Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators

Fabio Ramos*† Rafael Carvalhaes Possas*† Dieter Fox*‡
*NVIDIA †University of Sydney ‡University of Washington

Abstract—**We introduce BayesSim [1], a framework for robotics simulations allowing a full Bayesian treatment for the parameters of the simulator. As simulators become more sophisticated and able to represent the dynamics more accurately, fundamental problems in robotics such as motion planning and perception can be solved in simulation and solutions transferred to the physical robot. However, even the most complex simulator might still not be able to represent reality in all its details either due to inaccurate parametrization or simplistic assumptions in the dynamic models. BayesSim provides a principled framework to reason about the uncertainty of simulation parameters. Given a black box simulator (or generative model) that outputs trajectories of state and action pairs from unknown simulation parameters, followed by trajectories obtained with a physical robot, we develop a likelihood-free inference method that computes the posterior distribution of simulation parameters. This posterior can then be used in problems where Sim2Real is critical, for example in policy search. We compare the performance of BayesSim in obtaining accurate posteriors in a number of classical control and robotics problems. Results show that the posterior computed from BayesSim can be used for domain randomization outperforming alternative methods that randomize based on uniform priors.**

I. Introduction

Simulators are emerging as one of the most important tools for efficient learning in robotics. With physically accurate and photo-realistic simulation, perception models and control policies can be trained more easily before being transferred to real robots, saving both time and costs of running complex experiments. Unfortunately, in many cases, models and policies trained in simulation are not seamlessly transferable to the real systems. Lack of knowledge about the correct simulation parameters, oversimplified simulation models, or insufficient numerical precision for differential equation solvers can all play a significant role in this problem. To ameliorate this problem, a popular approach is to *sample* different simulation parameters during training and thereby learn models that are robust to simulation perturbations. This approach, often referred to as domain randomization (DR), has been shown to perform surprisingly well in areas such as learning to control a humanoid robot [23], manipulate table top objects [38], estimating 6D object poses from images [17], or dexterous in-hand manipulation [2].

A crucial question regarding domain randomization is which simulation parameters to randomize over and from which distributions to sample their values from. Typically, these parameters and their distributions are determined in a manual process by iteratively testing whether a model learned in randomized simulation works well on the real system. If the model does not work on the real robot, the randomization parameters are changed so that they better cover the conditions observed in the real world. To overcome this manual tuning process, [10] recently showed how policy executions on a real robot can be used to automatically update a Gaussian distribution over the sampling parameters such that the simulator better matches reality. However, by restricting sampling distributions to Gaussians, this approach cannot model more complex uncertainties and dependencies among parameters. Alternatively, one could perform system identification to better estimate simulation parameters from the real data. Since most of these techniques assume that the simulation equations are known and only provide point estimates for the parameters, they do not account for the uncertainties associated with the measurement process, numerical precision of differential equation solvers, or simplistic models [14].

In this paper we provide a principled Bayesian method to compute full posteriors over simulator parameters, thereby overcoming the limitations of previous approaches. Our technique, called BayesSim, leverages recent advances in likelihood-free inference for Bayesian analysis to update posteriors over simulation parameters based on small sets of observations obtained on the real system. The main difficulty in computing such posteriors relates to the evaluation of the likelihood function, which models the relationship between simulation parameters and corresponding system behavior, or observations in the real world. While a simulator *implicitly* defines this relationship, the likelihood function requires the *inverse* of the simulator model, *i.e.*, how observed system behavior can be used to derive corresponding simulation parameters. Importantly, BayesSim does not assume access to the internal differential equations underlying the simulator and treats the simulator as a black box.

We make the following contributions: First, we introduce BayesSim as a generic framework for probabilistic inference with robotics simulators and show that it can provide a full space of simulation parameters that best fit observed data. This is in contrast to traditional system identification methods that only provide the best fitting solution. Second, we propose a novel mixture density random Fourier network to approximate the conditional distribution $p(\boldsymbol{\theta}|\mathbf{x}^r)$ directly by learning from pairs $\{\boldsymbol{\theta}_i, \mathbf{x}_i^s\}_{i=1}^N$ generated from the proposal prior and the simulator. Finally, we show that learning policies with domain randomization where the simulator parameters are randomized

[1] Code available at https://github.com/rafaelpossas/bayes_sim

according to the posterior provided by BayesSim generates policies that are significantly more robust and easier to train than randomization directly from the prior.

II. Related Work

Simulators accelerate machine learning impact by allowing faster, highly-scalable and low cost data collection. Many other scientific domains such as economics [15], evolutionary biology [4] and cosmology [29] also rely on simulator-based modelling to provide further advancements in research. In robotics, "reality gap" is not only seen in control, robotics vision is also affected by this problem [38]. Algorithms trained on images from a simulation can frequently fail on different environments as the appearance of the world can differ greatly from one system to the other.

Randomizing the dynamics of a simulator while training a control policy has proven to mitigate the reality gap problem [25]. Simulation parameters could vary from physical settings like damping, friction and object masses [25] to visual parameters like objects textures, shapes and etc [38]. Another similar approach is that of adding noise to the system parameters [37] instead of sampling new parameters from a uniform prior distribution. Perturbation can also be seen on robot locomotion [22] where planning is done through an ensemble of perturbed models. Lastly, interleaving policy roll outs between simulation and reality has also proven to work well on swing-peg-in-hole and opening a cabinet drawer tasks [11].

Learning models from simulations of data can leverage one's understanding of the physical world potentially helping to solve the aformentioned problem. Until recently, Approximate Bayesian Computation [4] has been one of the main methods used to tackle this type of problem. Rejection ABC [26] is the most basic method where parameter settings are accepted/rejected if they are within a certain specified range. The set of accepted parameters approximates the posterior for the real parameters. Markov Chain Monte Carlo ABC (MCMC-ABC) [20] improves over its precedent by perturbing accepted parameters instead of independently proposing new parameters. Lastly, Sequential Monte Carlo ABC (SMC-ABC) [6] leverages sequential importance sampling to simulate slowly-change distributions where the last one is an approximation of the true parameter posterior. In this work, we use a ϵ-free approach [24] for likelihood-free inference, where a Mixture of Density Random Fourier Network estimates the parameters of the true posterior through a Gaussian mixture.

