2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Online BayesSim for Combined Simulator Parameter Inference and Policy Improvement

Rafael Possas*† Lucas Barcelos† Rafael Oliveira† Dieter Fox*‡ Fabio Ramos*†
*NVIDIA †University of Sydney ‡University of Washington

***Abstract*— Recent advancements in Bayesian likelihood-free inference enables a probabilistic treatment for the problem of estimating simulation parameters and their uncertainty given sequences of observations. Domain randomization can be performed much more effectively when a posterior distribution provides the correct uncertainty over parameters in a simulated environment. In this paper, we study the integration of simulation parameter inference with both model-free reinforcement learning and model-based control in a novel sequential algorithm that alternates between learning a better estimation of parameters and improving the controller. This approach exploits the interdependence between the two problems to generate computational efficiencies and improved reliability when a black-box simulator is available. Experimental results suggest that both control strategies have better performance when compared to traditional domain randomization methods.**

I. INTRODUCTION

Advancements in simulation have allowed robotics learning to become more efficient and realistic in recent years. By creating accurate models of the real world one could train perception models and control policies entirely in simulation and have them seamlessly deployed in a real robot. However, there is still a range of possible improvements in simulation techniques before they can capture reality with all its complexities. "Reality gap" is a term used when the environment model used in a simulator does not represent the targeted system accurately enough so we can achieve the desirable performance when deploying a robot in the real world.

It is known that oversimplified assumptions or insufficient numerical precision in solvers can play a major role in how well a simulator models its target desired system. Existing prior knowledge about simulation parameters is often incorporated through a series of trial and error experiments until a good approximation is reached. This process is inefficient and time consuming as it involves running non-optimal control strategies on expensive and fragile robots.

Domain Randomization (DR) is a popular technique often used to mitigate the above problem. In this approach, a robot is trained on a wide range of simulations, each of them initialized with different parameters [1] and sampled from an uniform distribution over the entire space. On one hand, this process can be fast as it exploits parallel computation where each simulator can be executed simultaneously. On the other hand, its efficacy has been proven to work only on a handful of closed-loop robotics problems.

System identification is another way of addressing the above scenario. In this approach one can leverage real data to better estimate simulation parameters. However, traditional methods assume that the equations of the underlying dynamical system are known, limiting their use to simulators that expose their physics models.

In this work, we build upon the idea of using probabilistic inference to learn distributions over simulation parameters [2]. This technique leverages recent advances in likelihood-free inference (LFI) for Bayesian analysis to learn posteriors over simulation parameters based on rollouts obtained from the target system. Previous work [2] managed to learn distributions over parameters, but it required a reasonable initial controller that was able to explore the dynamical system in relevant regions of the state-space. Alternatively, in this paper we propose an end-to-end approach that combines posterior updates with controller improvement.

More formally, we provide the following contributions:

- We present an end-to-end approach for solving the Sim2Real problem where we tie together controller performance with simulation parameter inference
- We propose a sufficient-statistics-free method (SSLFI), where Recurrent Neural Networks (RNN) are used to automatically learn a latent representation of rollouts. This is a more scalable solution as it enables traditional Likelihood-Free Inference (LFI) in other data domains (e.g. images, videos and etc).
- We provide thorough experimental results on Sim2Sim and Sim2Real problems that shows the flexibility of our method.

II. RELATED WORK

Modern simulators have enabled a more efficient framework for developing new robotics methods. However, these simulators often make many assumptions where approximations are required to solve complex dynamics represented by differential equations. Fortunately, the increase in computational power has led this new wave of parallel simulation. A single simulation can be certainly inaccurate, but, multiple simulations, when combined, can capture reality in all its details. As seen in recent work [2][3], its generally understood that an ensemble of simulations can provide better approximation to the underlying complexity in a dynamical system .

Simulation parameters can vary from physical properties such as damping, friction and object masses [1] to visual parameters like textures, shapes and reflection [4]. The work of finding accurate models of the environment can be dated back to research in system identification [5][6]. Still,

978-1-7281-6211-9/20/$31.00 ©2020 IEEE

models trained on images from a non-realistic simulator will frequently fail when deployed to a real world environment.

Lately, Reinforcement Learning (RL) has been successfully used to train controllers in robotics. However, due to the stochastic nature of such algorithms, learned models are less robust to changes in simulation parameters. To overcome this problem, policies have been trained on distributions of simulators whose parameters are fit to real-world data [3]. In this work, the algorithm switches back and forth updating the domain randomization distribution by minimizing the difference between simulated and real world trajectories. Generalizations of these methods have also been attempted by varying simulation parameters in control tasks [7]. This helps to understand how simulation accuracy impacts control policies. Another possible approach, when the simulator is differentiable, is to use its gradients to make inference of physical properties. This has been used to learn such parameters on a rigid-body system [8].

Alternatively, Model Predictive Control (MPC) can also be used as a robust technique in control problems. Methods in this domain often seeks to iteratively find a solution of an optimisation problem for a receding finite time-horizon using an approximate model of the system [9]. In this way, sampling can be made more efficient if the internal model of the system reflects well the real environment where the robot is being deployed. Therefore, the use of distributions over simulator parameters can have major applications in several areas of robotics control.

