Distributional Treatment of Real2Sim2Real for Object-Centric Agent Adaptation in Vision-Driven DLO Manipulation

Georgios Kamaras[1] and Subramanian Ramamoorthy[1]

***Abstract*—We present an integrated (or end-to-end) framework for the Real2Sim2Real problem of manipulating deformable linear objects (DLOs) based on visual perception. Working with a parameterised set of DLOs, we use likelihood-free inference (LFI) to compute the posterior distributions for the physical parameters using which we can approximately simulate the behaviour of each specific DLO. We use these posteriors for domain randomisation while training, in simulation, object-specific visuomotor policies (i.e. assuming only visual and proprioceptive sensory) for a DLO reaching task, using model-free reinforcement learning. We demonstrate the utility of this approach by deploying sim-trained DLO manipulation policies in the real world in a zero-shot manner, i.e. without any further fine-tuning. In this context, we evaluate the capacity of a prominent LFI method to perform fine classification over the parametric set of DLOs, using only visual and proprioceptive data obtained in a dynamic manipulation trajectory. We then study the implications of the resulting domain distributions in sim-based policy learning and real-world performance.**

***Index Terms*—Probabilistic inference, reinforcement learning, perception for grasping and manipulation**

I. Introduction

DEFORMABLE linear object (DLO) manipulation is an active area of robotics research that deals with challenging tasks, such as lace tying, surgical suturing, and rope whipping [1]–[3]. Even when we know a DLO geometry, adapting a policy to its finer physical parameters is crucial to achieve precise physical interactions. For this paper, let us consider a *visuomotor* reaching task that seemingly requires less dexterity, but indicates readiness to scale up toward the aforementioned tasks. We hold a DLO above a table and want to guide its entire body toward a 2D visual target within a fixed time horizon, using visual and proprioceptive observations. Our objective can be quantified as minimising the overall distance of the DLO body to the target (Fig. 1).

Let us think of how efficiently we would solve such a task; intuitively estimating DLO length and stiffness, from a few visual observations of manipulation trajectories, and adapting our actions to this estimation. An effective sequence of such actions would maximise whole-body convergence to the target for a given DLO parameterisation, while implicitly minimising adversarial factors, such as DLO drag on the table surface. Achieving a similar level of robotic dexterity requires conditioning control policies to physical parameter estimations [4], [5], while dealing with the high dimensionality and nonlinearity of DLO states, challenges further exacerbated by the inherent noise in visual servoing [6], [7].

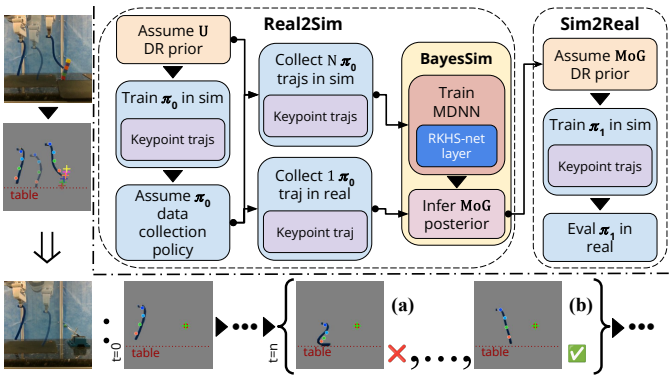

Fig. 1. Overview of our Real2Sim2Real framework (top). We perform LFI for the posterior distribution $\hat{p}$ over system parameters (Real2Sim). We use $\hat{p}$ to perform domain randomisation while training a PPO agent to perform a DLO reaching task. We deploy and evaluate our sim-trained policy in the real world (Sim2Real). We experiment with a DLO reaching task, which is an abstraction of more intricate physical interactions, such as whipping the top cube of a stack (left). We demonstrate strong object-centric agent adaptation relative to the underlying domain distribution $\hat{p}$. For example (bottom), an effective task policy should not drag the DLO on the table (a), but reach for the green target (cross) with a larger part of its body (b).

Learning policies in simulation and then performing a *Sim2Real* deployment [3], [8] follows the hypothesis that multiple simulated iterations of a task will robustify the learnt policy to parametric variations [9]. This requires overcoming the *"reality gap"* of robotic simulators, which is particularly problematic for soft objects. Thus, we first need a reliable *Real2Sim* calibration [10], [11] of the simulator parameters $\boldsymbol{\theta}$ to real-world parameters. Then, for robust deployment, we account for any uncertainty about these parameters by exposing our learning algorithm to different $\boldsymbol{\theta}$ hypotheses.

Likelihood-free inference (LFI) deals with solving the inverse problem of probabilistically mapping real-world sensor observations to the respective simulation parameters $\boldsymbol{\theta}$ that are most likely to account for the observations. This involves inferring the multimodal distribution $\hat{p}(\boldsymbol{\theta})$ from which we can sample sets of simulation parameters with a high probability of replicating the modelled real-world phenomenon [12]. Any uncertainty over $\boldsymbol{\theta}$ can be modelled as the variances associated with the modes.

Received 24 February 2025; accepted June, 5, 2025. Date of publication 20 June 2025. This article was recommended for publication by Associate Editor J. Zhu and Editor M. Vincze upon evaluation of the reviewers' comments. The work of Georgios Kamaras was supported by EPSRC, as part of the CDT in RAS of Heriot-Watt University and The University of Edinburgh. The work of Subramanian Ramamoorthy was supported by UKRI Turing AI World Leading Researcher Fellowship on AI for Person-Centred and Teachable Autonomy under Grant EP/Z534833/1. For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising from this submission. *(Corresponding author: Georgios Kamaras.)*

[1]The authors are with School of Informatics, The University of Edinburgh, EH8 9AB, UK. (e-mail: gkamaras@ed.ac.uk; s.ramamoorthy@ed.ac.uk).

Digital Object Identifier (DOI): 10.1109/LRA.2025.3581744.

This enables us to use $\hat{p}(\boldsymbol{\theta})$ as a sampler while performing Domain Randomisation (DR). DR aims to create a variety of simulated environments, each with randomised system parameters, and then to train a policy that works in all of them for a given control task objective. Assuming that the parameters of the real system is a sample in the distribution associated with the variations seen at training time, i.e. $\hat{p}(\boldsymbol{\theta})$, we can expect that a policy learnt in simulation under this regime will adapt to the real-world dynamics.

