Safe Trajectory Sampling in Model-based Reinforcement Learning

Sicelukwanda Zwane[1], Denis Hadjivelichkov[1], Yicheng Luo[1],
Yasemin Bekiroglu[1,2], Dimitrios Kanoulas[1], and Marc Peter Deisenroth[1]

***Abstract*— Model-based reinforcement learning aims to learn a policy to solve a target task by leveraging a learned dynamics model. This approach, paired with principled handling of uncertainty allows for data-efficient policy learning in robotics. However, the physical environment has feasibility and safety constraints that need to be incorporated into the policy before it is safe to execute on a real robot. In this work, we study how to enforce the aforementioned constraints in the context of model-based reinforcement learning with probabilistic dynamics models. In particular, we investigate how trajectories sampled from the learned dynamics model can be used on a real robot, while fulfilling user-specified safety requirements. We present a model-based reinforcement learning approach using Gaussian processes where safety constraints are taken into account without simplifying Gaussian assumptions on the predictive state distributions. We evaluate the proposed approach on different continuous control tasks with varying complexity and demonstrate how our safe trajectory-sampling approach can be directly used on a real robot without violating safety constraints.**

I. Introduction

Motion planning in robotics has been studied extensively, and a variety of approaches have been proposed based on, e.g. sampling [1], optimization [2], and probabilistic inference [3]. In this setting, nonparametric learning based methods such as Gaussian processes (GPs), have been shown to generate smooth trajectories without compromising accuracy [4] while providing a principled way of modeling uncertainties associated with operating on physical robot systems. When the system dynamics model is unknown/partially known, the motion planning problem can be addressed using a reinforcement learning (RL) approach. RL based methods have been utilized for robot control and motion planning [5] coping with nonlinearities and presenting flexibility in modeling a variety of task oriented skills. However, applying RL algorithms to robotic arm trajectory generation while avoiding obstacles and abiding by constraints is a data-intensive process. Improving training efficiency and safety of RL algorithms on physical systems is an active research area.

Model-based RL (MBRL) is a particularly promising approach for robotics because it addresses limitations that arise with data collection [6]. Unlike model-free RL algorithms [7], MBRL can alleviate this problem and provide better data-efficiency, as it exploits a model of the system's dynamics to generate trajectories for training the policy [8]. However, if the model does not sufficiently capture the true underlying dynamics of the robot, the learned policy may be rendered infeasible when applied to the real system [8]. This is one of the main reasons why studies of RL in robotics applications are often limited to simulated environments. Therefore, it is important to account for potential model errors by using probabilistic models [9], [8] that explicitly describe uncertainty about the model's parameters (epistemic uncertainty).

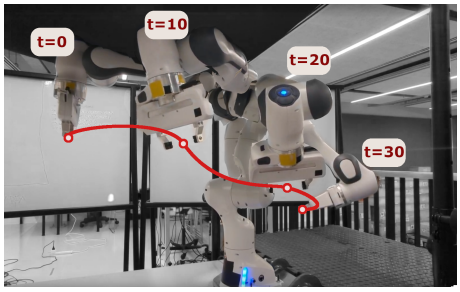


Fig. 1: A learned safe policy execution on a Panda robot without violating constraints, i.e., the low-hanging ceiling shown in black. The example trajectory is represented by the red line with configurations at time steps $t = \{0, 10, 20, 30\}$. The target pose is reached at $t = 30$.

The application of RL approaches to real robotic systems (Figure 1) is challenging because physical systems often require consideration of safety or feasibility constraints [10], which classical RL approaches do not explicitly model [11]. For example, picking and placing a mug with a robot manipulator requires monitoring the physical limits of the robot's joints, collisions between the robot and itself, collisions between the robot and other objects in the scene, and collisions between the mug and the robot. In this example, practitioners also need to consider additional, task-specific safety constraints, such as keeping the orientation of the mug upright so as to not spill its liquid contents. Unfortunately, the trial-and-error nature of RL means the learned policy has to experience constraint violations for such problems before learning to avoid them. Furthermore, robotics is one such domain where constraint violations should be minimized, or in some contexts [12], avoided at all costs.