Estimating accurate distributions of dynamical systems can enable robust policy learning in both open-loop and closed-loop scenarios [2]. In [10], the authors explore how using distributions over the model parameters can help training more robust model based controllers when compared to those using maximum-likelihood point estimates.

The use of Bayesian inference can be borrowed from more traditional statistics methods such as approximate Bayesian computation (ABC) [11]. Improvements over this method such as Rejection ABC [12], Markov Chain Monte Carlo ABC (MCMC-ABC) [13], Sequential Monte Carlo ABC (SMC-ABC) [14] and finally the ϵ-free approach [15] have enabled Bayesian inference on a wide range of problems. These methods are usually referred to as Likelihood-Free Inference (LFI) in traditional statistics literature. Their main advantage is that they can handle any problem where the simulation is a black-box generative model.

However, for LFI methods to work well, there is a strong dependency on manual specification of sufficient statistics representing trajectory roll-outs. In several scenarios, the data is either high dimensional (e.g. images, videos) or highly correlated (e.g. time series) which makes the definition of sufficient statistics challenging. Even state-of-the-art techniques [16] have still not addressed the issue of dealing with high dimensional input data. Attempts to use sufficient statistics in these cases will not provide a reasonable representation of simulation data which can drastically impact the overall performance. This, unfortunately, limits the usage of such methods to simpler time-series inputs. Our trajectory embedding technique with recurrent neural networks is an alternative to resolve some of these issues.

III. Likelihood-free Inference

In this section, we provide an overview on the use of likelihood-free inference (LFI) to overcome the "reality gap" in robotics simulators. We follow the derivation in [2] where black-box simulators with intractable likelihoods are incorporated into a learning method to estimate the posterior over simulation parameters. This can then be used to train robust policies with domain randomization that are capable of being deployed in a wider range of robotics tasks.

A simulator is a generative model usually parametrized by a vector of parameters $\boldsymbol{\theta}$. These parameters are used to generate an output vector $\mathbf{x}$ representing the behavior of a dynamical system. This process, implicitly, defines a likelihood function $p(\mathbf{x}|\boldsymbol{\theta})$ which we usually cannot evaluate but we can instead generate samples from. This is achieved by simply running the simulator; we hereby call this approach as forward modelling. However, one is actually interested on the inverse problem that maps real observations $\mathbf{x}^r$ back to the most likely parameters $\boldsymbol{\theta}$ that could have possibly generated them.

Given a black box generative model (e.g. simulator), where $\mathbf{x}^s = g(\boldsymbol{\theta})$ are the simulated observations and $\mathbf{x}^r$ are observations from the real world, the goal is to approximate a posterior $p(\boldsymbol{\theta}|\mathbf{x}^s, \mathbf{x}^r)$ with the likelihood function being implicitly defined by the generative process [17]. We assume that the simulator is a set of dynamical differential equations, typically solved using numerical or analytical solvers and often intractable and/or expensive to evaluate. Since there are no assumptions that equations are known, this type of approach can be used with black-box simulators.

A. BayesSim

As described in previous work [2], [15], recent LFI methods approximate the intractable posterior $p(\boldsymbol{\theta}|\mathbf{x} = \mathbf{x}^r)$ by directly learning a conditional density $q_\phi(\boldsymbol{\theta}|\mathbf{x})$ parameterized by parameters ϕ. This usually takes the form of a mixture of density network [18] where its outputs are the parameters of a Mixture of Gaussians. Parameters ϕ are learned in the following way: 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*. Observations $\mathbf{x}_n$ are generated by running the forward model using parameters $\boldsymbol{\theta}_n$ sampled from the chosen prior and collecting state-action pairs throughout the episode.

Then, we maximize the likelihood $\prod_n q_\phi(\boldsymbol{\theta}_n|\mathbf{x}_n)$ w.r.t. ϕ. In previous work [15], it has been shown that $q_\phi(\boldsymbol{\theta}|\mathbf{x})$ will be proportional to $\frac{\tilde{p}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}p(\boldsymbol{\theta}|\mathbf{x})$ if we optimize the log likelihood as follows:

$$\mathcal{L}(\phi) = \frac{1}{N}\sum_n \log q_\phi(\boldsymbol{\theta_n}|\mathbf{x}_n) \tag{1}$$

Subsequently, an estimate of the posterior can be obtained by:

$$\hat{p}(\boldsymbol{\theta}|\mathbf{x} = \mathbf{x}^r) \propto \frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})} q_\phi(\boldsymbol{\theta}|\mathbf{x} = \mathbf{x}^r), \tag{2}$$

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

The conditional density $q_\phi(\boldsymbol{\theta}|\mathbf{x})$ is a mixture of K Gaussians,

$$q_\phi(\boldsymbol{\theta}|\mathbf{x}) = \sum_k \alpha_k(\mathbf{x})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}(\mathbf{x})_k, \boldsymbol{\Sigma}_k(\mathbf{x})), \tag{3}$$

where $\{\alpha_k(\mathbf{x}\}_{k=1}^K$ are mixing functions, $\{\mu_k(\mathbf{x})\}_{k=1}^K$ are mean functions and $\{\Sigma_k(\mathbf{x})\}_{k=1}^K$ are covariance functions. These functions can be defined as linear combinations of Quasi Monte Carlo (QMC) random Fourier features [2] or neural networks as in [18]. The inputs $\mathbf{x}$ can be high-dimensional state-action trajectories or, for computational reasons, summary statistics calculated from trajectory data; e.g. cross-correlation, means and standard deviations. In this case we write $q_\phi(\boldsymbol{\theta}|\psi(\mathbf{x}))$ where $\psi(\cdot)$ is a function extracting the summary statistics from the trajectories.