A wide range of complex robotics control problems have been recently solved using Deep Reinforcement Learning (Deep RL) techniques [2, 25, 37]. Classic control problems like Pendulum, Mountain Car, Acrobot and Cartpole have been successfully tackled using policy search with algorithms like Trust Region Policy Optimization (TRPO) [32] and Proximal Policy Optimization (PPO) [33]. More complex tasks in robotics such as the ones in manipulation are still difficult to solve using traditional policy search. Both Push and Slide tasks (Figure 1) on the fetch robot [8] were only solved recently using the combination of Deep Deterministic Policy Gradients (DDPG) [18] and Hindsight Experience Replay (HER) [1].

III. Preliminaries

In this section we provide background on likelihood free inference and reinforcement learning. As we shall see, policy search via domain randomization is one of the applications in which BayesSim proved to be valuable.

A. Likelihood-free inference

BayesSim takes a *prior* $p(\boldsymbol{\theta})$ over simulation parameters $\boldsymbol{\theta}$, a black box generative model or simulator $\mathbf{x}^s = g(\boldsymbol{\theta})$ that generates simulated observations $\mathbf{x}^s$ from these parameters, and observations from the physical world $\mathbf{x}^r$ to compute the posterior $p(\boldsymbol{\theta}|\mathbf{x}^s, \mathbf{x}^r)$. The main difficulty in computing this posterior relates to the evaluation of the likelihood function $p(\mathbf{x}|\boldsymbol{\theta})$ which is defined *implicitly* from the simulator [12]. Here we assume that the simulator is a set of dynamical differential equations associated with a numerical or analytical solver which are typically intractable and expensive to evaluate. Furthermore, we do not assume these equations are known and treat the simulator as a black box. This allows our method to be utilized with many robotics simulators (even closed source ones) but requires a method where the likelihood cannot be evaluated directly but instead only sampled from, by performing forward simulations. This is referred to in statistics as likelihood-free inference of which the most popular family of algorithms to address it are known as approximate Bayesian computation (ABC) [4, 20, 34].

In ABC, the simulator is used to generate synthetic observations from samples following the parameters prior. These samples are accepted when features or sufficient statistics computed from the synthetic data are similar to those from real observations obtained from physical experiments. As a sampling-based technique, ABC can be notoriously slow to converge, particularly when the dimensionality of the parameter space is large. Formally, ABC approximates the posterior $p(\boldsymbol{\theta}|\mathbf{x} = \mathbf{x}^r) \propto p(\mathbf{x} = \mathbf{x}^r|\boldsymbol{\theta})p(\boldsymbol{\theta})$ using the Bayes' rule. However as the likelihood function $p(\mathbf{x} = \mathbf{x}^r|\boldsymbol{\theta})$ is not available, conventional methods for Bayesian inference cannot be applied. ABC sidesteps this problem by approximating $p(\mathbf{x} = \mathbf{x}^r|\boldsymbol{\theta})$ by $p(\| \ \mathbf{x} - \mathbf{x}^r \ \| < \ \epsilon|\boldsymbol{\theta})$, where ϵ is a small value defining a sphere around real observations $\mathbf{x}^r$, and using Monte Carlo to estimate its value. The quality of the approximation increases as ϵ decreases however, the computational cost can become prohibitive as most simulations will not fall within the acceptable region.

B. Reinforcement learning and policy search in robotics

We consider the default RL scenario where an agent interacts in discrete timesteps with an environment $\mathbf{E}$. At each step t the agent receives an observation $\mathbf{o}_t$, takes an action $\mathbf{a}_t$ and receives a real number reward r_t. In general, actions in robotics are real valued $\mathbf{a}_t \in \mathbb{R}^D$ and environments are usually partially observed so that the entire history of observation, action pairs $\boldsymbol{\eta} = \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{o}_t\}_{t=0}^{T-1}$. The goal is to maximize the expected sum of discounted future rewards by following a policy $\pi(\mathbf{a}_t|\mathbf{s}_t;\boldsymbol{\beta})$, parametrized by $\boldsymbol{\beta}$,

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

Many approaches in reinforcement learning make use of the recursive relationship known as the Bellman equation where Q^{π} is the action-value function describing the expected return after taking an action $\mathbf{a}_t$, in state $\mathbf{s}_t$ and thereafter following policy π.

$$Q^{\pi}(\mathbf{s}_t,\mathbf{a}_t) = \mathbb{E}_{r_t,s_{t+1}}[r(\mathbf{s}_t,\mathbf{a}_t) + \gamma\mathbb{E}_{a_{t+1}}[Q^{\pi}(\mathbf{s}_{t+1},\mathbf{a}_{t+1})]] \tag{2}$$

In recent years, the advancements in traditional RL methods have allowed their application to control tasks with continuous action spaces. Inheriting ideas from DQN [21], Deep Deterministic Policy Gradients have been relatively successful in a wide range of control problems. The main caveat of DDPG algorithms is that they rely on efficient experience sampling to perform well. Improving the way how experience is collected is one of most important topics in today's RL community. Experience Replay [19] and Prioritized Experience replay [31] still performs poorly in a repertoire of robotics tasks where the reward signal is sparse. Hindsight Experience replay (HER) [1], on the other hand, performs well in this scenario as it breaks down single trajectories/goals into smaller ones and, thus, provides the policy optimization algorithm with better reward signals. HER has been mostly based in a recent RL concept: Multi-Goal learning with Universal Function Approximators [30].

Another set of successful policy search algorithms is based on optimization through trust regions. They are less sensitive to the experience sampling problem mentioned above. The maximum step size for exploration is determined by its trust region and the optimal point is then evaluated progressively until convergence has been reached. The main idea is that updates are always limited by their own trust region, and, therefore, learning speed is better controlled. Proximal Policy Optimization [33] and Trust Region Policy optimization [32] have applied these ideas providing state of the art performance in a wide range of control problems.

Both techniques differ on the way they sample experiences. While the first is an off-policy algorithm - experiences are generated by a behaviour policy, the second is an on-policy algorithm where the policy used to generated experience is the same used to perform the control task. These algorithms will have comparable performance on different robotics control scenarios therefore should be considered the current state of the art on such problems.