In this work, we incorporate safety constraints into MBRL

[1]All authors are with the UCL Centre for Artificial Intelligence, University College London, UK. [2]Y. Bekiroglu is also with the Department of Electrical Engineering, Chalmers University of Technology, Sweden. Corresponding author email: sicelukwanda.zwane.20@ucl.ac.uk.

This work was partially supported by the the Engineering and Physical Sciences Research Council (EPSRC) [EP/S021566/1] and the UKRI Future Leaders Fellowship [MR/V025333/1] (RoboHike).

in order to minimize constraint violations during the training phase. More specifically, we check whether predictions of a probabilistic model violate safety constraints. We rely on a GP formulation as it provides data-efficiency and principled way of encoding uncertainty. Previous work on safe RL with GPs formulates the problem as a constrained optimization problem, where authors used moment-matching for long-term predictions [13]. The moment-matching approach, however, makes Gaussian assumptions about the predictive state distributions. At the same time, a sample trajectory to be executed by the robot will ignore time correlations between consecutive states [9]. We propose to use a trajectory-sampling approach that directly generates sample trajectories from the GP without making Gaussian assumptions on the predictive state marginals, obeys time correlations between consecutive states, and can be directly executed on the robot, as seen in Figure 1 with an example trajectory.

In summary, the main contributions of the paper are as follows: i) We propose a novel MBRL approach that takes into account safety constraints without making assumptions about underlying dynamical system characteristics. ii) The proposed method preserves temporal correlations in the sampled trajectories used in the MBRL policy training. We provide a comparison between two approaches for incorporating constraints across two complex continuous control tasks. iii) We show *sim-to-real* experiments for a constrained reaching task where a safe policy is learned successfully within 100 seconds of data.

II. Policy Search with Probabilistic Dynamics Models

We consider robotic systems, where the dynamics evolve over time according to

$$\boldsymbol{x}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}), \tag{1}$$

where $f : \mathbb{R}^D \times \mathbb{R}^U \to \mathbb{R}^D$ is an unknown transition function that governs how the system's state $\boldsymbol{x}_t \in \mathbb{R}^D$ evolves over time when a control action $\boldsymbol{u}_t \in \mathbb{R}^U$ is applied. Given a reward function $R : \mathbb{R}^D \to \mathbb{R}$, we are concerned with finding a state-feedback controller (parameterized policy) $\pi_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^U$ whose parameters $\boldsymbol{\theta}$ maximize the objective

$$J(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}\left[R(\boldsymbol{x}_t) \mid \boldsymbol{\theta}\right], \tag{2}$$

where $J(\boldsymbol{\theta})$ is the expected long-term return that measures how good a policy is as the expected sum of step-wise rewards R over a finite-horizon state trajectory $\boldsymbol{\tau} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$.

We assume the true system dynamics f are unknown or difficult to express mathematically. MBRL is a class of RL algorithms that address this issue by learning an approximate dynamics model $f_{\boldsymbol{\psi}}$ with parameters $\boldsymbol{\psi}$, which is then used for internal simulations of the real system. These simulations are used for training policy $\pi_{\boldsymbol{\theta}}$.

To cope with an incorrect model $f_{\boldsymbol{\psi}}$, e.g., due to insufficient data, we need to learn a probabilistic model that explicitly expresses uncertainty about the model parameters $\boldsymbol{\psi}$ [9], [8]. Probabilistic dynamics models have been applied successfully in robotics and have been shown to reduce model-misspecification errors. This includes locally-weighted regression [9], [14], probabilistic neural networks [15], and GPs [8]. In general, we can use a dynamics model for MBRL in one of two ways; 1) to generate predictions online for model-predictive control [13], 2) to make long-term predictions for training a global policy in model-based policy search [16]. In this paper, we focus on the latter case within the PILCO framework [17]. Here, a GP $f_{\boldsymbol{\psi}}$ is used as a model for the transition function f. In the original paper, PILCO uses moment-matching for generating long-term predictions of the state evolution, which are then used to produce an estimate of the expected long-term reward in Equation (2). Based on this reward estimate, the policy parameters $\boldsymbol{\theta}$ are optimized. Note that the PILCO framework does not consider the case of (state) safety constraints, which we consider.