Recent literature has demonstrated how distributional embeddings, such as reproducing kernel Hilbert spaces [13] of inferred low-dimensional keypoint trajectories, provide robustness to visual data noise and permutation invariance [14], thus enabling robust Real2Sim calibration. On the Sim2Real side, we have seen fruitful model-based Reinforcement Learning (RL) approaches [15], [16] to train deformable object manipulation policies in simulation and deploy them in the real world. However, we have yet to see an end-to-end *Real2Sim2Real* system which combines the expressiveness of Bayesian inference with the flexibility of model-free RL [17].

In this paper, we make the following contributions.

1) We propose an **end-to-end Real2Sim2Real framework** for robust **vision-based** DLO manipulation.
2) We examine the capacity of BayesSim [12] with distributional state embeddings to **finely classify** the physical properties of a DLO drawn from a parametric set.
3) We study the implications of **different randomisation domains** for model-free RL policy learning in simulation and how this translates to **real-world performance**.

Our experiments show that for a parameterised set of (real) DLOs, an integrated distributional treatment of parameter inference, policy training, and zero-shot deployment enables inferring fine differences in physical properties and adapting RL agent behaviour to them.

II. Background

A. Likelihood-free Inference

LFI treats a simulator as a black-box generative model g with intractable likelihood, which uses its parameterisation $\boldsymbol{\theta}$ to generate an output $\mathbf{x}$ that represents the behaviour of the simulated system [18]. This process defines a likelihood function $p(\mathbf{x} \mid \boldsymbol{\theta})$, also called the forward model of the system, which we cannot evaluate, but we can sample from by running the simulator. To overcome the "reality gap", we want to solve the inverse problem, which is mapping real observations $\mathbf{x}^r$ to the parameters $\boldsymbol{\theta}$ that are most likely to replicate them in simulation; let $\mathbf{x}^s = g(\boldsymbol{\theta})$. This defines the problem of approximating the posterior $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x}^s, \mathbf{x}^r)$.

BayesSim [12] approximates the posterior $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r)$ by learning the conditional density function $q_\phi(\boldsymbol{\theta} \mid \mathbf{x})$, parameterised by $\boldsymbol{\phi}$. Conditional density is modelled as a mixture of Gaussians (MoG), which can be approximated by mixture density neural networks (MDNN) [19]. The inputs $\mathbf{x}$ can be high-dimensional state-action trajectories, summary statistics $\psi(\cdot)$ of trajectory data, or kernel mappings.

BayesSim first requires training $q_\phi(\boldsymbol{\theta} \mid \mathbf{x})$ on a dataset of N pairs $(\boldsymbol{\theta}_n, \mathbf{x}_n)$, where the parameters $\boldsymbol{\theta}_n$ are drawn from a *proposal prior* $\tilde{p}(\boldsymbol{\theta})$ and the observation trajectories $\mathbf{x}_n$ are generated by running $g(\boldsymbol{\theta}_n)$ for the duration of a simulated episode and collecting the respective state-action pairs.

Then, given a single real world trajectory $\mathbf{x}^r$, BayesSim estimates the posterior as:

$$\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r) \propto \frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})} q_\phi(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r), \tag{1}$$

which allows the flexibility ('likelihood-free') for a desirable prior $p(\boldsymbol{\theta})$ which is different than the proposal prior. If $\tilde{p}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$, then $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r) \propto q_\phi(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r)$.

B. Domain Randomisation

Domain randomisation using wide uniform priors [20], combined with the inherent instability of training RL algorithms, has been shown to not provide the robustness expected in certain tasks [9], [21]. Using LFI, we obtain a posterior $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r)$ which we expect to be qualitatively more precise, i.e. narrower than the uniform prior, and more accurate, i.e. dense around the system's true parameters.

The goal of RL is to maximise the expected sum of future discounted rewards (by γ) following a policy $\pi_{\boldsymbol{\beta}}(\mathbf{a}_t \mid \mathbf{s}_t)$, parameterised by $\boldsymbol{\beta}$ and sampling action $\mathbf{a}_t$ given state $\mathbf{s}_t$. We formulate DR over the RL problem as maximising the joint objective:

$$\mathcal{J}(\boldsymbol{\beta}) = \mathbb{E}_{\boldsymbol{\theta}}\left[\mathbb{E}_{\boldsymbol{\eta}}\left[\sum_{t=0}^{T-1} \gamma^{(t)} r(\mathbf{s}_t, \mathbf{a}_t) \mid \boldsymbol{\beta}\right]\right], \tag{2}$$

with respect to policy's $\boldsymbol{\beta}$, where $\boldsymbol{\theta} \sim \hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r)$ and $\boldsymbol{\eta} = \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{o}_t\}_{t=0}^{T-1}$ the history of observation $\mathbf{o}_t$, action $\mathbf{a}_t$ pairs over time horizon T. In this paper, we consider $\mathbf{s}_t \equiv \mathbf{o}_t$.

C. Kernel Mean Embeddings & Neural Approximation

Kernel mean embeddings map distributions into infinite-dimensional feature spaces, thus representing probability distributions without information loss to a space that enables useful mathematical operations. This happens by turning the expectation operator into a reproducing kernel Hilbert space (RKHS) inner product, which has linear complexity in the number of training points [13].

Let X be a random variable in sample space $\mathcal{X}$, $\mathcal{F}$ be an RKHS with kernel $k(\mathbf{x}, \mathbf{x}')$, and data points $\mathbf{x} \in \mathcal{X}$. $\mathcal{F}$ is a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ with the inner product $\langle \cdot, \cdot \rangle_{\mathcal{F}}$ [22]. Given a distribution of functions $P(X)$ and a mean embedding map $\mu_X \in \mathcal{F}$, we can recover expectations of all functions, as $\mathbb{E}_X[f(x)] = \langle \mu_X, f \rangle_{\mathcal{F}}, \forall f \in \mathcal{F}$. By Riesz representation theorem [23], μ_X exists and is unique. A common choice for the μ_X mapping is the Radial Basis Function (RBF) kernel, with parameterisable length scale σ.

Following [14], we can define the empirical kernel embedding using i.i.d. samples $\mathbf{x}_1, ..., \mathbf{x}_N$ from $P(X)$, as:

$$\hat{\mu}_X := \frac{1}{N} \sum_{n=1}^{N} \phi(\mathbf{x}_n). \tag{3}$$

With *kernel trick* we avoid operating on infinite-dimensional implicit maps $\phi(\mathbf{x})$, and instead operate on a finite-dimensional Gram matrix $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j), i, j = 1...N$ [22],

which, however, is computationally expensive for large datasets ($\mathcal{O}(n^2)$).