IV. BayesSim

A. Problem setup

Following [24], BayesSim approximates the intractable posterior $p(\boldsymbol{\theta}|\mathbf{x}=\mathbf{x}^r)$ by directly learning a conditional density $q_{\phi}(\boldsymbol{\theta}|\mathbf{x})$ parameterised by parameters ϕ. As we shall see, $q_{\phi}(\boldsymbol{\theta}|\mathbf{x})$ takes the form of a mixture density random feature network. To learn the parameters ϕ we first generate a dataset with N pairs $(\boldsymbol{\theta}_n,\mathbf{x}_n)$ where $\boldsymbol{\theta}_n$ is drawn independently from a distribution $\tilde{p}(\boldsymbol{\theta})$ referred to as the *proposal prior*. $\mathbf{x}_n$ is obtained by running the simulator with parameter $\boldsymbol{\theta}_n$ such that $\mathbf{x}_n = g(\boldsymbol{\theta}_n)$. In [24] the authors show that $q_{\phi}(\boldsymbol{\theta}|\mathbf{x})$ is proportional to $\frac{\tilde{p}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}p(\boldsymbol{\theta}|\mathbf{x})$ when the likelihood $\prod_n q_{\phi}(\boldsymbol{\theta}_n|\mathbf{x}_n)$ is maximised w.r.t. ϕ. We follow a similar procedure and maximise the log likelihood,

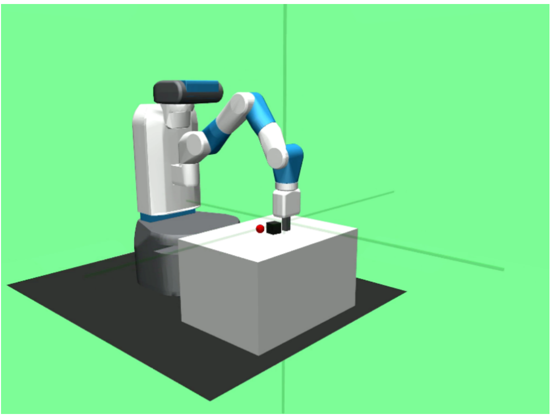

Fig. 1: Fetch Push and Sliding tasks: the robot has full access to the entire table and multiple iterations with the object (pushing) or one shot at pushing the object to its target (sliding).

$$\mathcal{L}(\phi) = \frac{1}{N}\sum_{n}\log q_{\phi}(\boldsymbol{\theta}_{\boldsymbol{n}}|\mathbf{x}_n) \tag{3}$$

to determine ϕ. After this is done, an estimate of the posterior is obtained by

$$\hat{p}(\boldsymbol{\theta}|\mathbf{x}=\mathbf{x}^r) \propto \frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})}q_{\phi}(\boldsymbol{\theta}|\mathbf{x}=\mathbf{x}^r), \tag{4}$$

where $p(\boldsymbol{\theta})$ is the desirable prior that might be different than the proposal prior. In the case when $\tilde{p}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$, it follows that $\hat{p}(\boldsymbol{\theta}|\mathbf{x}=\mathbf{x}^r)q_{\phi}(\boldsymbol{\theta}|\mathbf{x}=\mathbf{x}^r)$. When $\tilde{p}(\boldsymbol{\theta}) \neq p(\boldsymbol{\theta})$ we need to adjust the posterior as detailed in Section IV-E.

B. Mixture density random feature networks

We model the conditional density $q_{\phi}(\boldsymbol{\theta}|\mathbf{x})$ as a mixture of K Gaussians,

$$q_{\phi}(\boldsymbol{\theta}|\mathbf{x}) = \sum_{k}\alpha_k\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k), \tag{5}$$

where $\boldsymbol{\alpha} = (\alpha_1,\ldots,\alpha_K)$ are mixing coefficients, $\{\mu_k\}$ are means and $\{\Sigma_k\}$ are covariance matrices. This is analogous to mixture density networks [5] except that we replace the feedforward neural network with Quasi Monte Carlo (QMC) random Fourier features when computing $\boldsymbol{\alpha}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. We justify and describe these features in the next section.

Denoting $\Phi(\boldsymbol{x})$ as the feature vector, the mixing coeficients are calculated as

$$\boldsymbol{\alpha} = \text{softmax}(\mathbf{W}_{\boldsymbol{\alpha}}\Phi(\mathbf{x}) + \mathbf{b}_{\boldsymbol{\alpha}}). \tag{6}$$

Note that the operator $\text{softmax}(\mathbf{z}_i) = \frac{\exp(z_i)}{\sum_{k=1}^{K} \exp z_k}$ for $i = 1, \ldots, K$ enforces that the sum of coeficients equals to 1 and each coefficient is between 0 and 1.

The means are defined as linear combinations of feature vectors. For each component of the mixture,

$$\boldsymbol{\mu}_k = \mathbf{W}_{\boldsymbol{\mu}_k} \Phi(\mathbf{x}) + \mathbf{b}_{\boldsymbol{\mu}_k}. \tag{7}$$

Finally we parametrize the covariance matrices as diagonals matrices with

$$\text{diag}(\boldsymbol{\Sigma}_k) = \text{mELU}(\mathbf{W}_{\boldsymbol{\Sigma}_k} \Phi(\mathbf{x}) + \mathbf{b}_{\boldsymbol{\Sigma}_k}) \tag{8}$$

where mELU is a modified exponential linear unit defined as

$$mELU(z) = \begin{cases} \alpha(e^z - 1) + 1 & \text{for } z \leq 0 \\ z + 1 & \text{for } z > 0 \end{cases} \tag{9}$$

to enforce positive values. Experimentally this parametrization provided slightly better results than with the exponential function. The diagonal parametrization assumes independence between the dimensions of the simulator parameters $\boldsymbol{\theta}$. This turns out to be not too restrictive if the number of components in the mixture is large enough.

The full set of parameters for the mixture density network is then,

$$\phi = (\mathbf{W}_{\boldsymbol{\alpha}}, \mathbf{b}_{\boldsymbol{\alpha}}, \{\mathbf{W}_{\boldsymbol{\mu}_k}, \mathbf{b}_{\boldsymbol{\mu}_k}, \mathbf{W}_{\boldsymbol{\Sigma}_k}, \mathbf{b}_{\boldsymbol{\Sigma}_k}\}_{k=1}^{K}). \tag{10}$$

C. Neural Network features

BayesSim can use neural network features creating a model similar to the mixture density network in [5]. For a feed-forward neural network with two fully connected layers, the features take the form

$$\Phi(\mathbf{x}) = \sigma(\mathbf{W}_2(\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2), \tag{11}$$

where $\sigma(\cdot)$ is a sigmoid function; we use $\sigma(\cdot) = \tanh(\cdot)$ in our experiments. This network structure was used in the experiments and compared to the Quasi Monte Carlo random features described below.