III. Gaussian Process Dynamics Models

A Gaussian process (GP) is a distribution over functions $f : \mathcal{X} \to \mathcal{Y}$, and they are fully defined by a mean function $m(\cdot)$ and a covariance function (kernel) $k(\cdot, \cdot)$. If not stated otherwise, we assume $m \equiv 0$ throughout the paper. The kernel k controls the smoothness, differentiability, and variance properties of the GP. We consider the RBF kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Lambda}^{-1}(\boldsymbol{x} - \boldsymbol{x}')), \tag{3}$$

where $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^D$. Here, σ_f^2 is the variance of the function modelled and $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1^2, \ldots, \lambda_D^2)$ is a diagonal matrix of (squared) length-scales λ_d. The parameters σ_f and λ_d of the kernel are the GP's hyper-parameters $\boldsymbol{\psi}$, which we optimize by maximizing the log-marginal likelihood

$$\begin{aligned} \log p(\boldsymbol{y} \mid \boldsymbol{X}) = &-\frac{1}{2}\boldsymbol{y}^T \left(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} \\ &- \frac{1}{2}\log|\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I}| - \frac{N}{2}\log 2\pi, \end{aligned} \tag{4}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^T$ and $\boldsymbol{y} = [y_1, \ldots, y_N]^T$ are the N training inputs and targets, respectively.

The posterior predictive distribution of the GP at test location $\boldsymbol{x}_*$ is Gaussian with mean and variance given by

$$\boldsymbol{\mu}_* = \boldsymbol{k}_*^T \left(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} \tag{5}$$

$$\boldsymbol{\Sigma}_* = \boldsymbol{k}_{**} - \boldsymbol{k}_*^T \left(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}_*, \tag{6}$$

where $\boldsymbol{k}_{**} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$, $\boldsymbol{k}_* = k(\boldsymbol{X}, \boldsymbol{x}_*)$, $\boldsymbol{K} = k(\boldsymbol{X}, \boldsymbol{X})$, and $\sigma_\varepsilon^2 = 10^{-4}$ is a "nugget" for numerical stability.

To learn a dynamics model f that describes the transition dynamics $p(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$ for RL, we train a GP using Equation (4) on one-step transitions $((\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}), \boldsymbol{x}_t)$, mapping each state-action pair $(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$ to next states $\boldsymbol{x}_t$. To make GP predictions at test locations $(\boldsymbol{x}_{*,t-1}, \boldsymbol{u}_{*,t-1})$, we evaluate Equation (5)–Equation (6). These equations are a distributional representation where the GP posterior is specified using its distributional parameters. In later sections, we discuss a pathwise (sample-based) representation and highlight its advantages in this setting.

GP dynamics models have sample-efficiency and smoothness properties, so that they can be applied to real-world robotic systems [8].

A. Long-term Predictions

In episodic task settings, the evaluation of the expected return $J(\boldsymbol{\theta})$ for any policy $\pi_{\boldsymbol{\theta}}$, requires us to compute the predictive marginal state distributions $p(\boldsymbol{x}_1), \dots, p(\boldsymbol{x}_T)$. In the context of GP dynamics models, these marginals cannot be computed exactly due to the nonlinearity of the GP, so that we need to resort to approximations. In [17], [13], the authors use a moment-matching approximation, where the mean $\boldsymbol{\mu}_t$ and covariance $\boldsymbol{\Sigma}_t$ of every state marginal $p(\boldsymbol{x}_t)$ is computed. These computations can be done analytically for a suitable choice of kernel functions, such as the RBF kernel in Equation (3). Alternative approaches to computing the desired moments have also been proposed, e.g., by means of linearisation of the GP [8], where the assumptions on the kernel are much less restrictive. In [18], the authors determine the means and covariances of the state marginals as Monte Carlo estimates within a sampling setting. In the context of neural network ensembles, it is possible to determine the moments of the state marginals as Monte Carlo estimates [15].

To generate a *realisation* of a trajectory from these predictive state marginals, it is possible to individually sample states $\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. However, a trajectory generated in this way is "disconnected" because it does not account for temporal correlations, making it unsafe for robot execution. The moment-matching distributional representation assumes that the state marginal distributions $p(\boldsymbol{x}_t)$ are Gaussian. This assumption may hinder our ability to learn meaningful policies in complex environments where the state marginals can be multi-modal and highly non-Gaussian. Wherever possible, we must relax our modeling assumptions to improve the chances of executing trajectories sampled from learned models on physical hardware.