Following [12] (Eq. 12-15) and [14] (Eq. 6) derivations, we can use *Random Fourier Features* (RFF) [24] to approximate a shift-invariant kernel $k(\mathbf{x}, \mathbf{x}')$ by a dot product $\hat{\phi}(\mathbf{x})^T \hat{\phi}(\mathbf{x}')$, which is a more scalable approximation of K. $\hat{\phi}(\mathbf{x})$ is a finite-dimensional approximation of $\phi(\mathbf{x})$, with randomised features of the form $\cos(\boldsymbol{\omega}_m^T \mathbf{x} + \mathbf{b}_m)$ (cos-only) or $\langle \cos(\sigma^{-1} \circ \boldsymbol{\omega}_m^T \mathbf{x}), \sin(\sigma^{-1} \circ \boldsymbol{\omega}_m^T \mathbf{x}) \rangle$ (cos-sin) [25].

Following [14], we use the RFF feature approximation to construct the kernel mean embedding of vectors (3) that can benefit from a distributional representation. This embedding approach can be integrated into a learning architecture in a fully differentiable manner, by constructing a neural network layer (RKHS-Net) that obtains random samples for frequencies $\boldsymbol{\omega}$, biases $\mathbf{b}$, and optionally σ (e.g., in cos-sin), and adjusts them during training by computing gradients w.r.t. the overall architecture loss.

III. Method

A. Keypoint detection from segmentation images

Our perception module receives an RGB image as input, in which it detects the environment objects of interest, i.e. the controlled DLO and the task's target. We use detection masks to create segmentation images, through which we track our task's features for parameter inference and policy learning.

We use segmentation images to efficiently learn an unsupervised keypoint detection model, using the *transporter* method [26], as implemented by [27]. The resulting keypoints, although temporally consistent, are inferred in every timeframe, creating problems such as pixel position noise and permutations [14], particularly when working with real-world image data of deformable objects. The lack of permutation invariance hinders the creation of reward functions specific to DLO parts, such as the tip [2], [28]. Focussing on whole-body guidance abstracts this challenge and raises the importance of inferred posteriors to implicitly learn successful behaviours using less elaborate reward functions.

B. Real2Sim with likelihood-free inference

Considering our prefaced DLO visuomotor reaching task, we want to learn a policy in simulation and deploy it in the real world, without any further fine-tuning. The Real2Sim part of our work deals with calibrating physical parameters that induce the deformable object's behaviour, such as its dimensions and material behaviour, which are difficult to tune manually. Details such as camera and workspace object placement, controller stiffness parameters, etc. are beyond the scope of our work, although they can be incorporated as additional tunable parameters in extensions of our method.

For our inference and domain randomisation experiments, we define a physical parameter vector $\boldsymbol{\theta} = \langle l, E \rangle$, where l denotes the length of our DLO and E denotes its Young's modulus. Thus, we want to infer a joint posterior $\hat{p}$, which contains information on both the dimensions and the material properties of our deformable object. For this, we use the BayesSim-RKHS variant [14].

Algorithm 1 Real2Sim2Real for DLO manipulation

Given: N_{LFI}: inference iterations
Assume uniform proposal prior $\tilde{p}(\boldsymbol{\theta}) \approx U$
Assign reference prior $p_0 \leftarrow \tilde{p}$
Train initial policy $\pi_{\boldsymbol{\beta}_0}(\mathbf{a}_t \mid \mathbf{s}_t)$, $\boldsymbol{\theta} \sim p_0$
Run 1 $\pi_{\boldsymbol{\beta}_0}$ rollout in the real env to collect $\mathbf{x}^r$
// 1. Real2Sim DLO parameter inference (LFI)
$i \leftarrow 0$
while $i < N_{\text{LFI}}$ **do**
  $\{(\boldsymbol{\theta}, \mathbf{x}^s)\}^N \leftarrow$ Run N $\pi_{\boldsymbol{\beta}_0}$ rollouts in sim, $\boldsymbol{\theta} \sim p_i$
  Train q_ϕ over $\{(\boldsymbol{\theta}, \mathbf{x}^s)\}^N$
  $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r) \propto p_i(\boldsymbol{\theta}) \,/\, \tilde{p}(\boldsymbol{\theta}) q_\phi(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r)$
  $i \leftarrow i + 1$
  Update reference prior $p_i \leftarrow \hat{p}$
end while
// 2. Policy training in sim
Train policy $\pi_{\boldsymbol{\beta}_1}(\mathbf{a}_t \mid \mathbf{s}_t)$, $\boldsymbol{\theta} \sim \hat{p}$
// 3. Sim2Real policy deployment
Evaluate $\pi_{\boldsymbol{\beta}_1}$ in the real env by running 1 $\pi_{\boldsymbol{\beta}_1}$ rollout

Following Algorithm 1, we begin by assuming a uniform proposal prior $\tilde{p}(\boldsymbol{\theta})$, which we use to initialise our reference prior $p_0 = \tilde{p}$. We then perform domain randomisation as $\boldsymbol{\theta} \sim p_0$, while training in simulation an initial policy $\pi_{\boldsymbol{\beta}_0}$ for our task. We execute a rollout of $\pi_{\boldsymbol{\beta}_0}$ in the real world environment, to collect a real trajectory $\mathbf{x}^r$ while manipulating a specific DLO. We perform LFI through *multiple* BayesSim iterations [14], [21]. In each inference iteration i, we use $\pi_{\boldsymbol{\beta}_0}$ as a data collection policy, running N rollouts of $\pi_{\boldsymbol{\beta}_0}$ in simulation to collect a dataset $\{(\boldsymbol{\theta}, \mathbf{x}^s)\}^N, \boldsymbol{\theta} \sim \tilde{p}$, on which we train our BayesSim conditional density function q_ϕ. We use our q_ϕ and $\mathbf{x}^r$ to compute the posterior $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}^r)$. We then update our reference prior $p_i = \hat{p}$ and loop again.

We can now adapt our domain randomisation distribution to the latest inferred posterior $\hat{p}$, and we proceed to retrain our task policy $\pi_{\boldsymbol{\beta}_1}$. Our hypothesis is that by sampling $\boldsymbol{\theta}$ from our object-specific posterior, we will get faster $\pi_{\boldsymbol{\beta}_1}$ convergence to consistently successful behaviour, and that running a rollout of $\pi_{\boldsymbol{\beta}_1}$ in the real-world environment we will collect higher cumulative reward for the specific DLO within the task episode's horizon.

C. Keypoint trajectories in RKHS

We use an RKHS-net layer [14] to construct a distributional state representation (Fig. 2), which is provably robust to visual noise and uncertainty due to inferred keypoint permutations. Intuitively, through our kernel mean embeddings [13] we pull data from the input space (keypoint trajectory) to the feature space (RKHS). The explicit locations of the pulled points are not necessarily known or might be uncertain, but the relative similarity (inner product) of the pulled data points is known in the feature space [22]. This gives us noise robustness and permutation invariance, and it generally means that functional representations that are similar to each other are embedded closer within the RKHS, and thus they are more likely to be classified as similar.