IV. Domain Randomization

Domain Randomization is often used to alleviate the "reality gap" in over simplified simulation models and improve the performance of robotics control strategies. The goal is to create a variety of simulated environments with randomized properties and train a model that works across all of them. It is Likely that this model can adapt to the real-world environment as it is expected that the real system is one sample in that rich distribution of training variations. However, as we shall see, this solution only works on a limited set of problems. More specifically, the use of wide uniform distributions to sample simulation configurations only increases the complexity, for example, in learning RL policies for robotics applications.

Some recent control strategies with model free reinforcement learning have shown promising results in solving complex tasks. However, most methods still rely on good simulation environments to learn its optimal controllers. For instance, Deep Deterministic Policy Gradients [19] has been successful when applied to these problems by leveraging ideas from Deep-Q-Learning (DQN) [20]. Another set of policy search algorithms also suitable for continuous domains rely on the idea of trust region optimization. The exploration-exploitation trade-off is solved by limiting the maximum step size during updates. This is determined by its trust regions and its optimal point is then evaluated progressively until convergence. In this way, updates are always limited by their own trust region which results in a not only faster but more stable optimization process. Proximal Policy Optimization (PPO) [21] and Trust Region Policy Optimization (TRPO) have incorporated such ideas to provide state-of-the-art performance in a wide range of control problems. Some algorithms such as DDPG relies on storing an experience buffer and sampling from it during policy training, this is commonly known as off-policy learning. Techniques like Experience Replay [22], Prioritized Experience replay [23] and Hindsight Experience Replay (HER) [24] aim to make this sampling procedure better. The latter has been developed to improve performance on sparse RL problems.

Typically, on RL problems, an agent interacts with an environment $\mathbf{E}$ in discrete time steps. 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 continuous values $\mathbf{a}_t \in \mathbb{R}^D$ and environments are usually partially observed so that the entire history of observations, 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 $\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)$, parametrized by $\boldsymbol{\beta}$,

$$J(\boldsymbol{\beta}) = \mathbb{E}_{\boldsymbol{\eta}}\left[\sum_{t=0}^{T-1} \gamma^{(t)} r(\mathbf{s}_t, \mathbf{a}_t)|\boldsymbol{\beta}\right]. \tag{4}$$

Policy learning in RL can be quite challenging in a Domain Randomization setting. This is due to the already inherited unstable nature of RL algorithms. As it is shown in Equation 4, the loss function $J(\boldsymbol{\beta})$ is optimized over expectations of rewards weighted by $\gamma^{(t)}$. Therefore, updates on $\pi_\beta(\mathbf{a}_t|\mathbf{s}_t)$ not only follow gradients that are calculated over a noisy measurement but they also are biased since the policy being trained is the same one used to explore. This makes the "reality gap" problem even worse as the environment in which we collect data is constantly changing. Gradients can have completely different directions and magnitudes depending on the current simulation configuration and, while performing domain randomization over uniform priors [1], the "averaged out" gradient might not provide the robustness expected from such methods in some set of tasks.

Using LFI to solve the above problem follows the same core approach: at each rollout a parameter is sampled from a posterior $\hat{p}(\boldsymbol{\theta}|\mathbf{x} = \mathbf{x}^r)$ where we maximize the objective,

$$J(\boldsymbol{\beta}) = \mathbb{E}_{\boldsymbol{\theta}}\left[\mathbb{E}_{\boldsymbol{\eta}}\left[\sum_{t-0}^{T-1} \gamma^{(t)} r(\mathbf{s}_t, \mathbf{a}_t)|\boldsymbol{\beta}\right]\right], \tag{5}$$

where $\boldsymbol{\theta} \sim \hat{p}(\boldsymbol{\theta}|\mathbf{x} = \mathbf{x}^r)$ with respect to the policy parameters $\boldsymbol{\beta}$. The main difference is that the distribution is now learned from data which gives the desired density around true parameter values. This results in a more stable learning algorithm and efficient policy performance in the target environment.

Alternatively, model based methods rely on approximate system dynamics to evaluate future trajectories and compute the policy online, either by solving an optimisation problem as in iLQG [25], [26], or by sampling actions stochastically as in IT-MPC [9]. In either case, the first action of the computed trajectory is executed, a new observation gathered, and the remaining trajectory is re-optimised, a procedure known as Receding Horizon Control (RHC). As the control horizon gets larger, the importance of estimating the action-value of the terminal state diminishes, but we need to have more accurate dynamics [27]. Therefore, these methods can also benefit from domain randomization by incorporating uncertainty in the model parameters when evaluating future trajectories, such as in DISCO [10].