IV. Safe Trajectory Sampling

In this section, we detail our approach to safe RL using an approach that uses sampled trajectories from the GP posterior instead of moment-matching in order to compute the long-term expected return. Further, we will explicitly account for safety constraints.

A. Efficient Trajectory Sampling from GP Posteriors

Instead of using state marginal distributions to generate a trajectory realisation from the GP's predictions, it is possible to generate such a trajectory from the GP posterior. In [19], the authors present an efficient GP posterior sampling method where, instead of a distributional representation with a mean and covariance, the GP posterior is considered as a collection of sample paths. This pathwise perspective of the GP posterior arises from representing the GP prior in weight space [20]. By using Matheron's trick, the pathwise formulation of the GP posterior is

$$\underbrace{(f \mid \boldsymbol{X}, \boldsymbol{y})(\cdot)}_{\text{posterior}} \overset{d}{=} \underbrace{f(\cdot)}_{\text{prior}} + \underbrace{k(\cdot, \boldsymbol{X})(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I})^{-1}(\boldsymbol{y} - f(\boldsymbol{X}))}_{\text{data-dependent update}}. \tag{7}$$

Here, the posterior is a sum of the (random) prior and a (deterministic) data-dependent update term. Naively generating function/trajectory samples from this equation scales cubically in the number of query points $\boldsymbol{x}_*$. The key idea behind the efficient sampling strategy proposed in [19] is to use different representations of the GP prior and the (deterministic) update term in Equation (7). Specifically, they use a random Fourier features (RFF) prior, so that sampling from this prior scales linearly in the number of query points but cubically in the number of Fourier features. The update term in Equation (7) is approximated by a sparse variational GP update term [21], so that evaluating this term also scales linearly in the number of query points.

In [19], the authors use this trajectory-sampling approach instead of moment-matching within the PILCO framework for long-term predictions. Using these sampled trajectories, the expected return is then computed as a Monte Carlo estimate according to

$$J(\boldsymbol{\theta}) \approx \frac{1}{P} \sum_{i=1}^{P} \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}[R(\boldsymbol{x}_{i,t}) \mid \boldsymbol{\theta}], \tag{8}$$

where P is the number of sampled paths from the GP posterior.

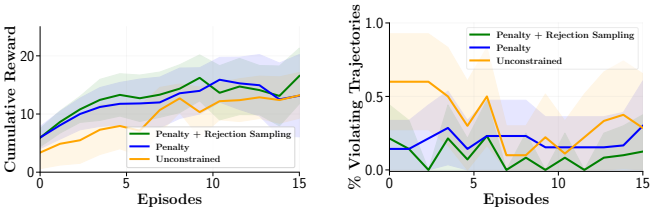
While this approach works well for uncertain initial distributions $p(\boldsymbol{x}_0)$, it does not account for safety constraints, which makes it impractical for use in a real-world system. In this paper, we will generalize this approach and explicitly account for safety constraints, bringing us a step closer applying RL to a real-world robotic system.

B. Adding Safety

We consider a set of safety constraints, such that trajectories $\boldsymbol{\tau}$ are safe to evaluate on the physical system. We compare two approaches for incorporating constraints into the policy optimization step: 1) we add an explicit penalty to the reward function, so that $R(\boldsymbol{x}_t) = r(\boldsymbol{x}_t) + c(\boldsymbol{x}_t)$, where $r(\boldsymbol{x}_t)$ is the original reward objective for the task and $c(\boldsymbol{x}_t)$ is a penalty term that accounts for the task constraints; 2) a rejection sampling approach which, in addition to the penalty term, also updates the policy using only the trajectories that satisfy constraints. We assume that the reward functions $r(\boldsymbol{x}_t)$ and penalty terms $c(\boldsymbol{x}_t)$ are known. Therefore, we can assess how well a trajectory solves the task according to $R(\boldsymbol{x}_t)$ and how safe it is as the number of sampled trajectories that violate constraints. To mimic real-world conditions, each learning episode is terminated as soon as a constraint is violated.