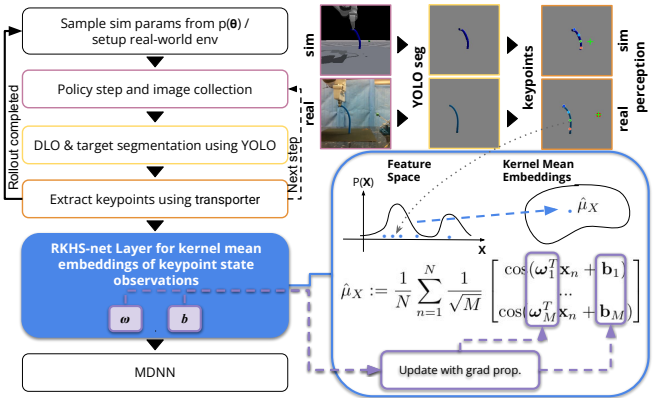

Fig. 2. Overview of our policy rollout and trajectory perception method. In each policy step we extract keypoints from segmentation images. Once all rollouts are completed, for each keypoint trajectory we compute the cos-only kernel mean embeddings of each step's keypoints using the RKHS-net layer (BayesSim-RKHS). Margins colour coding indicates association between sample images and algorithm parts. Illustration inspired by [14].

The input of the RKHS-net layer is a vector of samples of a distribution. In this work, we compute the distributional embedding of a noisy keypoint trajectory $\mathbf{x} = (\mathbf{x_1}, ..., \mathbf{x_n})$, each keypoint defined in a 2D RGB pixel space.

D. Policy Learning and Sim2Real Deployment

We use Proximal Policy Optimisation (PPO) [17] as our reference model-free *on-policy* RL algorithm. PPO maximises the expected reward with a clipped surrogate objective to prevent large updates. It samples actions from a Gaussian distribution, which is parameterised during policy learning. We learn policies in simulation, performing domain randomisation using our LFI *reference prior* p as a task parameterisation sampler, where p is either the default uniform distribution U, or an inferred MoG posterior (Alg. 1, l. 13).

IV. Experimental Setup

Our experiments address the following questions.

1) Can our inferred posteriors **describe different physical parameterisations of similarly shaped DLOs**?
2) What is **the impact of these posteriors on domain randomisation** using the respective MoG?
3) How does this translate into **object-centric agent performance adaptation** during real-world deployment?

A. Reference Task

To evaluate the parameter inference performance of our setup, and the implications of the inferred parameter posterior $\hat{p}(\boldsymbol{\theta})$ for policy learning and deployment, we design a visuo-motor reaching task that studies the guidance of the entire DLO body towards a visual target.

We set up our task with a robot arm that picks up the DLO from a designated position on the table $\mathbf{x}_0$ by grasping it near one of its tips, and raises it to a designated height h_0. Both $\mathbf{x}_0$ and h_0, as well as our initial DLO pose, remain fixed for all of our simulation and real experiment iterations. The execution of our policy begins immediately after the object has been picked up and raised to h_0. Thus, the dynamic nature of our task is constituted by the underactuated manner in which we control our DLO, which dangles from one of its tips, as well as the momentum forces (drag, inertia) acting on its body ever since it was raised from the table.

The positions of the DLO and the target are tracked in the 2D pixel space of an RGB image. The DLO is tracked by an inferred cluster of 4 keypoints, extracted over a segmentation image, and the target with the computed pixel space median of its own segmentation mask, which we treat as an artificial 5th keypoint. In both our parameter inference (real2sim) and policy training and deployment (sim2real) experiments, we position-control our Panda end-effector (EEF) by commanding its Cartesian pose. Our simulator uses the IsaacGym attractors implementation, whereas in the real world we use a Cartesian impedance controller [29].

B. Simulation & Hardware setup

We setup our simulation in IsaacGym [30]. Our simulated environment contains a Franka Emika Panda 7-DoF robot arm equipped with a parallel gripper, which is on top of a tabletop workspace. On top of this workspace, a blue DLO exists, implemented as a tetrahedral grid, and simulated using the corotational finite element method of the FleX physics engine. A small green spherical object is our task's reaching target. Across all our parameter inference and policy learning experiments, the simulated DLO is parameterised in the $[1e3, 5e4]$ (Pa) range for Young's modulus and the $[195, 305]$ (mm) range for length. Thus, *median* of our simulated parameter space is $\mu = (2.55e4 \text{ Pa},\ 250\text{mm})$.

Our real-world setup closely replicates the simulation setup. We extend an open-source sample-efficient robotic RL infrastructure [29] for the purpose of evaluating PPO policies [31]. For visual observations, we use a RealSense D435i camera capturing 60 fps, mounted to view our workspace from the right side. To avoid the overhead of a more elaborate sim and real camera calibration, we select our real camera placement so that the captured images qualitatively approximate the respective sim images similar to [14], [32].

For our experiments, we manufacture 4 blue DLOs using Shore hardness A-40 (DLO-0 with len. 200mm), 00-20 (DLO-1 with len. 200mm and DLO-3 with len. 290mm) and 00-50 (DLO-2 with len. 270mm) silicone polymers [33]. Similarly to simulation, our real DLOs are shaped as grids, with negligibly small height and width at 15mm each. Table I summarises our DLO indexes, parameterisations, relative hardness descriptions, and mass. Items are sorted primarily on increasing length and then on increasing softness (as *medium soft* → *soft* → *extra soft*). We will be referencing DLOs using their index, or as "⟨Length⟩; ⟨Shore Hardness⟩".

The real green target ball is made of plasticine. It is simulated with a fixed rigid sphere of radius $r = 1$cm. Since DLO-target physical interactions are irrelevant to our task, this reality gap of the target material is also irrelevant.

C. Perception setup

We collect visual observations using the real-time RGB image stream of our side-view camera. For computational

TABLE I
REAL DLO INDEXES AND PARAMETERISATIONS.

DLO idx.	0	1	2	3
Length	200mm	200mm	270mm	290mm
Shore Hardness	A-40 *(medium soft)*	00-20 *(extra soft)*	00-50 *(soft)*	00-20 *(extra soft)*
Mass	47.47g	42.94g	57.8g	63.94g

efficiency, we focus our observations on our blue DLO and the green target ball using segmentation images. For this, we fine-tune the segmentation task version of YOLOv8.2 [34]. We use a dataset of 183 manually labelled workspace images, pre-processed through auto-orientation and resizing to 256×256, and augmented using random Gaussian blur and salt-and-pepper noise, totalling 439 images. With this dataset, we train the open source YOLO weights for another 64 epochs.