D. Quasi Monte Carlo random features

BayesSim can use random Fourier features [27] instead of neural nets to parameterise the mixture density. There are several reasons why this can be good choice. Notably, 1) random Fourier features – of which QMC features are a particular type – approximate possibly infinite Hilbert spaces with properties defined by the choice of the associated kernel. In this way prior information about properties of the function space can be readily incorporated by selecting a suitable positive semi-definite kernel; 2) the approximation converges to the original Hilbert space with order $\mathcal{O}(1/\sqrt{s})$, where s is the number of features, therefore independent of the input dimensionality; 3) experimentally, we verified that mixture densities with random Fourier features are more stable to different initialisations and converge to the same local maximum in most cases.

Random Fourier features approximate a shift invariant kernel $k(\boldsymbol{\tau})$, where $\boldsymbol{\tau} = \|\mathbf{x} - \mathbf{x}'\|$, by a dot product $k(\boldsymbol{\tau}) \approx \Phi(\mathbf{x})^T \Phi(\mathbf{x})$ of finite dimensional features $\Phi(\mathbf{x})$. This is possible by first applying the Bochner's theorem [36] stated below:

Theorem 1 *(Bochner's Theorem) A shift invariant kernel $k(\boldsymbol{\tau})$, $\boldsymbol{\tau} \in \mathbb{R}^D$, associated with a positive finite measure $d\mu(\boldsymbol{\omega})$ can be represented in terms of its Fourier transform as,*

$$k(\boldsymbol{\tau}) = \int_{\mathbb{R}^D} e^{-i\boldsymbol{\omega}\cdot\boldsymbol{\tau}} d\mu(\boldsymbol{\omega}). \tag{12}$$

The proof can be found in [13]. When μ has density $\mathcal{K}(\boldsymbol{\omega})$ then $\mathcal{K}$ represents the spectral distribution for a positive semi-definite k. In this case $k(\boldsymbol{\tau})$ and $\mathcal{K}(\boldsymbol{\omega})$ are Fourier duals:

$$k(\boldsymbol{\tau}) = \int \mathcal{K}(\boldsymbol{\omega}) e^{-i\boldsymbol{\omega}\cdot\boldsymbol{\tau}} d\boldsymbol{\omega}. \tag{13}$$

Approximating Equation 13 with a Monte Carlo estimate with N samples, yields

$$k(\boldsymbol{\tau}) \approx \frac{1}{N} \sum_{n=1}^{N} (e^{-i\boldsymbol{\omega}_n \mathbf{x}})(e^{-i\boldsymbol{\omega}_n \mathbf{x}'}), \tag{14}$$

where $\boldsymbol{\omega}$ is sampled from the density $\mathcal{K}(\boldsymbol{\omega})$.

Finally, using Euler's formula ($e^{-ix} = \cos(x) - i\sin(x)$) we recover the features:

$$\begin{aligned} \Phi(\mathbf{x}) = \frac{1}{\sqrt{N}} [&\cos(\boldsymbol{\omega}_1 \mathbf{x} + b_1), \ldots, \cos(\boldsymbol{\omega}_n \mathbf{x} + b_n), \\ &-i \cdot \sin(\boldsymbol{\omega}_1 \mathbf{x} + b_1), \ldots, -i \cdot \sin(\boldsymbol{\omega}_n \mathbf{x} + b_n)]. \end{aligned} \tag{15}$$

where bias terms $\mathbf{b}_i$ are introduced with the goal of rotating the projection and allowing for more flexibility in capturing the correct frequencies.

This approximation can be used with all shift invariant kernels proving flexibility in introducing prior knowledge by selecting a suitable kernel for the problem. For example, the RBF kernel can be approximated using the features above with $\boldsymbol{\omega} \sim \mathcal{N}(0, 2\sigma^{-2}I)$ and $b \sim \mathcal{U}[-\pi, \pi]$. σ is a hyperparameter that corresponds to the kernel length scale and is usually set up with cross validation.

We further adopt a quasi Monte Carlo strategy for sampling the frequencies. In particular we use Halton sequences [7] which has been shown in [3] to have better convergence rate and lower approximation error than standard Monte Carlo.

E. Posterior recovery

From Equation 4 we note that when the proposal prior is different than the desirable prior, we need to adjust the posterior by weighting it with the ratio $p(\boldsymbol{\theta})/\tilde{p}(\boldsymbol{\theta})$.

In this paper we assume the prior to be uniform, either with finite support – defined within a range and zero elsewhere – or improper, constant value everywhere. Therefore,

$$\hat{p}(\boldsymbol{\theta} | \mathbf{x} = \mathbf{x}^r) \propto \frac{q_\phi(\boldsymbol{\theta} | \mathbf{x}^r)}{\tilde{p}(\boldsymbol{\theta})}. \tag{16}$$

When the proposal prior is Gaussian, we can compute the division between a mixture and a single Gaussian analytically.

In this case, since $q_\phi(\boldsymbol{\theta}|\mathbf{x})$ is a mixture of Gaussians and $\tilde{p}(\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, the solution is given by

$$\hat{p}(\boldsymbol{\theta}|\mathbf{x}=\mathbf{x}^r) = \sum_k \alpha'_k \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}'_k, \boldsymbol{\Sigma}'_k) \tag{17}$$

where,

$$\boldsymbol{\Sigma}'_k = \left(\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_0^{-1}\right)^{-1} \tag{18}$$

$$\boldsymbol{\mu}'_k = \boldsymbol{\Sigma}_k^{-1}\left(\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0\right) \tag{19}$$

$$\alpha'_k = \frac{\alpha_k \exp(-\frac{1}{2}\lambda_k)}{\sum_{k'} \alpha_{k'} \exp(-\frac{1}{2}\lambda_{k'})}, \tag{20}$$

and the coefficients λ_k are given by

$$\lambda_k = \log\det\boldsymbol{\Sigma}_k - \log\det\boldsymbol{\Sigma}_0 - \log\det\boldsymbol{\Sigma}'_k + \boldsymbol{\mu}_k^T\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k \\ - \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_k'^T\boldsymbol{\Sigma}_k'^{-1}\boldsymbol{\mu}'_k. \tag{21}$$