C. Reward Penalties

A common approach for incorporating domain knowledge, inductive bias and safety-specifications into the learned policy in RL is the addition of penalty functions to the



(a) Rewards (b) Constraint violations

Fig. 2: Metrics (constrained cartpole) averaged across 8 random seeds, where we execute the latest policy and assess rewards and constraint violations on the resulting trajectories. (a) Cumulative reward based on step-wise rewards R. (b) Number of constraint-violating trajectories across all seeds.

reward function. These penalties are soft constraints that discourage the policy from certain regions in the state space by associating nearby states with increasingly high negative reward the closer they are to the bad regions. For each environment, we design penalty terms $c(\boldsymbol{x}_t)$ which indicate proximity to constraint bounds. In line with MBRL algorithms which differentiate through the reward function R [17], [22], [18], we approximate discontinuous constraints with smooth differentiable functions.

D. Rejection Sampling

Leveraging the pathwise representation of the trajectory distribution [19], we check individual states in a batch of trajectories sampled from the dynamics model for safety constraint violations. If any constraint-violations are present,the corresponding trajectory is discarded. Only the remaining set of trajectories are used to estimate the expected return in Equation (2) by way of Monte Carlo. In cases where all trajectories are rejected, we re-sample a new set of paths and attempt rejection sampling again. We repeat until non constraint-violating paths are found or the number of path *re-sampling attempts* are exhausted. In such cases, we count on the diversity of the initial state distribution to start the next policy update step from a different starting state where it may be easier for the current policy to avoid constraint violations. The pathwise trajectory sampling [19] approach presents the GP posterior distribution as a collection of individual *full-horizon* trajectories in functional form; see Equation (7). Unlike the moment-matching trajectory representation, this form allows us to more easily evaluate and differentiate between safe and unsafe trajectories.

V. Experiments

In this section we evaluate the aforementioned methods for safe trajectory sampling in Section IV according to; 1) *performance* as specified by the combined reward $R(\boldsymbol{x}_t)$, 2) *safety* as the number of constraint-violating trajectories. In particular, we evaluate our method on two constrained continuous control tasks, namely a cartpole swing-up task and a reaching task on a 7 DOF robot. For the latter environment, we also demonstrate real-world policy execution and show that a learned safe policy can be executed on a real-robot; see Figure 3a.

For each task considered, we train a separate independent multivariate GP for each output dimension of the learned dynamics model $f : \mathbb{R}^D \times \mathbb{R}^U \to \mathbb{R}^D$. Furthermore, the targets of this model are the differences $\Delta_t = \boldsymbol{x}_t - \boldsymbol{x}_{t-1}$ of two consecutive states. Learning state differences Δ_t is a common design choice for MBRL with GP dynamics models [17], [23]. To infer next states $\boldsymbol{x}_t$, we now first sample Δ_t from the drift distribution $p(\Delta_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$ and add it to the previous state $\boldsymbol{x}_{t-1}$.

For all task settings, we learn an RBF network policy

$$\pi_\theta(\cdot) = g\left(\sum_{i=1}^{N_c} \omega_i k(\cdot, \boldsymbol{c}_i)\right), \tag{9}$$

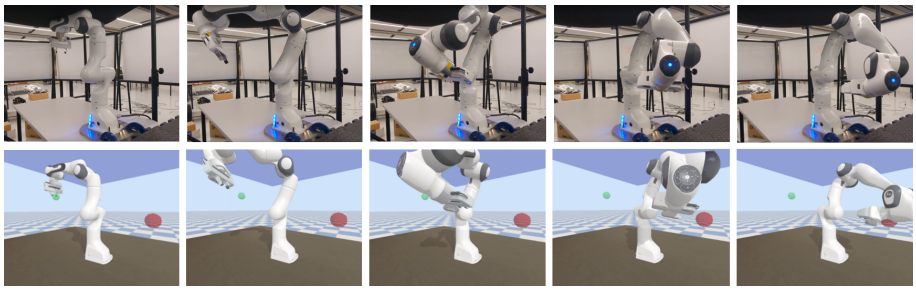
where $g(x)$ is a function that scales the output of the RBF network to an appropriate range $[\boldsymbol{a}, \boldsymbol{b}]$ where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^U$, thereby accounting for torque limits or other constraints on the control signals. The parameters θ of the RBF network are a set of N_c weights ω_i, RBF kernel centers $\boldsymbol{c}_i$, and RBF kernel hyperparameters; see Equation (3). As with the dynamics model, we also train a separate independent RBF network for each output dimension u of the policy.