We reduce the dimensionality of our visual observations by applying a learnt model of 4 keypoints to each segmentation image in real time. We train this keypoint model using a dataset of 1500 random policy rollouts of our simulated control task, while sampling our system parameters $\boldsymbol{\theta}$ from a uniform prior. In this data set, we also incorporate a small number of real-world policy rollouts, and we train for 50 epochs. These keypoints are our DLO observation vector.

D. Domain Randomisation setup

In practice, we see that in popular robotics simulators, such as IsaacGym, re-parameterising an existing deformable object simulation to change physical properties such as stiffness would require to re-initialise the whole simulation. This makes it difficult to integrate such simulators in an RL environment that follows the well-established *gym* style [35].

Thus, to perform DR for our RL task, we train our policies in *vectorised* environments [31] and launch parallel instances of our simulation, each with a different $\boldsymbol{\theta}$, sampled by our current reference prior $p(\boldsymbol{\theta})$. To further robustify the Sim2Real transfer of our policies and minimise the need for camera calibration, we introduce a small randomness in our camera and target object positions by uniformly sampling (x, y, z) offsets in the $(\pm 0.025,\ \pm 0.025,\ \pm 0.025)$ (m) and $(\pm 0.02,\ 0,\ \pm 0.02)$ (m) ranges, respectively.

We empirically chose to launch 12 concurrent environments, from 12 respective domain samples, to manage our computationally intensive deformable object simulations. This places the descriptiveness of our inferred posteriors at the forefront of our experimental study, since inaccuracy and imprecision of the inferred MoGs can have worse consequences in small-data experiments. We sample our MoGs in a *low-variance* method, thus each component is likely to contribute to our set of domain samples.

E. Control setup

We command our EEF by sending it cartesian pose commands (7D vectors). For our control experiments, we constrain our EEF motion into two dimensions, moving only along the x and z axes by controlling the respective deltas. Thus, our sampled policy actions are 2D $\langle dx, dz \rangle$ vectors that we sample in the $[-0.06, 0.06]$ (m) range to maintain smooth EEF transitions. Our 12D observations are constructed by flattening and concatenating our EEF's $\langle x, z \rangle$ 2D position vector (proprioception) and the $5 \times$ 2D visual observation, which are the positions of our DLO and target keypoints.

In each state s_t, we compute the distance d_t of our DLO keypoint cluster to the target using the Frobenius norm of the respective distance matrix. Our sparse reward function uses this d_t and a pixel space distance threshold $d_{\text{thresh}} = 1.5$ to count the reward r_t. Thus, whenever $d_t \leq d_{\text{thresh}}$, our respective reward is the distance scaled to $[0.0, 1.0]$, denoted as $r_t = 1.0 - (d_t / d_{\text{thresh}})$. The episode ends if $r_t \geq 0.75$.

For safety reasons, we restrict our EEF motion within the $\langle x, y, z \rangle \in \langle [0.275, 0.6], [-0.1, 0.1], [0.1, 0.5] \rangle$ (m) world frame coordinates, with the robot arm based on $(0, 0, 0)$. Whenever the EEF leaves this designated workspace, the episode ends as a failure with a reward of -1.

F. Parameter Inference & Policy Algorithms training

For parameter inference, we implement the BayesSim-RKHS variant [14] (Fig. 2). Our RKHS-net layer embeds the 5 observed keypoints using 500 random Fourier features and cos-only components. Our MDNN models 4 mixture components with full covariance and uses 3 fully connected layers of 1024 units each. We use Adam optimiser with a learning rate of $1e{-}6$. As in Alg. 1, line 8, we approximate our posterior $\hat{p}(\boldsymbol{\theta})$ through 15 iterations, each increasing the training set with 100 more trajectories, whose parameters $\boldsymbol{\theta}$ have been sampled using the latest posterior.

For policy learning, we use the Stable Baselines3 [31] PPO implementation. To maintain the generality of our proposed methodology, we keep the default implementation hyperparameters, which reflect the hyperparameters originally proposed in the seminal work [17]. We train for $120,000$ total steps, with a maximum duration of episodes of 16 steps, and a batch size of also 16. We empirically tune our real impedance controller damping and stiffness so that 16 real-world steps approximate the respective 16 simulation steps.

V. RESULTS

A. Different Physical Parameterisations in LFI & DR

Figure 3 illustrates our Real2Sim performance using 2D MoG posterior heatmaps, along with the respective scatter plots of the 12 resulting domain samples. Each MoG has 4 components, whose mean, variance, and mixture coefficient are parameterised during inference. The tightness and spread of the posteriors serve as a qualitative indication of the precision of the inference. The crosses that represent the means of different Gaussians capture alternative hypotheses of the observed DLO's parameterisation.

We see that for our set of DLOs, BayesSim-RKHS **correctly classifies the varied softness** (Young's modulus); however, it **struggles to cleanly classify the varied dimension** (length). This is indicated by the MoG variance across each parameter's axis and the respective spread of the component means, which inherently encode any uncertainty about the parameter estimation.

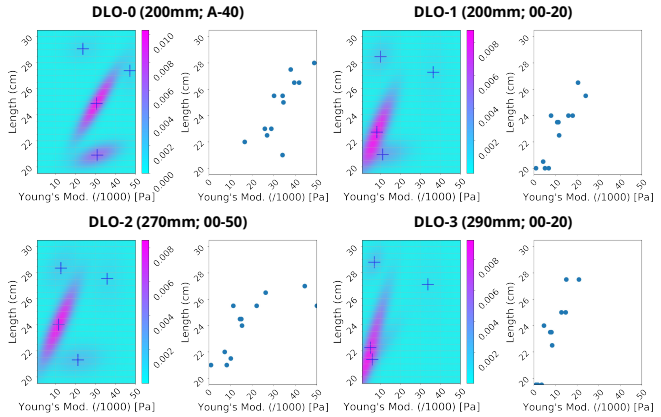

Fig. 3. Inferred MoG posterior heatmaps and the induced domain samples when each MoG is used for DR. MoG component means are displayed in blue crosses and colorbar quantifies likelihood.

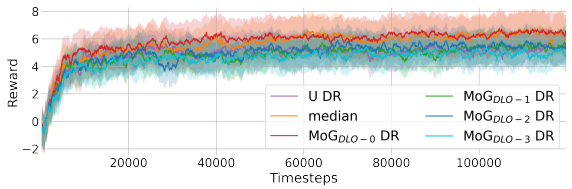

Fig. 4. Learning curves of PPO agent training when performing DR using different domain distributions.