F. Sufficient statistics for state-action trajectories

Trajectories of state and action pairs in typical problems can be long sequences making the input dimensionality to the model prohibitive large and computationally expensive. We adopt a strategy commonly used in ABC; instead of inputting raw state and action sequences to the model, we first compute some sufficient statistics. Formally, $\mathbf{x} = \psi(\mathbf{S}, \mathbf{A})$ where $\mathbf{S} = \{\mathbf{s}^t\}_{t=1}^T$ and $\mathbf{A} = \{\mathbf{a}^t\}_{t=1}^T$ are sequences of states and actions from $t = 1$ to T. There are many options in the literature for sufficient statistics for time series or trajectory data. For example, the mean, log variance and autocorrelation for each time series as well as cross-correlation between two time series. Another possibility is to learn these from data, for example with an unsupervised encoder-decoder recurrent neural network [35]. However, such a representation would need to be trained with simulated trajectories and might not be able to capture complexities in the real trajectories. This will be investigated in future work. Here we adopt a simpler strategy and use statistics commonly applied to stochastic dynamic systems such as the Lotka-Volterra model [40].

Defining $\boldsymbol{\tau} = \{\mathbf{s}^t - \mathbf{s}^{t-1}\}_{t=1}^T$ as the difference between immediate future states and current states, the statistics

$$\psi(\mathbf{S}, \mathbf{A}) = (\{\langle\boldsymbol{\tau}_i, \mathbf{A}_j\rangle\}_{i=1,j=1}^{D_s,D_a}, \mathrm{E}[\boldsymbol{\tau}], \mathrm{Var}[\boldsymbol{\tau}]), \tag{22}$$

where D_s is the dimensionality of the state space, D_a is the dimensionality of the action space, $\langle\cdot,\cdot\rangle$ denotes the dot product, $\mathrm{E}[\cdot]$ is the expectation, and $\mathrm{Var}[\cdot]$ the variance.

G. Example: CartPole posterior

We provide a simple example to demonstrate the algorithm in estimating unknown simulation parameters for the famous CartPole problem. In this problem a pole installed on a cart needs to be balanced by applying forces to the left or to the right of the cart. For this example we assume that both the mass and the length of the pole are not available and we use BayesSim to obtain the posterior for these parameters. We assume uniform priors for both parameters and collect 1000 simulations following a rl-zoo policy [2] to train BayesSim. With the model trained, we collected 10 trajectories with the correct parameters to simulate the real observations. Figure 2 shows the posteriors for both problems. As with many problems involving two related variables, `masspole` and pole `length` exhibit statistical dependencies that generate multiple explanations for their values. For example, the pole might have lower mass and longer length, or vice versa. BayesSim is able to recover the multi-modality nature of the posterior providing densities that represent the uncertainty of the problem accurately.

H. Domain randomization with BayesSim

Here we describe the domain randomization strategy to take full advantage of the posterior obtained by the inference method. Given the posterior obtained from the simulation parameters $\hat{p}(\boldsymbol{\theta}|\mathbf{x}=\mathbf{x}^r)$ we maximize the objective,

$$J(\boldsymbol{\beta}) = \mathbb{E}_{\boldsymbol{\theta}}\left[\mathbb{E}_{\boldsymbol{\eta}}\left[\sum_{t-0}^{T-1}\gamma^{(t)}r(\mathbf{s}_t, \mathbf{a}_t)|\boldsymbol{\beta}\right]\right], \tag{23}$$

where $\boldsymbol{\theta} \sim \hat{p}(\boldsymbol{\theta}|\mathbf{x} = \mathbf{x}^r)$ with respect to the policy parameters $\boldsymbol{\beta}$. Since the posterior is a mixture of Gaussians, the first expectation can be approximated by sampling a mixture component following the distribution over $\boldsymbol{\alpha}$ to obtain a component k, followed by sampling the corresponding Gaussian $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

V. Experiments

Experiments are presented in two different cases to demonstrate and assess the performance of BayesSim. In Section V-A we verify and compare the accuracy of the posterior recovered. In Section V-B we compare the robustness of policies trained by randomizing following the prior versus posterior distribution over simulation parameters.

A. Posterior recovery

The first analysis we carry out is the quality of the posteriors obtained for different problems and methods. We use the log probability of the target under the mixture model as the measure, defined as $\log p(\boldsymbol{\theta}_*|\mathbf{x} = \mathbf{x}^r)$, where $\boldsymbol{\theta}_*$ is the actual value for the parameter. We compare Rejection-ABC [26] as the baseline, the recent ϵ-Free [24] which also provides a mixture model as the posterior, and BayesSim using either a two layer neural network with 24 units in each layer, and BayesSim with quasi random Fourier Features. For the later we use the Matern 5/2 kernel [28] and set up the the sampling precision σ by cross validation. Three different simulators were used for different problems; OpenAI Gym [9], PyBullet [3], and MuJoCo [39]. Finally, the following problems were considered; CartPole (Gym), Pendulum (Gym), Mountain Car (Gym), Acrobot (Gym), Hopper (PyBullet), Fetch Push (MuJoCo) and Fetch Slide (MuJoCo). For all configurations of methods and parameters, training and testing were performed 5 times with the log probabilities averaged and standard deviation computed. To extract the real observations,