We set the maximum number of re-sample attempts for rejection sampling to 10 for all experiments.

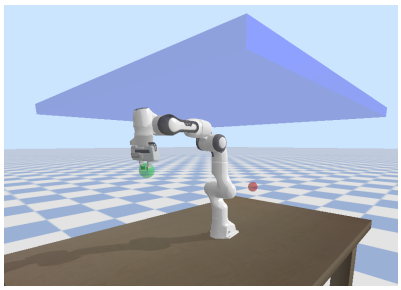
A. Constrained CartPole Swing-up

The cartpole swing-up task is a commonly used benchmark in model-based RL. It consists of an under-actuated system where the objective is to swing a pole and balance it upright by moving the cart. The observations $\boldsymbol{x}_t$ are made up of the position of the cart x, the angle of the pole θ, the velocity of the cart $\dot{x}$, and the angular velocity of pole $\dot{\theta}$. We control the system by applying a horizontal force $u \in [-10, 10]$.

In the constrained version of the task, the cart's position x is restricted such that $x \in [-0.5, 0.5]\,\text{m}$. This constraint places a bound on the left and right sides of the cart, limiting the set of optimal policies to those that swing the pole within this narrow region. We implement the penalty function for this bound as the sum of two Gaussian distributions with mean $\mu_1 = -0.5$ and $\mu_2 = 0.5$ respectively. The variance σ^2, which controls the strength of the penalty term, is set to $\sigma^2 = 0.000625$ for both distributions. This means the penalty meaningfully contributes to the reward R when the cart's position x is within $0.05\,\text{m}$ from either boundary. We count states where $x \leq -0.5$ and $x \geq 0.5$ as violations and terminate the episode immediately. We also use this termination condition during rejection sampling to decide which trajectories to discard and which trajectories to keep for evaluating the expected return in Equation (8). We set an episode length of 4.0 seconds and divide it into a time horizon of 50 time steps. All other task settings for cartpole are kept the same as in [13].



(a) Policy comparison on simulated (bottom) and real robot (top)



(b) Limit-Z environment

Fig. 3: Illustration of policy execution (a). The environment (b) consists of the robot placed on a table, with a target position $\boldsymbol{p}$ (red sphere) behind the robot and a low ceiling constraint (blue plane) above the robot.

1) Performance Evaluation: Our results in Figure 2a for cartpole are consistent with that of [19] where the MBRL approach of combining PILCO and a pathwise GP dynamics model was first introduced. Both methods are able to learn successful policies for swinging up the pole within 6–8 episodes even though introducing constraints makes solving the task more challenging.

2) Safety Evaluation: In general, our results in Figure 2b show that both the penalty term and the rejection sampling approaches had difficulties staying safe during training for this environment. We attribute this difficulty to the unique consideration of immediately terminating the episode as soon as a single constraint violation is detected, a behavior which is analogous to most safety-critical real-world systems. Terminating the episode early means the RL agent has to operate with less data to solve the problem. For the cartpole setting, the agent had enough data to solve the task but not while respecting constraints. However, both proposed safety approaches still outperform the baseline approach where constraints are ignored altogether.

B. Constrained Reaching Task

We set up the learning environment in PyBullet [24] where a simulated 7-DOF Franka Panda robot arm has to reach a fixed Cartesian target with its end-effector. The state observations $\boldsymbol{x}_t = [q_1, q_2, ..., q_7, \dot{q}_1, \dot{q}_2, ..., \dot{q}_7] \in \mathbb{R}^{14}$ consist of joint positions q_i and angular velocities $\dot{q}_i$. We control the robot with a continuous set of actions $\boldsymbol{u}_t = [\Delta q_1, \Delta q_2, ..., \Delta q_7] \in \mathbb{R}^7$, where each action corresponds to a specific joint residual value which indicates the change in joint position at time t. The low-level PID controller of the robot is thus provided with joint position targets $\boldsymbol{q}_{t-1} + \boldsymbol{u}_t$. The benefit of controlling the robot this way is that actions are centered at $\mathbf{0}$, i.e., an action $\boldsymbol{u}_t = \mathbf{0}$ translates to the robot staying still. The range of each residual value is constrained to facilitate smooth robot motion.