B. Uncertainty over Parameter Estimation & DR

Following the above property of MoG posteriors, we observe how **the relative certainty (*sharpness*) of a posterior** along the dimension of a variable **results in proportionately tightly clustered domain samples** along this dimension. Thus we see domain samples being mainly spread along the less certain length dim. of posteriors. This is particularly visible for the notably similar posteriors of the very soft DLOs 1 and 3. For DLOs 0 and 2, we see that some of the *alternative hypotheses* (component means) for $\boldsymbol{\theta}$ contribute to our low-variance sampling, due to the relative magnitude of the respective mixture coefficients. This contribution is evident in the spread of the respective domain samples.

C. Object-centric Agent Performance

We train (Fig. 4) 4 PPO policies using our 4 inferred MoG posteriors (Fig. 3) for domain randomisation. We also train a policy by performing DR over a uniform distribution (PPO-U) and a policy which assumes that the simulated DLO is parameterised according to the median of our parameter space (PPO-μ). We evaluate these 6 policies in each one of our 4 real-world DLOs, repeating each evaluation 4 times for a total of 96 sim2real policy deployments. Similarly, we also evaluate in simulation over a DLO-μ parameterised per our parameter space median μ.

For each experiment, we report the mean trajectories resulting from the accumulation of commanded EEF position deltas (commanded *actions*, $\langle dx, dz \rangle$) over 4 repetitions. These are presented in Fig. 5, with shaded regions showing the standard deviation between different repetitions. We observe many interesting **motion patterns, which indicate adaptation of agent performance to the physical parameterisation of the respective evaluation DLO**. An indicative subset of these observations is extrinsically corroborated by Fig. 6. The following observations are highlighted in **olive** annotations. Policy underlines match plot line colours.

We see MoG$_{\text{DLO-0}}$ policy (PPO-0) showing the tightest "roaming pattern" **(1)** to maximise the shorter and stiff DLO-0 reward close to the target. PPO-0 also follows the same first half of the trajectory **(2)** for the very soft DLOs 1 and 3. Similarly, the MoG$_{\text{DLO-1}}$ policy (PPO-1), follows a very similar first third of the trajectory **(3)** for the softer DLOs 1, 2 and 3, with some material-related variance, and then attempts a similar "loop pattern" for the last 6 reaching steps with the softest DLOs 1 and 3 **(4)**. In a related adaptation, we see that although both PPO-1 and MoG$_{\text{DLO-3}}$ policy (PPO-3) follow a similar pattern for the latter half of the trajectory for the shorter and stiff DLO-0, the PPO-1 pattern looks cleaner, due to being inherently conditioned on shorter DLOs **(5)**.

For the MoG$_{\text{DLO-2}}$ policy (PPO-2), we see a similar first half of the trajectory for the softest DLOs 1 and 3 (00-20), as well as for DLO-2 (00-50), although with greater variance **(6)**. We also see a similar reaching pattern in the last 6 steps for the longer DLOs 2 (270mm) and 3 (290mm) **(7)**, with differences attributed to their different softness. PPO-2 and PPO-3 also exhibit a similar reaching pattern in the last 6 steps for the shorter and softer DLO-1 **(8)**. The most distinct sign of real-world adaptation to domain distribution occurs in PPO-3 trajectories maintaining a greater distance between the EEF and the table than any other DR policy **(9)**. For DLO-μ, reasonably, PPO-μ is the most consistent (least variance). Notably, all simulated rollouts require a smaller sum of commanded $\langle dx, dz \rangle$ to approach the target, due to the *perfect* dynamics of the robot and controller simulation.

Fig. 7 shows the space of the mean EEF trajectories (4 rep. average) as a similarity heatmap, to illustrate the variation in the policy space for our 6 policies evaluated over our 4 real and 1 simulated (median) DLOs. $U, \mu, 0, 1, 2, 3$ denote the respective PPO trajectories. For DLO-0, we see that the PPO-$\{0, 1\}$ trajectories, both conditioned on equally short 200mm DLOs, show the greatest similarity. For DLO-1, we see that the PPO-$\{U, 2\}$ trajectories are the most similar on average. For the longer DLOs 2 and 3, we have PPO-$\{1, 2\}$ being the most similar on average. For DLO-μ (sim), PPO-$\{1, 3\}$ trajectories are the most different, followed by PPO-$\{U, \mu\}$ and PPO-$\{U, 1\}$. PPO-$\{1, 2\}$ are again the most similar. The greater similarity among all trajectories is due to the median parameters of the DLO, which shows the importance of DLO and domain distribution alignment.

Despite these extrinsic adaptation indications, Fig. 5 averaged rewards and pixel-distance to the target show relatively similar quantitative results, both in real and sim trajectories. This suggests that adaptation occurs at a behavioural level not fully captured by our sparse reward function. This highlights the **importance of trajectory-based evaluation in dynamic DLO manipulation**. Due to drag, inertia, and non-trivial DLO dynamics, success may require nuanced motion patterns that are difficult to capture by scalar proximity metrics.

