechanical engineering from the Technical University of Munich, in 2021, where he is currently pursuing the Ph.D. degree in informatics. His research interests include machine learning, especially combining deep reinforcement learning with classical control methods. He is also interested in applying these to interactive multi-agent scenarios, such as autonomous racing and robotics.

CRISTINA PIAZZA (Senior Member, IEEE) received the B.Sc. degree in biomedical engineering, the M.S. degree in automation and robotics engineering, and the Ph.D. degree (summa cum laude) in robotics from the University of Pisa, Italy, in 2019. Subsequently, she moved to Chicago, USA, where she was a Postdoctoral Researcher with Northwestern University. Since 2020, she has been a tenure-track Assistant Professor with the Technical University of Munich

MARCO CACCAMO (Fellow, IEEE) received the Ph.D. degree in computer engineering from Scuola Superiore Sant'Anna, Italy, in 2002. Shortly after graduation, he joined the University of Illinois at Urbana–Champaign as an Assistant Professor of computer science and was promoted to a Full Professor, in 2014. Since 2018, he has been with the Chair of Cyber-Physical Systems in Production Engineering, Technical University of Munich, Germany. In 2003, he received the NSF CAREER Award. He is a recipient of the Alexander von Humboldt Professorship.



ALBERTO L. SANGIOVANNI-VINCENTELLI (Life Fellow, IEEE) is currently the Edgar L. and Harold H. Buttner Chair of Electrical Engineering and Computer Sciences with the University of California at Berkeley. Previously, he was the Co-Founder of Cadence and Synopsys, the two leading companies in the area of electronic design automation. He is the author of over 1000 articles, 17 books, and three patents in the area of design tools and methodologies, large-scale systems, embedded systems, hybrid systems, and AI. He is a Board Member of eight companies, including Cadence, and the Chairperson of the Board of Quantum Motion, Innatera, Phoelex, e4Life, and Phononic Vibes. He was a recipient of several academic honors and research awards, including the IEEE/RSE Wolfson James Clerk Maxwell Medal "for groundbreaking contributions that have had an exceptional impact on the development of electronics and electrical engineering or related fields," the BBVA Frontiers of Knowledge Award in the Information and Communication Technologies Category, the Kaufmann Award for Seminal Contributions to EDA, the IEEE Darlington Award, the EDAA Lifetime Achievement Award, and four Honorary Doctorates from the University of Aalborg, KTH, AGH, and the University of Rome, Tor Vergata.