[2] https://github.com/araffin/rl-baselines-zoo

[3] https://pypi.org/project/pybullet/

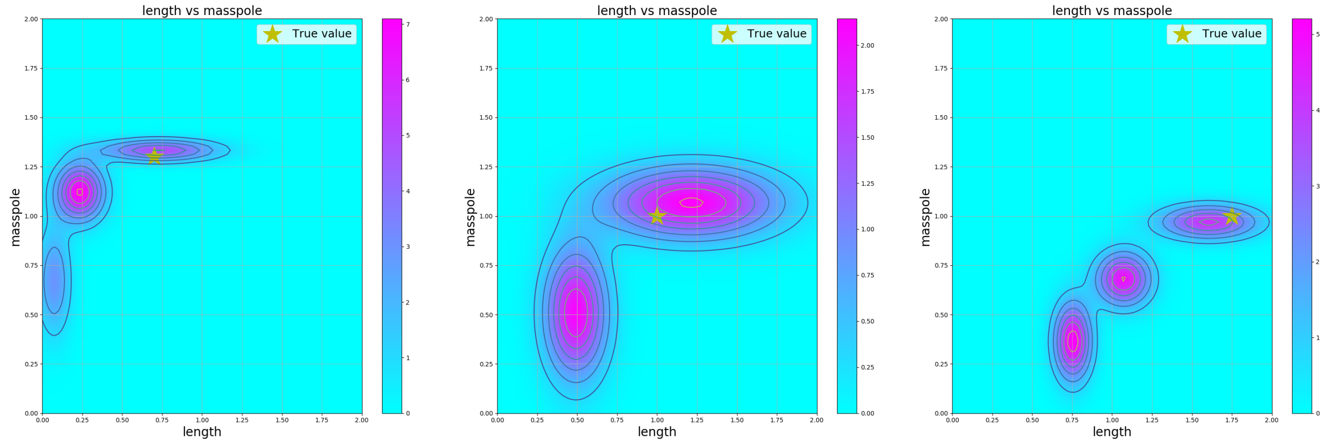

Fig. 2: Example of joint posteriors obtained for the CartPole problem with different parametrizations for `length` and `masspole`. The true value is indicated by a star. Note that the joint posteriors capture the multimodality of the problem when two or more explanations seem likely, for example, a longer pole `length` with a lighter `masspole` or vice versa.

we simulate the environments with the actual parameters 10 times and average the sufficient statistics to obtain $\mathbf{x}^r$. In all cases we collect sufficient statistics by performing rollouts for either a maximum of 200 time steps or until the end of the episode.

Table I shows the results (means and standard deviations) for the log probabilities. BayesSim with either RFF or Neural Network features provides generally higher log-probabilities and lower standard deviation than Rejection ABC. This indicates that the posteriors provided by BayesSim are more peaked and centered around the correct values for the parameters. Compared to ϵ-Free, the results are equivalent in terms of the means but BayesSim generally provides lower standard deviation across multiple runs of the method, indicating it is more stable than ϵ-Free. Comparing BayesSim with RFF and NN, the RFF features lead to higher log probabilities in most cases but BayesSim with neural networks have lower standard deviation.

These results suggest that BayesSim with either RFF or NN is comparable to the state-of-art, and in many cases superior when estimating the posterior distribution over the simulation parameters. For the robotics problems analyzed in the next section, however, BayesSim with RFF provide significant superior results than the other methods and slightly better than BayesSim with NN. This can be better observed when we plot the posteriors in Figure 3. BayesSim RFF is significantly more peaked and centered around the true friction value.

B. Robustness of policies

We evaluate robustness of policies by comparing their performance on the uniform prior and the learned posterior provided by BayesSim. Evaluation is done over a pre-defined range of simulator settings and the average reward is shown for each parameter value.

In the first set of experiments we use the CartPole problem as a simple example to illustrate the benefits of posterior randomization. We trained two policies, the first randomizing with a uniform prior for *length* and *masspole* as indicated in Table I. The second, randomized based on the posterior provided by BayesSim with RFF. In both cases we use PPO to train the policies with 100 samples from the prior and posterior, for 2M timesteps. The results are presented in Figure 4, averaged over several runs with the corresponding standard deviations. It can be observed that randomization over the posterior yields a significantly more robust policy, in particular at the actual parameter value. Also noticeable is the reduction in performance for lower *length* values and higher *masspole* values. This is expected as it is more difficult to control the pole position when the length is short due to the increased dynamics of the system. Similarly, when the mass increases too much, beyond the value it was actually trained on, the controller struggles to maintain the pole balanced. Importantly, the policy learned with the posterior seems much

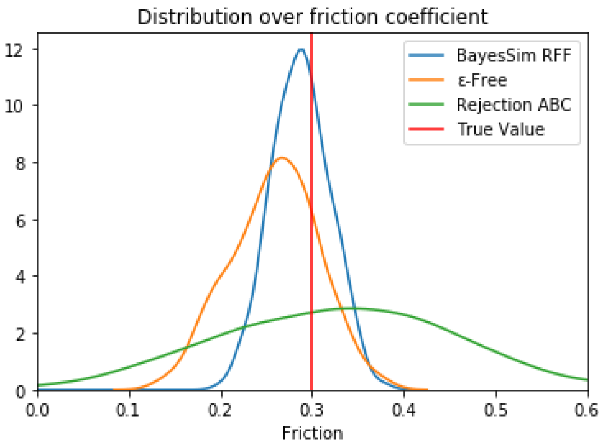

Fig. 3: Posteriors recovered by different methods for the Fetch slide problem. Note that BayesSim with random features provides a posterior that is more peaked around the true value.

Problem	Parameter	Uniform prior	Rejection ABC	ϵ-Free	BayesSim RFF	BayesSim NN
CartPole	pole length	$[0.1, 2.0]$	-0.342±0.15	**-0.211±0.07**	-0.609±0.39	-0.657±0.25
	pole mass	$[0.1, 2.0]$	0.032±0.21	0.056±0.14	**0.973 ± 0.26**	0.633± 0.52
Pendulum	dt	$[0.01, 0.3]$	2.101±1.04	2.307±0.84	3.192±0.30	**3.199±0.17**
Mountain Car	power	$[0.0005, 0.1]$	3.69±1.21	3.800±1.06	3.863±0.52	**3.901±0.2**
Acrobot	link mass 1	$[0.5, 2.0]$	1.704±0.82	1.883±0.79	**2.046±0.37**	1.331±0.22
	link mass 2	$[0.5, 2.0]$	1.832±0.93	**2.237±0.76**	0.321±1.85	1.513±0.39
	link length 1	$[0.1, 1.5]$	**2.421±0.75**	2.135±0.50	2.072±0.76	1.856±0.18
	link length 2	$[0.5, 1.5]$	-0.521±0.36	-0.703±0.16	**-0.148±0.19**	-0.672±0.09
Hopper	lateral friction	$[0.3, 0.5]$	3.032±0.43	3.154±0.81	2.622±0.64	**3.391±0.08**
Fetch Push	friction	$[0.1, 1.0]$	1.332±0.54	2.013±0.09	**2.423±0.07**	2.404±0.05
Fetch Slide	friction	$[0.1, 1.0]$	1.014±0.38	1.614±0.12	**2.391±0.06**	2.111±0.03

TABLE I: Mean and standard deviation of log predicted probabilities for several likelihood-free methods, applied to seven different problems and parameters.

