GP-ILQG: Data-driven Robust Optimal Control for Uncertain Nonlinear Dynamical Systems

Gilwoo Lee **Siddhartha S. Srinivasa** **Matthew T. Mason** *

Abstract

As we aim to control complex systems, use of a simulator in model-based reinforcement learning is becoming more common. However, it has been challenging to overcome the Reality Gap, which comes from nonlinear model bias and susceptibility to disturbance. To address these problems, we propose a novel algorithm that combines data-driven system identification approach (Gaussian Process) with a Differential-Dynamic-Programming-based robust optimal control method (Iterative Linear Quadratic Control). Our algorithm uses the simulator's model as the mean function for a Gaussian Process and learns only the difference between the simulator's prediction and actual observations, making it a natural hybrid of simulation and real-world observation. We show that our approach quickly corrects incorrect models, comes up with robust optimal controllers, and transfers its acquired model knowledge to new tasks efficiently.

1 Introduction

As we aim to control more complex robotic systems autonomously, simulators are being more frequently used for training in model-based reinforcement learning [7]. A simulator allows us to explore various policies without damaging the robot, and is also capable of generating a large amount of synthetic data with little cost and time.

However, we often observe that simulator-based policies perform poorly in real world, due to model discrepancy between the simulation and the real world. This discrepancy arises from two fundamental challenges: (1) system identification to match the simulation model with the real world requires the exploration of a large state space at the risk of damaging the robot, and (2) even with good system identification, there is still discrepancy due to the limitations of a simulator's ability to render real-world physics.

Stochastic optimal control algorithms attempt to partially address this issue by artificially injecting noise into the simulation during training [5, 24], or by explicitly modeling multiplicative noise [22, 20].

If the task domain is predefined, exploration can be limited to task-specific trajectories, and system identification can be coupled with trajectory optimization [21]. Some recent works have suggested policy training with multiple models, which results in a policy robust to model variance [11, 3]. While these methods have shown some successful results, they are still limited by the expressiveness of the simulator's model. If the true model is outside of the simulator's model space, little can be guaranteed.

Thus, although these algorithms produce more robust policies, they fail to address the fundamental issue: there is unknown but structured model discrepancy between the simulation and the real world.

In this paper, we propose a novel algorithm that addresses both model bias and multiplicative noise. Our work is based on the following key insight:

*gilwool, siddh, mason@cs.cmu.edu. All authors are affiliated with Robotics Institute, Carnegie Mellon University.

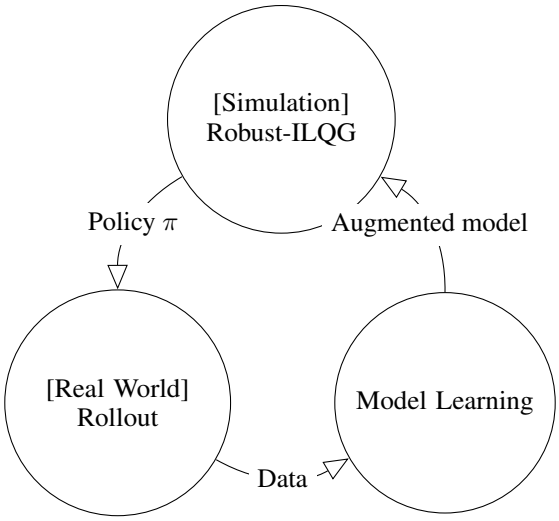

Figure 1: GP-ILQG overview

> Explicitly correcting model bias and incorporating the correction as well as our uncertainty of the correction in optimal control enables lifelong learning of the system *and* robust control under uncertainty.

Our algorithm iterates over simulation-based optimal control, real-world data collection, and model learning, as illustrated in Figure 1. Starting from a potentially incorrect model given by the simulator, we obtain a control policy, with which we collect data in the real world. This data feeds into model learning, during which we correct model bias and estimate our uncertainty of the correction. Both the correction and its uncertainty are incorporated into computing a robust optimal control policy, which then gets used to collect more data.

Our approach improves any simulator beyond the scope of its model space to match real-world observations and produces an optimal control policy robust to model uncertainty and multiplicative noise. The improved simulator uses previous real-world observations to infer the true model when it explores previously visited space, but when it encounters a new region, it relies on the simulator's original model. Due to this hybrid nature, our algorithm shows faster convergence to the optimal policy than a pure data-driven approach [13] or a pure simulation-based approach. Moreover, as it permanently improves the simulator, it shows even faster convergence in new tasks in similar task domain.

2 Related Work

Most model-based reinforcement learning has both model learning (system identification) and policy optimization components [7]. The data for a model comes either from real world or simulation, and is combined to construct a model via nonlinear function approximators such as Locally Weighted Regression [2], Gaussian Processes [17], or Neural Networks [12]. Once the model is built, a typical policy gradient method computes the derivatives of the cost function with respect to control parameters [4, 8].