We place a ceiling constraint at a height of 0.85 m to mimic a factory setting where the robot has to operate in an environment with a reduced workspace. This constraint is shown as a blue plane in Figure 3b and is violated if the robot's end-effector goes above it. To discourage the policy from approaching the constraint, we add a penalty term $c(\boldsymbol{x}_t) = -\text{sigmoid}(-100(FK(\boldsymbol{x}_t)_z - 0.75))$ where $FK(\boldsymbol{x}_t)_z$ refers to the z-component of the end-effector position computed using forward kinematics $FK(\cdot)$. The rejection criteria for this environment is $FK(\boldsymbol{x}_t)_z \geq 0.85$ m.

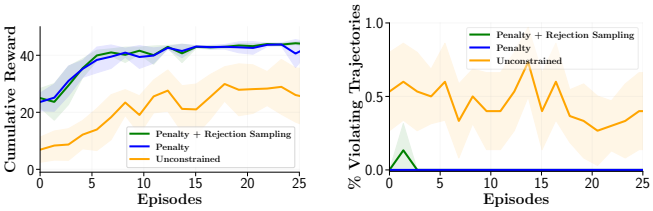
For our reaching experiments (shown in Figure 3b), the target $\boldsymbol{p}$ is positioned behind the robot at $\boldsymbol{p} = [-0.4, 0.0, 0.3]$ to increase the difficulty of the task. The reward $r(FK(\boldsymbol{x}_t))$ is a Gaussian with mean $\boldsymbol{\mu} = \boldsymbol{p}$ and covariance $\boldsymbol{\Sigma} = \boldsymbol{I}$. We set a time horizon of 50 time steps which corresponds to an episode length of 12.0 seconds.

1) Performance Evaluation: As with the cartpole domain, the PILCO algorithm with a pathwise GP dynamics model is able to learn the task successfully (Figure 4a). Both approaches (reward penalty and rejection sampling) converge to a solution at around 5 episodes. This is extremely data-efficient since it corresponds to around 100 seconds of data in the Pybullet simulation (including data used to pre-train the GP).

2) Safety Evaluation: In our results shown in Figure 4b the rejection sampling approach has constraint violations in the beginning of training but improves in later episodes where the number of constraint violations converges to 0. However, the penalty term approach avoids constraint violations even after the first policy update at episode 0 in Figure 4b. After 25 episodes, both approaches have learned safe policies according to the constraints we consider.

C. Real-world Policy Execution

We replicate the constrained reaching task in the real-world by placing a black tarp at 0.85m from the table as shown in Figure 3a. After training a policy and a dynamics model in the PyBullet environment (Figure 3b), we execute it directly on the physical robot system. As in the Pybullet simulation, we provide the policy with real-robot joint positions and angular velocities at every time step and predict control commands $\boldsymbol{u}_t = \Delta \boldsymbol{q}_t + \boldsymbol{q}_{t-1}$. These are sent to a low-level controller running on the robot which in turn moves the robot. We repeat this process until we reach the target. We found that the physical robot behaves similarly to the simulated robot and avoids the "ceiling" constraint successfully, even though the starting state is not exactly the same each time due to measurement noise. A comparison of executing the same policy in the simulated environment and on the robot is shown in Figure 3a.



(a) Rewards (b) Constraint Violations

Fig. 4: Results for the constrained 7-DOF reaching environment averaged across 9 random seeds. a) The cumulative reward based on step-wise rewards R. b) The number of constraint violating trajectories across all seeds. For both metrics, we execute the latest policy and assess rewards and constraint violations on the resulting trajectories.

VI. Discussion

Our results show that both the reward penalty and rejection sampling approach are effective at reducing constraint violations in the environments we consider compared to the baseline. We also observe that rejection sampling does not hurt the task performance or severely worsen the number of constraint violations by the learned policy despite the rejection sampling approach working with less data when computing the expected return. A possible explanation for this observation is that even though the penalty-only approach uses more trajectories from the model, the additional (faulty) trajectories may contribute to the variance of the expected return. As such, the policy's learning and constraint-satisfaction ability is impacted.