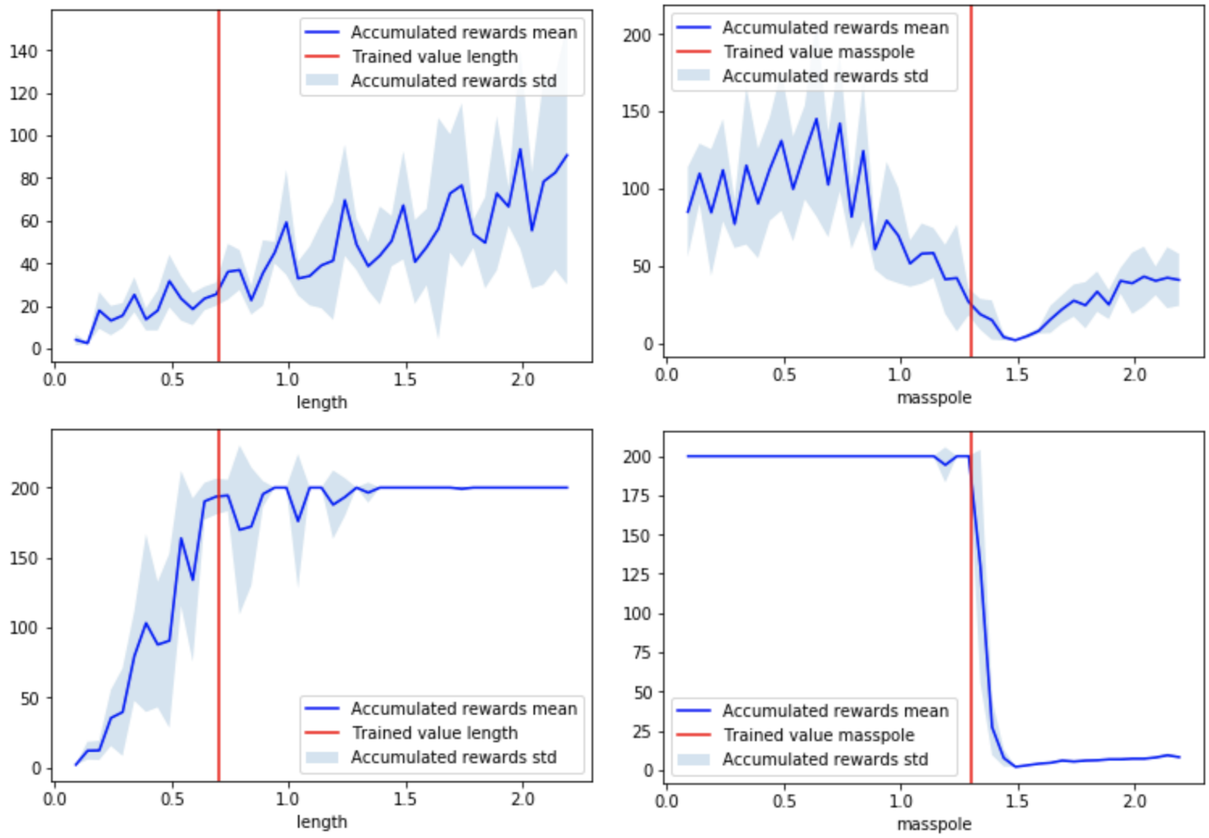

Fig. 4: Accumulated rewards for CartPole policies trained with PPO by randomizing over prior and posterior joint densities. Top left: Performance of the policy trained with the prior, over parameter `length`. `masspole` is set to actual. Top right: Similar to top left, but over multiple `masspole` values. Bottom left: Performance of policy trained with the posterior, over parameter `length`. Bottom right: Similar to bottom left, but over multiple `masspole` values.

more stable across multiple runs as indicated by the lower variance in the plots.

In the second set of experiments we use a Fetch robot available in OpenAI Gym [8] to perform both push and slide tasks. The first is a closed loop scenario, where the arm is always in range of the entire table and, hence, it can correct its trajectories according to the input it receives from the environment. The second is a more difficult open loop scenario, where the robot has usually only one shot at pushing the puck to its desired target. For both tasks, the friction coefficient of the object and the surface plays a major role in the final result as they are strictly related to how far the object goes after each force is applied. A very low friction coefficient means that the object is harder to control as it slides more easily and a very high one means that more force needs to be applied in order to make the object to move.

Our goal is to recover a good approximation of the posterior over friction coefficients using BayesSim. Initially, we need to learn a policy with a fixed friction coefficient that will be used for data generation purposes. We train this policy using

DDPG with experiences being sampled using HER for 200 epochs with 100 episodes/rollouts per epoch. Gradient updates are done using Adam with step size of 0.001. We then run this policy multiple times with different friction coefficients in order to approximate the likelihood function and recover the full posterior over simulation parameters. With the dynamics model in hand, we can finally recover the desired posterior using some data sampled from the environment we want to learn the dynamics from. Training is carried out using the same aforementioned settings but instead of using a fixed friction coefficient, we sample a new one from its respective distribution every time a new episode starts.

The results from both tasks are presented on Figure 5. As it has been shown in previously work [25], the uniform prior works remarkably well on the push task. This happens as the robot has the opportunity to correct its trajectory whether something goes wrong. As it has been exposed to a wide range of scenarios involving different dynamics, it can then use the input of the environment to perform corrective actions and still be able to achieve its goal. However, the results for slide task differ significantly since using a wide uniform prior has led the robot to achieve a very poor performance. This happens as not only the actions for different coefficients in most times are completely different but also because the robot has no option of correcting its trajectory. This is where methods like BayesSim are useful as it recovers a distribution with very high density around the true parameter and, hence, leads to a better overall control policy. Our results shows that higher rewards are achieved around the true friction value while the uniform prior results are mostly flat throughout all values.