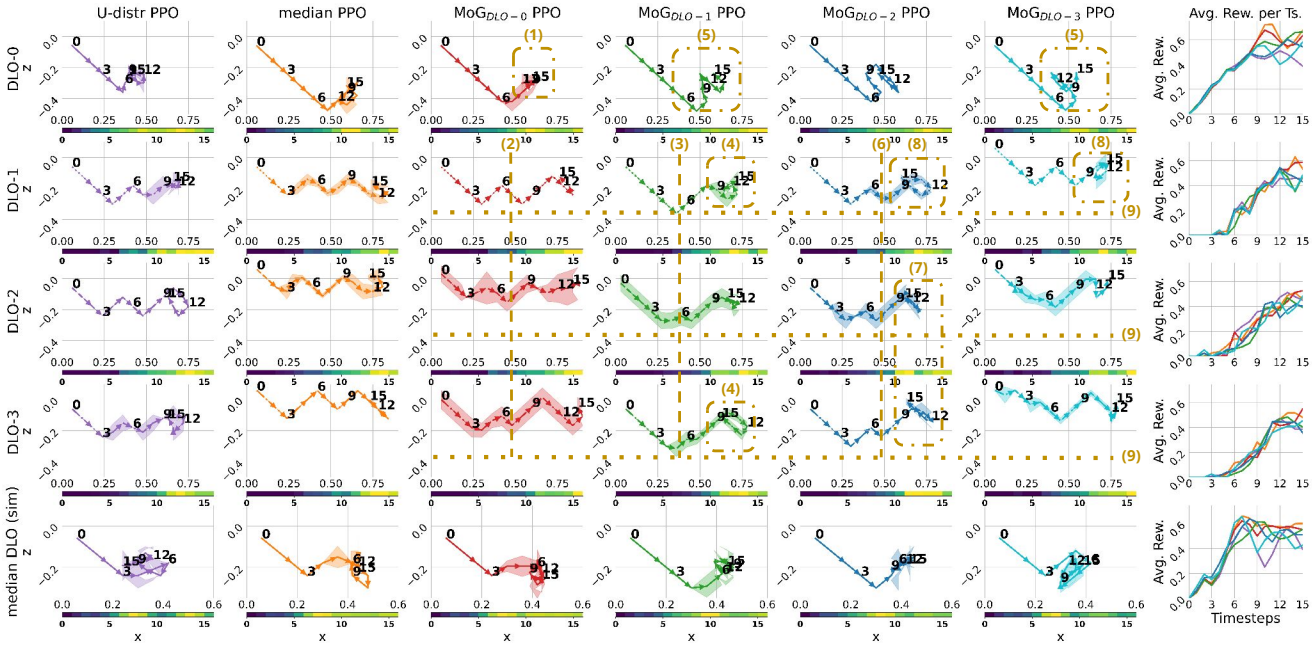

Fig. 5. EEF trajectories during the deployment of 6 policies trained using different domain distributions (col. 1-6) in real world (rows 1-4; 4 real DLOs) and in simulation (row 5; median DLO). We repeat each deployment 4 times and average the measured accumulation of commanded EEF translations along the x and z axes (commanded *actions*, $\langle dx, dz \rangle$), with shaded regions reporting standard deviation. Colorbars illustrate DLO keypoints pixel-distance to the target d_t, reversing reward function for $d_{\text{thresh}} = 1.5$. The rightmost column shows the averaged rewards per timestep for the respective row's trajectories.

VI. Discussion

In this paper, we extend the Real2Sim2Real literature with a proof of concept that an integrated distributional treatment of parameter inference, policy training, and zero-shot deployment can lead to significant agent behaviour adaptation over a set of parameterised DLOs. We use a prominent LFI method (BayesSim) to capture fine parameterisation differences of similarly shaped DLOs. Using the inferred MoG posteriors for DR during RL policy training may not lead to direct benefits in training performance (although this is largely task dependent); however, we observe strong object-centric agent performance indications during real-world deployment.

Through our framework practitioners of soft robotics and soft body manipulation can condition RL policies on system parameterisations inferred through vision, thus inducing object-centric performance, all the while operating in a shared action and observation space for both parameter inference and policy learning. In this context, a constraint to be addressed is that, although we reduce the reality gap in terms of *realism* (observations), we do not yet account for *physical accuracy* (true states). Thus, e.g., the inferred Young's of a DLO may not match the real value, especially if the Real2Sim algorithm has to account for higher-order $\boldsymbol{\theta}$.

References

[1] H. Zhang, J. Ichnowski, D. Seita, J. Wang, H. Huang, and K. Goldberg, "Robots of the lost arc: Self-supervised learning to dynamically manipulate fixed-endpoint cables," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 2021, pp. 4560–4567.

[2] C. Chi, B. Burchfiel, E. Cousineau, S. Feng, and S. Song, "Iterative residual policy: for goal-conditioned dynamic manipulation of deformable objects," *Int. J. Robot. Res.*, vol. 43, no. 4, pp. 389–404, 2024.

[3] M. Haiderbhai, R. Gondokaryono, A. Wu, and L. A. Kahrs, "Sim2real rope cutting with a surgical robot using vision-based reinforcement learning," *IEEE Trans. Automat. Sci. Eng.*, 2024.

[4] S. Kuroki, J. Guo, T. Matsushima, T. Okubo, M. Kobayashi, Y. Ikeda, R. Takanami, P. Yoo, Y. Matsuo, and Y. Iwasawa, "Gendom: Generalizable one-shot deformable object manipulation with parameter-aware policy," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 2024, pp. 14 792–14 799.

[5] K. Zhang, B. Li, K. Hauser, and Y. Li, "Adaptigraph: Material-adaptive graph-based neural dynamics for robotic manipulation," in *Robotics: Sci. Syst.*, 2024.

[6] V. E. Arriola-Rios, P. Guler, F. Ficuciello, D. Kragic, B. Siciliano, and J. L. Wyatt, "Modeling of deformable objects for robotic manipulation: A tutorial and review," *Front. Robot. AI*, p. 82, 2020.

[7] H. Yin, A. Varava, and D. Kragic, "Modeling, learning, perception, and control methods for deformable object manipulation," *Sci. Robot.*, vol. 6, no. 54, 2021, Art. no. eabd8803.

[8] X. Liang, F. Liu, Y. Zhang, Y. Li, S. Lin, and M. Yip, "Real-to-sim deformable object manipulation: Optimizing physics models with residual mappings for robotic surgery," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 2024, pp. 15 471–15 477.

[9] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 2018, pp. 3803–3810.

[10] B. Mehta, A. Handa, D. Fox, and F. Ramos, "A user's guide to calibrating robotic simulators," in *Proc. Conf. Robot Learn.*, 2021, pp. 1326–1340.

[11] J. Liang, S. Saxena, and O. Kroemer, "Learning active task-oriented exploration policies for bridging the sim-to-real gap," in *Robotics: Sci. Syst.*, 2020.

[12] F. Ramos, R. C. Possas, and D. Fox, "Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators," in *Robotics: Sci. Syst.*, 2019.

[13] K. Muandet, K. Fukumizu, B. Sriperumbudur, and B. Schölkopf, "Kernel mean embedding of distributions: A review and beyond," *Found. Trends Mach. Learn.*, vol. 10, no. 1-2, pp. 1–141, 2017.

[14] R. Antonova, J. Yang, P. Sundaresan, D. Fox, F. Ramos, and J. Bohg, "A bayesian treatment of real-to-sim for deformable object manipulation," *IEEE Robot. Automat. Lett.*, vol. 7, no. 3, pp. 5819–5826, 2022.

[15] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani, and J. Lee, "Transporter networks: Rearranging the visual world for robotic manipulation," in *Proc. Conf. Robot Learn.*, 2021, pp. 726–747.