If an analytic model is given, e.g., via equations of motion or as a simulator[2], one can use classical optimal control techniques such as Differential Dynamic Programming (DDP) [6], which compute a reference trajectory as well as linear feedback control law. For robustness, Iterative Linear Quadratic Gaussian Control (ILQG) [22] or H-∞ Control [20] can be used to incorporate multiplicative noise. Variants of [22] have been used to generate guiding policies for data-driven RL methods [8, 28]. Recently, there have been some attempts to combine DDP or ILQG with data-driven models by replacing analytical models with locally linear models [10, 25] or nonlinear models [13, 15, 26, 14] learned by Gaussian Processes or Neural Networks.

[2]We consider black-box simulators as analytical models, as derivatives can be taken by finite differencing.

The goal of our algorithm is closely aligned with those of [27] and [1]. [27] has proposed a framework in which the robot maintains two controllers, one for the simulator and another for the real world, and aims to narrow the difference. [1] assumes a deterministic real world, constructs an optimal policy based on the simulator's deterministic model, and evaluates its performance in the real world, while successively augmenting the simulator's model with time-dependent corrections based on real-world observations. Our method considers a stochastic system and the correction is not time-dependent.

3 Approach

We work with stochastic nonlinear dynamical systems, whose evolution is described by the stochastic differential equation

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u})\, dt + \mathsf{F}(\mathbf{x}, \mathbf{u})\, d\boldsymbol{\omega} \tag{1}$$

where the applied control $\mathbf{u} \in \mathbb{R}^m$ transitions the state $\mathbf{x} \in \mathbb{R}^n$ of the system according to a linear sum of the system dynamics $\mathbf{f} \in \mathbb{R}^n$ and state-dependent amplification $\mathsf{F} \in \mathbb{R}^{n \times p}$ of Brownian noise increment $d\boldsymbol{\omega} \in \mathbb{R}^p$.

We assume that we have full knowledge of F and partial knowledge of $\mathbf{f}$, e.g., from a simulator. We represent $\mathbf{f}$ as the sum of simulated component $\mathbf{f}_{\text{sim}}$ and unknown residual ϵ,

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{f}_{\text{sim}}(\mathbf{x}, \mathbf{u}) + \epsilon(\mathbf{x}, \mathbf{u}).$$

Then, in a discrete setting with fixed step size δt, (1) can be rewritten as the following:

$$\Delta\mathbf{x} = (\mathbf{f}_{\text{sim}} + \epsilon)\delta t + \mathsf{F}\, \boldsymbol{\xi}_{\mathsf{F}} \sqrt{\delta t} \tag{2}$$

where $\boldsymbol{\xi}_{\mathsf{F}} \sim \mathcal{N}(0, I_{p \times p})$ and $\sqrt{\delta t}$ appears because the noise covariance increases linearly with time. We omit $\mathbf{x}$ and $\mathbf{u}$ for clarity.

Nonzero model bias results in nonzero difference between the model's prediction $\mathbf{f}_{\text{sim}}\, \delta t$ and the actual $\delta\mathbf{x}$. From (2), the difference is equivalent to

$$\Delta\mathbf{x} - \mathbf{f}_{\text{sim}}\, \delta t = \epsilon\, \delta t + \mathsf{F}\, \boldsymbol{\xi}_{\mathsf{F}} \sqrt{\delta t}$$

In expectation,

$$\mathbb{E}[\epsilon\, \delta t] = \mathbb{E}[\Delta\mathbf{x} - \mathbf{f}_{\text{sim}}\, \delta t - \mathsf{F}\, \boldsymbol{\xi}_{\mathsf{F}} \sqrt{\delta t}] = \mathbb{E}[\Delta\mathbf{x} - \mathbf{f}_{\text{sim}}\, \delta t]$$

as the mean of $\boldsymbol{\xi}_{\mathsf{F}}$ is 0.

In order to correct the nonzero model bias, we approximate $\epsilon\, \delta t$ from data. From rollouts of trajectories, we collect a set of $\{\mathbf{x}, \mathbf{u}, \Delta\mathbf{x} - \mathbf{f}_{\text{sim}}\, \delta t\}$ tuples. Assuming that the the covariance of $\epsilon\, \delta t$ grows linearly with δt, we use a Gaussian Process (GP) [17], which estimates $\epsilon\, \delta t$ as a Gaussian distribution:

$$\epsilon(\mathbf{x}, \mathbf{u})\delta t \sim \mathcal{N}\big(\hat{\epsilon}(\mathbf{x}, \mathbf{u})\delta t, \Sigma(\mathbf{x}, \mathbf{u})\delta t\big)$$

whose derivation we provide in Section 3.1. Let $\Sigma(\mathbf{x}, \mathbf{u})$ be decomposed into $\mathsf{G}(\mathbf{x}, \mathbf{u})\, \mathsf{G}(\mathbf{x}, \mathbf{u})^\top$, with $\mathsf{G} \in \mathbb{R}^{n \times n}$. Now (2) can be rewritten as the following:

$$\Delta\mathbf{x} = (\mathbf{f}_{\text{sim}} + \hat{\epsilon})\, \delta t + \mathsf{F}\, \boldsymbol{\xi}_{\mathsf{F}} \sqrt{\delta t} + \mathsf{G}\, \boldsymbol{\xi}_{\mathsf{G}} \sqrt{\delta t} \tag{3}$$

where $\boldsymbol{\xi}_{\mathsf{G}} \sim \mathcal{N}(0, I_{n \times n})$.