V. ONLINE BAYESSIM

Here we present the main contribution of the paper: Online BayesSim. We leverage previous work in likelihood-free inference to simultaneously improve a controller and learn a distribution over the simulator parameters. Additionally, we propose a methodology to automate the computation of a low-dimensional representation of state-action trajectories using Recurrent Neural Networks (RNN). The difficulty in representing high-dimensional time series has been one of the major reasons why LFI methods do not scale well to higher dimensional spaces. We show that with an RNN, latent representations from entire trajectories can be learnt and used directly for the posterior estimation. This removes the need to manually define meaningful summary statistics, which sometimes, can be a quite difficult and complex task.

A. Automatic Trajectory Embedding

In robotics control, trajectories of state-action pairs can be long sequences of correlated data, making the input dimensionality to the LFI framework prohibitively large and computationally costly. A common strategy inherited from traditional frameworks such as approximate Bayesian computation is to first extract sufficient statistics over the raw data generated by the simulator. This strategy has been proven successful in previous work [2], [15]. Although this approach seemed useful for simpler problems, it does not scale well to higher dimensional and more complex problems.

By introducing temporal models such as RNNs for automated sufficient statistics learning we achieve better scalability and automation on feature extraction from trajectory data. Let $\mathbf{z}$ be a latent representation defined as $\mathbf{z} = \psi_\gamma(\mathbf{S}, \mathbf{A})$ 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 and γ are the weights of a LSTM [28] denoted by $\psi_\gamma(\cdot)$. We input the latent embedding $\mathbf{z}$ of state and action pairs directly into the posterior estimator, $q_\phi(\boldsymbol{\theta}|\mathbf{z})$ and train both set of parameters γ and $\boldsymbol{\phi}$ jointly. Gradients are propagated throughout the entire network in a single back-propagation operation. From now on, whenever we refer to updates to the model $q_\phi(\boldsymbol{\theta}|\mathbf{z})$, we will be referring to updates in the parameters of both LSTM feature extractor γ and posterior estimator ϕ.

B. Sequential Learning

Recent work in domain randomization [2] requires policy optimization and dynamical model learning to happen in two separate steps. For the likelihood-free inference scenario, where a posterior is approximated through the observations collected in a simulated environment, we shall see that the controllers used to explore the state-action space are not required to be optimal. This allows us to learn both control strategies and environment models in a more iterative and principled way. Alternatively, we also show that the sequential nature of our algorithm can also be used online in MPC methods like [10] to update its internal model to reflect variations in the environment, e.g. adjust friction coefficients, compensate for extra weights added to the robot and etc.

In this work we follow an iterative process where at each step t we use the data generated on simulation configurations conditioned on the distribution over parameters $\boldsymbol{\theta}$ learned in the previous step $t-1$. Our approach can be seen as a domain randomization method where the distribution is updated during training by receiving feedback of the current interactions with the environment. We end up with controllers that not only have better performance but also are capable of adapting to changes in the environment.

Formally, we start with a stochastic controller $\pi_\beta(\boldsymbol{a_t}|\boldsymbol{s_t})$ and no prior knowledge of the true parameters represented by an uniform prior $p(\boldsymbol{\theta})$. In the first iteration $\pi_\beta(\boldsymbol{a_t}|\boldsymbol{s_t})$ is initialised with samples from the uniform prior $p(\boldsymbol{\theta})$. Trajectories $\mathbf{S}^s, \mathbf{A}^s$ are collected using current $\pi_\beta(\boldsymbol{a_t}|\boldsymbol{s_t})$ which are then used to update our Mixture of Gaussians model $q_\phi(\boldsymbol{\theta}|\mathbf{z})$. New data $\mathbf{S}^r, \mathbf{A}^r$ is then collected in the target system (e.g. real environment, proxy simulator and etc) using the same controller which is subsequently used to recover a new posterior and update the control strategy. $p(\boldsymbol{\theta}|\mathbf{S}, \mathbf{A} = \mathbf{S}^r, \mathbf{A}^r)$. The prior $p(\boldsymbol{\theta})$ is then replaced by the new posterior and the algorithm iterates until we achieve the desired controller performance. As we shall see in the experiments, there are two possible interpretations for Algorithm 1; When used on a RL setting, improving the controller means performing a gradient step in the direction that maximizes a given reward function. Alternatively in a MPC setup, the controller is improved by replacing its internal environment model to one that better represents the multi-modality of a physical world. A better representation of the environment for such methods means more efficient trajectory sampling that ultimately results in minimizing the involved cost function. In both scenarios, line 13 in the algorithm can be seen as improving our current control strategy to further collect data for our next step of posterior estimation. More details of the entire workflow can be seen in Algorithm 1.

VI. RESULTS

Method performance is evaluated on both Sim2Sim, where in some cases the second simulator has a different engine, and on a Sim2Real scenario where we use a Skid-Steer Robot. We show that very accurate posterior estimation is not entirely required to achieve state of the art control. As we shall see, in some cases an optimal controller can be trained without a fully converged dynamics model. This is an advantage of using full distributions for dynamics modelling instead of single point estimates.

We have chosen Pendulum, Cartpole, Acrobot and Hopper for simple control tasks. Sparse rewards robotics setups are also tested: both push and slide tasks (OpenAI Gym) performed by the fetch robot were evaluated for a more complex Reinforcement Learning problem. Lastly, we also evaluate the performance on a real robot using IT-MPC techniques where posteriors are updated in simulation with data collected from its real world counterpart and then used to replace the internal model of the controller in real time.