Motion planning methods produce a state trajectory $\boldsymbol{\tau} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ and execute it on the physical robot using a low-level trajectory-follower controller. However, we opt for directly executing a policy instead since it is more flexible and can smoothly recover from deviations from the optimal, safer trajectory in case of disturbances. Low-level controllers such as PID, in comparison, would jerk the robot proportional to how far it is from the desired trajectory and may overshoot, potentially causing constraint violations as a result. Also, trajectory following may not even be possible since some robot systems for real-world execution issue an abort status when the current state differs too much from the next way-point in the trajectory.

The dynamics of the experiments considered in this work are smooth. Particularly for the reaching task in *free space*. This means the state configuration of the robot changes smoothly given control inputs. As such, we can model them sufficiently accurately with an RBF-kernel GP. However, for more complex tasks with contact dynamics such as picking and placing, the true dynamics would be non-smooth. In such a setting, additional consideration would have to be taken when learning the dynamics model with GPs.

VII. Conclusion

In this paper, we introduce an approach that explicitly accounts for safety constraints within a model-based RL set-up. In particular, we use sampled trajectories from a GP dynamical model to evaluate whether safety constraints have been violated. We evaluate two different cases of incorporating constraints into learning: a) by adding a penalty function to the reward; b) by additionally rejecting trajectory samples that violate the constraints. Both approaches can work well as we demonstrate on two challenging robotic systems: the cart-pole swing-up and a reaching task with a Panda robot in both simulation and real world. In future work, we will extend the proposed approach to deal with robotic manipulation tasks using multi-modal data such as tactile, visual, and proprioceptive.

References

[1] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *IJRR*, 2011.

[2] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. A. Bagnell, and S. Srinivasa, "CHOMP: Covariant Hamiltonian Optimization for Motion Planning," *IJRR*, 2013.

[3] H. Attias, "Planning by probabilistic inference," in *AISTATS*, 2003.

[4] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion Planning as Probabilistic Inference using Gaussian Processes and Factor Graphs." in *RSS*, 2016.

[5] D. Zhou, R. Jia, H. Yao, and M. Xie, "Robotic Arm Motion Planning Based on Residual Reinforcement Learning," in *ICCAE*, 2021.

[6] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *JIRS*, 2017.

[7] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates," in *ICRA*, 2017.

[8] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *TPAMI*, 2015.

[9] J. Schneider, "Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning," *NeurIPS*, 1996.

[10] S. Rubrecht, V. Padois, P. Bidaud, M. Broissia, and M. Da Silva Simoes, "Motion Safety and Constraints Compatibility for Multibody Robots," *AR*, 2012.

[11] J. García and F. Fernández, "A Comprehensive Survey on Safe Reinforcement Learning," *JMLR*, 2015.

[12] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey," *IJRR*, 2013.

[13] S. Kamthe and M. P. Deisenroth, "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control," in *AISTATS*, 2018.

[14] H. Kim, M. Jordan, S. Sastry, and A. Ng, "Autonomous Helicopter Flight via Reinforcement Learning," *NeurIPS*, 2003.

[15] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models," *NeurIPS*, 2018.

[16] M. P. Deisenroth, G. Neumann, and J. Peters, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, 2013.

[17] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A Model-based and Data-efficient Approach to Policy Search," in *ICML*, 2011.

[18] P. Parmas, C. E. Rasmussen, J. Peters, and K. Doya, "Flexible Model-based Policy Search Robust to the Curse of Chaos," in *ICML*, 2018.

[19] J. T. Wilson, V. Borovitskiy, A. Terenin, P. Mostowsky, and M. P. Deisenroth, "Pathwise Conditioning of Gaussian Processes," *JMLR*, 2021.

[20] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[21] M. Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," in *AISTATS*, 2009.

[22] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *arXiv preprint arXiv:1802.10592*, 2018.

[23] B. Van Niekerk, A. Damianou, and B. Rosman, "Online Constrained Model-based Reinforcement Learning," *arXiv preprint arXiv:2004.03499*, 2020.

[24] E. Coumans and Y. Bai, "PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning," http://pybullet.org, 2016–2021.