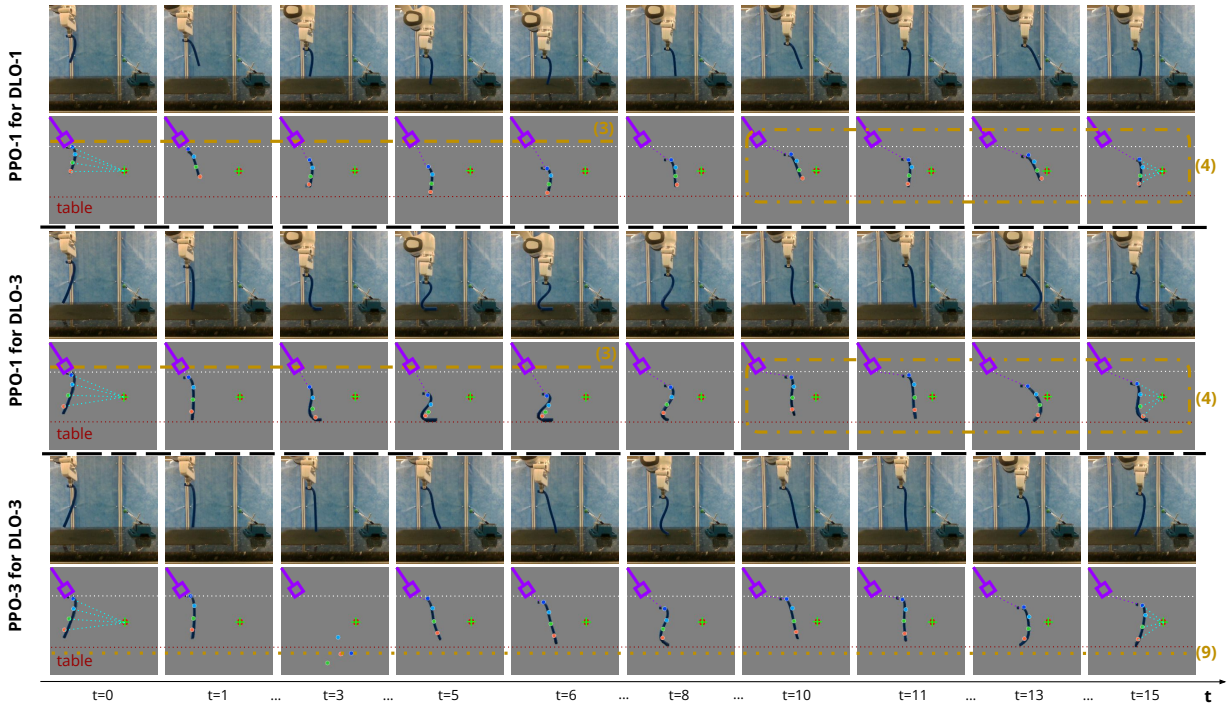

Fig. 6. Timelapses of real-world PPO 1 and 3 policy evaluations using DLOs 1 and 3. In conjuction with Figure 5, we extrinsically confirm reported observations **(3)** (same $1/3$ of trajectory), **(4)** (similar reaching pattern) and **(9)** (moving higher above the table). Dotted white and red lines visually mark the initial EEF z and table surface respectively. A purple indicator on top left of each image marks the initial EEF position, with dotted purple lines marking its visual displacement. Dotted cyan lines in $t = \{0, 15\}$ mark DLO body distance to target.

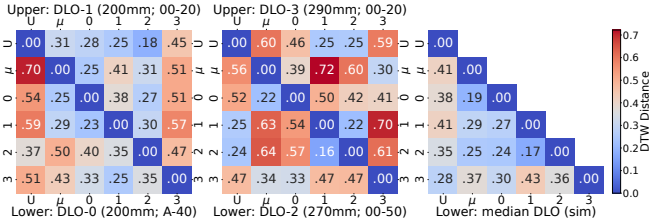

Fig. 7. Dynamic Time Warping (DTW) heatmap of real and sim EEF trajectories (accum. commanded *actions*) over PPO-$\{U, \mu, 0, 1, 2, 3\}$ evaluations.

[16] D. Seita, P. Florence, J. Tompson, E. Coumans, V. Sindhwani, K. Goldberg, and A. Zeng, "Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 2021, pp. 4568–4575.

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

[18] G. Papamakarios and I. Murray, "Fast ε-free inference of simulation models with bayesian conditional density estimation," *Adv. Neural Inf. Process. Syst.*, vol. 29, 2016.

[19] C. M. Bishop, *Mixture density networks*. Aston University, 1994.

[20] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 23–30.

[21] R. Possas, L. Barcelos, R. Oliveira, D. Fox, and F. Ramos, "Online bayessim for combined simulator parameter inference and policy improvement," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5445–5452.

[22] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, "Reproducing Kernel Hilbert space, Mercer's Theorem, Eigenfunctions, Nyström Method, and Use of Kernels in Machine Learning: Tutorial and Survey," *arXiv preprint arXiv:2106.08443*, 2021.

[23] N. I. Akhiezer and I. M. Glazman, *Theory of linear operators in Hilbert space*. Courier Corporation, 2013.

[24] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *Adv. Neural Inf. Process. Syst.*, vol. 20, pp. 1177–1184, 2007.

[25] F. T. Ramos, R. Antonova, A. Handa, and D. Fox, "Method to incorporate uncertain inputs into neural networks," Jun. 1 2023, US Patent App. 17/540,107.

[26] T. D. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih, "Unsupervised learning of object keypoints for perception and control," *Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[27] Y. Li, A. Torralba, A. Anandkumar, D. Fox, and A. Garg, "Causal discovery in physical systems from videos," *Adv. Neural Inf. Process. Syst.*, vol. 33, 2020.

[28] V. Lim, H. Huang, L. Y. Chen, J. Wang, J. Ichnowski, D. Seita, M. Laskey, and K. Goldberg, "Real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 2022, pp. 8282–8289.

[29] J. Luo, Z. Hu, C. Xu, Y. L. Tan, J. Berg, A. Sharma, S. Schaal, C. Finn, A. Gupta, and S. Levine, "Serl: A software suite for sample-efficient robotic reinforcement learning," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 2024, pp. 16 961–16 969.

[30] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance GPU based physics simulation for robot learning," *Adv. Neural Inf. Process. Syst. Datasets and Benchmarks Track*, 2021.

[31] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021.

[32] C. Matl, Y. Narang, R. Bajcsy, F. Ramos, and D. Fox, "Inferring the material properties of granular media for robotic tasks," in *Proc. IEEE Int. Conf. Robot. and Automat.*, 2020, pp. 2770–2777.

[33] Z. Liao, M. Hossain, and X. Yao, "Ecoflex polymer of different shore hardnesses: Experimental investigations and constitutive modelling," *Mech. Mater.*, vol. 144, p. 103366, 2020.

[34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proce. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.

[35] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.