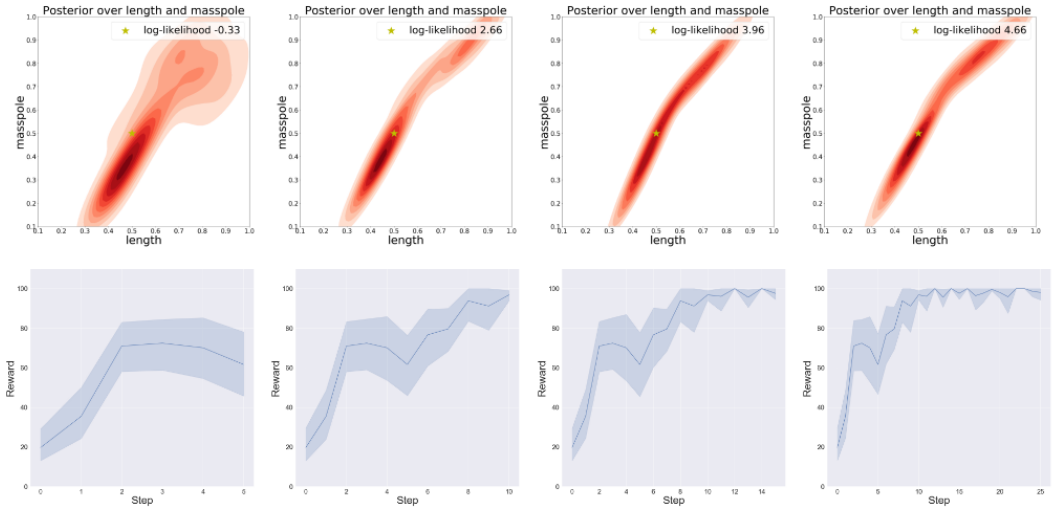

Fig. 1: Accumulated rewards and recovered posterior at steps 5, 10, 15 and 25 shows that a policy for PPO2 can be trained while we approximate the distribution over true parameters (length and masspole)

Algorithm 1 Online BayesSim

//observed and real trajectories: $\mathbf{S}^s, \mathbf{A}^s$ *and* $\mathbf{S}^r, \mathbf{A}^r$
//RNN Mixture of Gaussians (MoG) estimator: $q_\phi(\boldsymbol{\theta}|\mathbf{z})$
//RL Policy: $\pi_\beta(\boldsymbol{a_t}|\boldsymbol{s_t})$
Inputs: *total_steps*, *policy_train_steps*, *mog_train_steps*, *num_sampled_params*, $p(\boldsymbol{\theta}_0)$
Outputs: $q_\phi(\boldsymbol{\theta}|\mathbf{z}), \pi_\beta(\boldsymbol{a_t}|\boldsymbol{s_t})$

Initialize weights $\boldsymbol{\beta}$ and $\boldsymbol{\phi}$ randomly

$t \leftarrow 1$
repeat
  $\boldsymbol{\theta}_t \sim p(\boldsymbol{\theta}_{t-1})$
  $\mathbf{S}^s, \mathbf{A}^s \leftarrow$ Run $\pi_{\beta_t}(\boldsymbol{a_t}|\boldsymbol{s_t})$ in sim with $\boldsymbol{\theta}_t$.
  $\boldsymbol{\beta}_t \leftarrow \boldsymbol{\beta}_{t-1} + \lambda \nabla \pi_{\beta_t}(\boldsymbol{a_t}|\boldsymbol{s_t})$
  $\phi_t \leftarrow \phi_{t-1} + \lambda \nabla q_\phi(\boldsymbol{\theta}_t|\psi_{\gamma_t}(\mathbf{S}^s, \mathbf{A}^s))$
  $\mathbf{S}^r, \mathbf{A}^r \leftarrow$ Run $\pi_{\beta_t}(\boldsymbol{a_t}|\boldsymbol{s_t})$ on real env.
  $p(\boldsymbol{\theta}_t|\mathbf{S}, \mathbf{A}) \leftarrow q_\phi(\boldsymbol{\theta}_t|\psi_{\gamma_t}(\mathbf{S}^r, \mathbf{A}^r))$
  $p(\boldsymbol{\theta}_t) \leftarrow p(\boldsymbol{\theta}_t|\mathbf{S}, \mathbf{A})$
  $t \leftarrow t + 1$
until $t < total\_steps$ or convergence reached

For posterior estimation benchmarking, we compare our sufficient-statistics free posterior estimation with both ϵ-Free [15] and RFF (Random Fourier Features) [2] methods. On the real robot setup, we show that our work can assist IT-MPC methods like DISCO [10] to react to changes in the environment by performing iterative and real time posterior updates. For all experiments we have collected data from multiple attempts in order to better capture the variance of different initializations.

A. Classic control tasks

While in the Pendulum, Cartpole and Acrobot we use different instances of the same simulation, in the Hopper problem we use different simulation engines to evaluate our method. In the latter, the "real world" is represented by MuJoCo while the controller is trained entirely on PyBullet. For all RL policies we have used the stable-baselines framework implementation with their default parameters. For posterior estimation the mixture of gaussians has been chosen to have 5 components to allow the recovery of multimodal distributions.