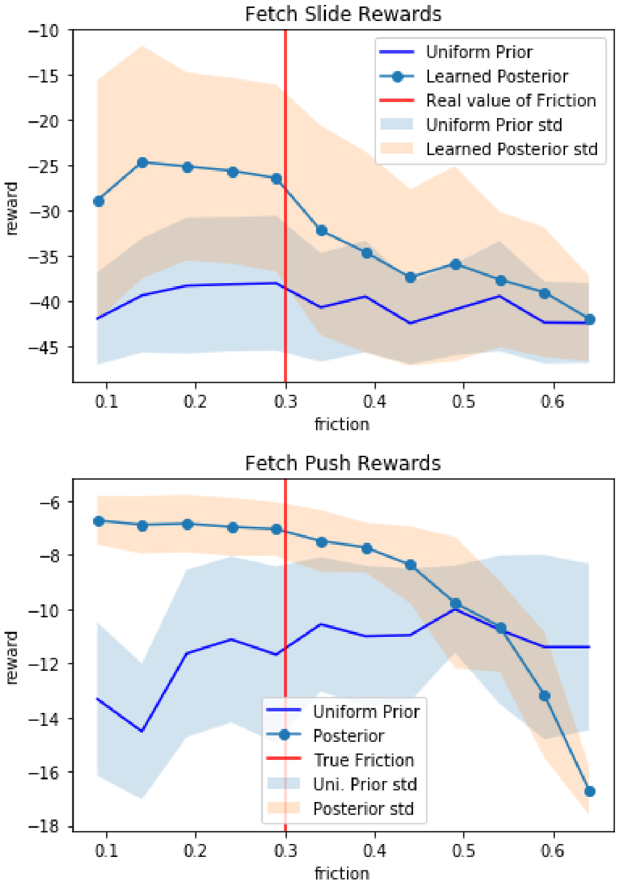

Fig. 5: Comparison between policies trained on randomizing the prior vs BayesSim posterior for different values of the simulation parameter. Top: Fetch slide problem. Bottom: Fetch push problem.

VI. Conclusions

This paper represents the first step towards a Bayesian treatment of robotics simulation parameters, combined with domain randomization for policy search. Our approach is connected to system identification in that both attempt to estimate dynamic models, but ours uses a black-box generative model, or simulator, totally integrated into the framework. Prior distributions can also be provided and incorporated into the model to compute a full, potentially multi-modal posterior over the parameters. The method proposed here, BayesSim, performs comparably to other state-of-the-art likelihood-free approaches for Bayesian inference but appears more stable to different initializations, and across multiple runs when recovering the true posterior. Finally, we show that domain randomization with the posterior leads to more robust policies over multiple parameter values compared to policies trained on uniform prior randomization.

The two applications described in the paper for likelihood-free inference are two instances of a large range of problems where simulators can make use of a full set of parametrizations to best represent reality. In this manner, our framework can be integrated in many other problems involving simulators. An interesting line of research for future work is to use BayesSim to help simulators synthesize images by randomizing over background properties. This can potentially help in making many computer vision problems more robust to environment variability in many tasks including object recognition, 3D pose estimation, or motion tracking.

As typical in the likelihood-free inference literature, BayesSim relies on the definition of meaningful sufficient statistics for the trajectories of states and actions. Alternatively, a lower dimensional representation for the trajectories could be created using recent encoder-decoder methods and recurrent neural networks known to perform well for time series prediction such as LSTMs [16]. Hence, the entire framework can be learnt end to end. This is an interesting area for future development but careful consideration should be given to potential overfitting to simulation data. LSTMs usually require a lot of data for training therefore most of the training trajectories will be generated from simulated trajectories. This can introduce undesirable specific characteristics of the simulator in the low dimensional representation that are not observed in real trajectories, making the representation less sensitive to variations in the simulator parameters to be estimated. Despite this, automating the process of generating robust statistics from trajectories remains a valuable direction for future research.

REFERENCES

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[2] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[3] Haim Avron, Vikas Sindhwani, Jiyan Yang, and Michael W. Mahoney. Quasi-Monte Carlo feature maps for shift-invariant kernels. *Journal of Machine Learning Research*, 17(120):1–38, 2016.

[4] Mark A Beaumont, Wenyang Zhang, and David J Balding. Approximate bayesian computation in population genetics. *Genetics*, 4(162):2025–2035, 2002.

[5] Christopher M Bishop. Mixture density networks. Technical report, Citeseer, 1994.

[6] Fernando V Bonassi, Mike West, et al. Sequential Monte Carlo with adaptive weights for approximate bayesian computation. *Bayesian Analysis*, 10(1):171–187, 2015.

[7] E. Braaten and G. Weller. An improved low-discrepancy sequence for multidimensional quasi-Monte Carlo integration. *Journal of Computational Physics*, 33:249–258, November 1979.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[10] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[11] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *arXiv preprint arXiv:1810.05687*, 2018.

[12] P.J. Diggle and R.J. Gratton. Monte Carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society. Series B (Methodological)*, 2 (46):193–227, 1984.

[13] I. I. Gihman and A. V. Skorohod. *The Theory of Stochastic Processes*, volume 1. Springer Verlag, Berlin, 1974.

[14] Graham C. Goodwin and Robert L. Payne. *Dynamic System Identification: Experiment Design and Data Analysis*. Academic Press, 1977.

[15] Christian Gourieroux, Alain Monfort, and Eric Renault. Indirect inference. *Journal of applied econometrics*, 8 (S1):S85–S118, 1993.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[17] Balakumar Sundaralingam Yu Xiang Dieter Fox Jonathan Tremblay, Thang To and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *Conference on Robot Learning (CoRL)*, 2018.

[18] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[19] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[20] Paul Marjoram, John Molitor, Vincent Plagnol, and Simon Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, 2003.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[22] Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5307–5314. IEEE, 2015.

[23] Lowrey K. Mordatch, I. and E. Todorov. Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*. IEEE, 2015.

[24] George Papamakarios and Iain Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. In *Advances in Neural Information Processing Systems*, pages 1028–1036, 2016.

[25] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[26] Jonathan K Pritchard, Mark T Seielstad, Anna Perez-Lezaun, and Marcus W Feldman. Population growth of human y chromosomes: a study of y chromosome microsatellites. *Molecular biology and evolution*, 16(12): 1791–1798, 1999.

[27] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural*

Information Processing Systems 20, pages 1177–1184. Curran Associates, Inc., 2008.

[28] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[29] Chad M Schafer and Peter E Freeman. Likelihood-free inference in cosmology: Potential for the estimation of luminosity functions. In *Statistical Challenges in Modern Astronomy V*, pages 3–19. Springer, 2012.

[30] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.

[31] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[32] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

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

[34] S.A. Sisson, Y. Fan, and M.M. Tanaka. Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765, 2007.

[35] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 843–852, Lille, France, 07–09 Jul 2015. PMLR.

[36] M. L. Stein. *Interpolation of Spatial Data*. Springer-Verlag, New york, 1999.

[37] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[38] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

[39] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[40] Darren J. Wilkinson. *Stochastic Modelling for Systems Biology, Second Edition*. Chapman & Hall/CRC Mathematical and Computational Biology. Taylor & Francis, 2011.