On each inner-loop step of Online BayesSim we trained a PPO2 (Cartpole, Acrobot) [21] and SAC (Pendulum) [29] policy $\pi_{\boldsymbol{\beta}}(a_t|s_t)$ for 300 steps where each episode has 50 steps and $q_\phi(\boldsymbol{\theta}|\mathbf{z})$ for 200 epochs with 200 trajectories generated by the current policy. In both tasks, there is no initial training from the uniform prior, the policy used in the first iteration has random weights. We run the outer-loop for a total of 30 epochs and we achieve state of the art log-likelihood on posterior estimation for all tasks while also achieving maximum reward with the learned controller. Results on figure 1 shows that the same multi-modality recovered in previous work [2] is very similar to the one achieved in here. However, the distribution is more peaked due to the higher log-likelihood as depicted on table I. In the Hopper problem, we train a PPO2 policy with the same configuration as above but the only difference is that we run the outer-loop for longer (50 epochs). Although highest log-likelihood value is not achieved in this problem, values are close enough to consider successful transfer between two different simulation engines. It is worth to note that previous work has evaluated the same problem using the same engine on both ends. Our evaluation is more challenging as each simulation engine have specific ways of implementing their differential equations.

Lastly, the highest difference in posterior evaluation can be seen on the Acrobot task. As we increase the dimensionality of the parameters being estimated, methods with sufficient statistics will not capture the correct latent space of state-

Problem	Parameter	Prior	Online BayesSim	BayesSim RFF	ϵ-Free
CartPole	Length / Mass	$[0.1, 1.0]$	**4.66±0.22**	2.68±0.08	2.88±0.15
Pendulum	Length / Mass	$[0.1, 1.0]$	**4.076±0.12**	3.89±0.34	3.332±0.41
Fetch Push	Friction	$[0.1, 2.0]$	2.09±0.12	**2.18±0.19**	2.10±0.27
Fetch Slide	Friction	$[0.1, 2.0]$	**3.24±0.21**	3.12±0.08	2.55±0.26
Hopper	Lat. Friction	$[0.1, 0.5]$	3.25±0.33	3.134±0.11	**3.384±0.25**
Acrobot	Link Mass 1 & 2	$[0.5, 2.0]$	**2.85±0.12**	1.534±0.22	1.210±0.32
	Link Length 1 & 2	$[0.1, 1.5]$	**2.25±0.13**	1.426±0.11	1.012±0.21

TABLE I: Mean and standard deviation of log-likelihood of the joint distribution for offline and online likelihood-free methods, applied to different problems and combination of parameters

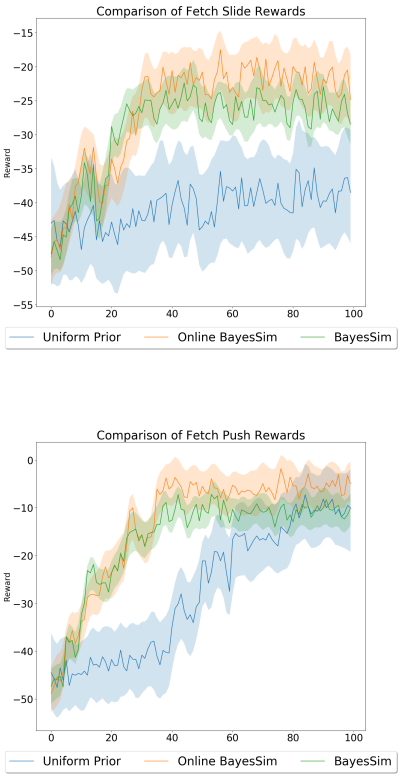

Fig. 2: (Top) Comparison of different DR techniques in a open loop task shows that Online BayesSim outperforms its offline counterpart while uniform DR does not seem to work. (Bottom) Comparison on a Closed Loop task, uniform DR seems to work better but still would require more steps to reach the same reward as the two other methods.

action pairs. This shows that using RNNs for automatic trajectory embedding works better on problems where we estimate a higher number of simulation parameters.

B. Sparse reward robotics tasks

Both experiments with the Fetch robot required a different setup from the ones above. While in the classical control tasks we start with a completely random controller, in this set of experiments we have to train the initial controller on a uniform prior for 10 epochs before we start the main method loop. This happens as we have to firstly learn basic kinematics to then be able to cause the perturbation on the state space that is required to learn the dynamics model.

In order to solve the sparse reward problem of both tasks we have used DDPG coupled with Hindsight Experience Replay. In the first iteration we train the policy on uniform prior for 15 epochs and for the remaining of the loop we train the controller for 5 epochs before fitting a new posterior. In order to work well, we were also required to reduce the size of the HER replay buffer. Since the replay buffer stores past experiences and our simulator is changing at every step, we require a buffer that is able to replace past experiences quickly so as to accommodate the constant changes in simulation.

Results depicted on Figure 2 shows that higher reward is achieved for both experiments when compared to different domain randomization techniques. While on the push task the uniform domain randomization is able to achieve similar reward at the end of training when compared to other methods, in the fetch slide there is no visible reward increase. The log-likelihood presented on Table I shows that our method outperforms previous work on the more challenging slide task while reaches similar log-likelihood on the push task with lower variance and higher overall reward. Variance in rewards for both experiments is higher as we use non-converged dynamics distributions along the way, therefore it causes the robot to experiment more uncertainty while optimizing its controller.

C. Experiments on a physical robot

This section presents experimental results with a physical robot equipped with a skid-steering drive mechanism (Figure 3). We modelled the kinematics of the robot based on a modified unicycle model, which accounts for skidding via an additional parameter [30]. The parameters to be estimated via Online BayesSim are the robot's wheel radius, axial distance, i.e. the distance between the wheels, and the displacement of the robot's instant centre of rotation (ICR) from the robot's centre. A non-zero value on the latter affects turning by sliding the robot sideways. The control task was defined as following a circular path at a constant tangential speed. Costs were set to make the robot follow a circle of 0.75 m radius with $c(\mathbf{s}_t) = \sqrt{d_t^2 + (v_t - v_0)^2}$, where d_t represents the robot's distance to the edge of the circle and $v_0 = 0.2$ m/s is a reference linear speed.

We have used DISCO [10] as the robot's controller, a stochastic non-linear MPC based on MPPI [9]. Such method

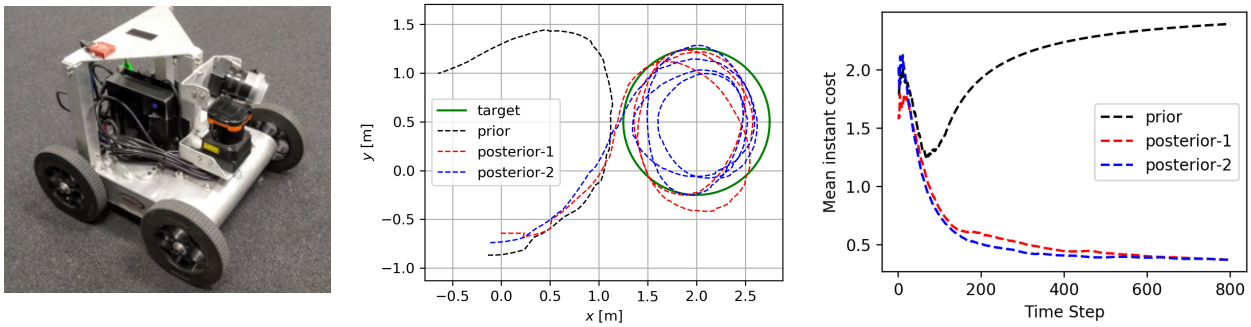

Fig. 3: (Left) Skid-steer Robot. (Center) As posteriors are refined the controller has fewer overshoots on the circular trajectory. (Right) Cumulative average of the cost over time.

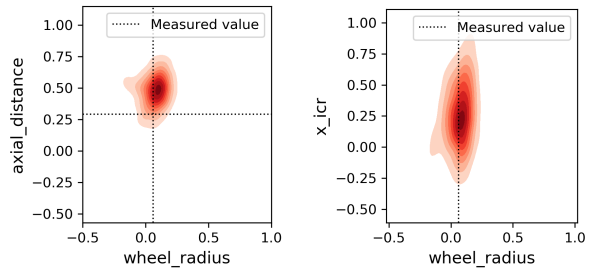

Fig. 4: Posterior distribution for the *first* iteration of online BayesSim on the experiments with a physical robot. Available measured values are indicated by a dashed line.

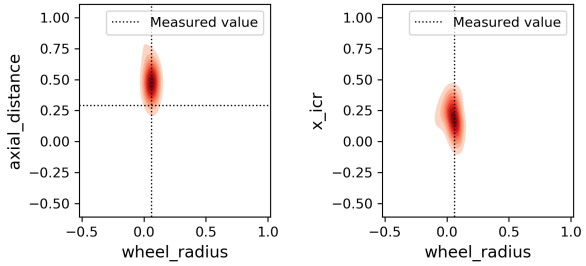

Fig. 5: Posterior distribution for the *second* iteration of online BayesSim on the experiments with a physical robot. Available measured values are indicated by a dashed line.

considers the uncertainty in the system's parameters and, hence, can be used together with the posteriors recovered in each step of Online BayesSim. The main advantage of using Online BayesSim instead of its offline counterpart is that we can continuously update the internal model of the controller to reflect changes in the environment in real time. As the robot experiences different scenarios, the data is then incorporated in the main loop where multimodal posteriors can arise naturally to reflect changes such as increase/decrease in robot weight, different friction coefficients and etc.

In order to evaluate our method in the above task, we firstly initialize the controller with a wide uniform prior and iteratively update it with trajectory data through BayesSim.

Sampling distribution for θ	Mean cost
Prior	2.39 ± 0.48
Posterior after first iteration	0.37 ± 0.35
Posterior after second iteration	$\mathbf{0.36 \pm 0.32}$

TABLE II: Average task cost for the robotic experiment with different types of sampling distribution for the model parameters.

The resulting posterior is then updated in the controller for the next batch of data collection. This process is repeated until there are no considerable changes in the estimated posterior or reasonable performance is achieved by the controller.

The results presented in Figure 3 and Table II show the qualitative and quantitative improvement in the control task as the posterior distribution is refined. As expected, once the robot is able to collect data from the real environment and refine its knowledge of the world, the results improve significantly. After that, the posterior distribution keeps adjusting to the idiosyncrasies of the environment, as reflected by the further improvement, albeit not as substantially. The main benefit of this setup, however, is the fact that refining the posterior distribution provides a way of adjusting to the non-stationary characteristic of the environment with a fast response to the sudden changes.

Figure 4 presents the first posterior, which was estimated from data obtained by running DISCO with the uniform prior. The second posterior estimated by running DISCO with the previous posterior is shown in Figure 5. As evident from the plots, the second posterior distribution is more concentrated, providing a more informative guess of the true physical parameters to the controller. Another evidence to highlight is that, although one can measure the wheel radius and the axial distance without many difficulties, the parameter indicating the robot's ICR is hard to measure, involving an elaborate weighing process. The estimate from Online BayesSim then again highlights the usefulness of Bayesian inference for robotic problems with unknown physical parameters.

VII. Discussion

This paper presented a principled framework for solving the "reality gap" problem in robotics simulators, combining parameter estimation with policy improvement. The approach is capable of leveraging these two problems within a single framework, where sequential improvements in controller performance are used to estimate better simulation parameters and the associated uncertainty. Online BayesSim, as seen in table I and figure 2, outperforms previous work in terms of accumulated reward and quality of posterior. It is also more general as it does not assume an initial optimal controller. Moreover, we provide a method that connects traditional statistics literature with more recent ideas on learning latent representations for dynamical systems with recurrent neural networks. Lastly, we perform experiments on a real robot and we show how our method can be used to improve environmental beliefs in real time through our iterative posterior distribution refinement.The combination of Bayesian inference for black-box generative models and policy search opens a myriad of possibilities for applications in robotics when a simulator is available and a controller needs to be optimized for further deployment into a real system. The automatic trajectory embedding method can be further extended to receive sequences of images as inputs, leveraging powerful simulators with realistic graphics capabilities. In this manner, even appearance parameters such as textures and illumination could be estimated from sequences of images. Finally, we highlight the importance of treating the estimation of simulation parameters from a probabilistic point of view since inferring posterior distributions as opposed to point estimates can lead to more robust, safe and reliable controllers.

References

[1] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[2] F. Ramos, R. C. Possas, and D. Fox, "Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators," *arXiv preprint arXiv:1906.01728*, 2019.

[3] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," *arXiv preprint arXiv:1810.05687*, 2018.

[4] 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 *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 23–30.

[5] L. Ljung and T. Söderström, *Theory and practice of recursive identification*. MIT press, 1983.

[6] F. Giri and E.-W. Bai, *Block-oriented nonlinear system identification*. Springer, 2010, vol. 1.

[7] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song, "Assessing generalization in deep reinforcement learning," *arXiv preprint arXiv:1810.12282*, 2018.

[8] E. Heiden, D. Millard, H. Zhang, and G. S. Sukhatme, "Interactive differentiable simulation," 2019.

[9] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving," vol. 34, no. 6, pp. 1603–1622.

[10] L. Barcelos, R. Oliveira, R. Possas, L. Ott, and F. Ramos, "DISCO: Double Likelihood-free Inference Stochastic Control." [Online]. Available: http://arxiv.org/abs/2002.07379

[11] M. A. Beaumont, W. Zhang, and D. J. Balding, "Approximate bayesian computation in population genetics," *Genetics*, vol. 4, no. 162, pp. 2025–2035, 2002.

[12] J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman, "Population growth of human y chromosomes: a study of y chromosome microsatellites." *Molecular biology and evolution*, vol. 16, no. 12, pp. 1791–1798, 1999.

[13] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré, "Markov chain Monte Carlo without likelihoods," *Proceedings of the National Academy of Sciences*, vol. 100, no. 26, pp. 15 324–15 328, 2003.

[14] F. V. Bonassi, M. West, *et al.*, "Sequential Monte Carlo with adaptive weights for approximate bayesian computation," *Bayesian Analysis*, vol. 10, no. 1, pp. 171–187, 2015.

[15] G. Papamakarios and I. Murray, "Fast ε-free inference of simulation models with bayesian conditional density estimation," in *Advances in Neural Information Processing Systems*, 2016, pp. 1028–1036.

[16] G. Papamakarios, D. C. Sterratt, and I. Murray, "Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows," *arXiv preprint arXiv:1805.07226*, 2018.

[17] P. Diggle and R. Gratton, "Monte Carlo methods of inference for implicit statistical models," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 2, no. 46, pp. 193–227, 1984.

[18] C. M. Bishop, "Mixture density networks," Citeseer, Tech. Rep., 1994.

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

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

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

[22] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[23] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[24] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[25] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 292–299.

[26] E. Todorov and Weiwei Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.* IEEE, pp. 300–306. [Online]. Available: http://ieeexplore.ieee.org/document/1469949/

[27] B. Recht, "A Tour of Reinforcement Learning: The View from Continuous Control," vol. 2, no. 1, pp. annurev–control–053 018–023 825. [Online]. Available: https://www.annualreviews.org/doi/10.1146/annurev-control-053018-023825

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[29] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[30] K. Kozłowski and D. Pazderski, "Modeling and Control of a 4-wheel Skid-steering Mobile Robot," *Int. J. Appl. Math. Comput. Sci.*, vol. 14, no. 4, pp. 477–496, 2004.