The original ILQG handles stochastic systems without uncertainty, but the above formulation makes it straightforward to extend ILQG to incorporate uncertainty, which we refer to as Robust-ILQG (see Section 3.2).

Our approach is summarized in Algorithm 1. Initially, with zero data and no policy, we start by using ILQG[3] to obtain a locally optimal policy for the suboptimal model provided by $\mathbf{f}_{\text{sim}}$. Then, with rollouts of the computed policy, we compute the error between $\mathbf{f}$ and $\mathbf{f}_{\text{sim}}$, with which we train a GP that estimates ϵ. We run Robust-ILQG again, but with the updated dynamics. This process is repeated until the policy converges.

[3] Robust-ILQG is equivalent to ILQG in the absence of ϵ.

Algorithm 1 GP-ILQG

Require: π, $\mathbf{f}_{\text{sim}}$, F, $\mathbf{D}$
  if $\mathbf{D} \neq \emptyset$ **then**
    $[\hat{\epsilon}, \mathsf{G}] \leftarrow \texttt{GP}(\mathbf{D})$
  end if
  if $\pi == NIL$ **then**
    $\pi \leftarrow \texttt{Robust-ILQG}(\mathbf{f}_{\text{sim}} + \hat{\epsilon}, \mathsf{F}, \mathsf{G})$
  end if
  while $\Delta\pi > \gamma$ **do**
    Collect $\{(\mathbf{x}, \mathbf{u}, \delta\mathbf{x} - \mathbf{f}\,\delta t)\}$ with rollouts of π.
    $\mathbf{D} \leftarrow \mathbf{D} \cup \{(\mathbf{x}, \mathbf{u}, \delta\mathbf{x} - \mathbf{f}\,\delta t)\}$
    $[\hat{\epsilon}, \mathsf{G}] \leftarrow \texttt{GP}(\mathbf{D})$
    $\pi \leftarrow \texttt{Robust-ILQG}(\mathbf{f}_{\text{sim}} + \hat{\epsilon}, \mathsf{F}, \mathsf{G})$.
  end while

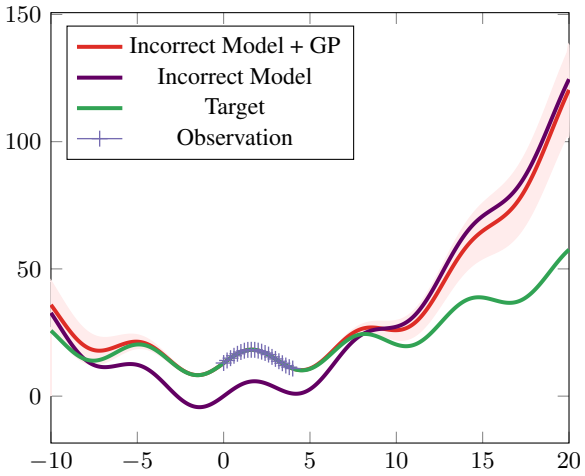

Figure 2: Use of Gaussian Process to correct an incorrect model. Near previous observations, the model's prediction is corrected to match the target. When the test input is far from the prior observations, predictions resort to the incorrect model. Shaded area indicates 95% confidence interval.

One major advantage of our approach is that it is straightforward to use the improved model for a new task. Given observations learned from a previous task, Algorithm 1 starts with an estimate of ϵ. Whenever the algorithm explores a previously visited state-control space, the learner corrects any simulator error and provides smaller uncertainty covariance; in a new region, the algorithm relies on the original model $\mathbf{f}_{\text{sim}}$ with larger uncertainty covariance than the explored region. This results in a more accurate model, and as Robust-ILQG takes into account uncertainty, it also results in a more robust policy than those that rely only on the simulator or on a locally learned dynamics. When using only a partially-correct simulated model, the policy generated is always limited by the simulator's accuracy; when using a locally learned dynamics, the robot knows very little outside previously explored regions. Our algorithm, which combines both, our algorithm quickly outperforms the simulator-only approaches and requires less data than the latter.

3.1 Gaussian Process with the Simulator as a Mean Function

Gaussian Process Regression [17] is a nonlinear regression technique that has been successfully used in many model-based reinforcement learning approaches [4, 13]. A Gaussian Process is a stochastic process in which any finite subset is jointly Gaussian. Given a set of inputs and corresponding outputs $\{x_i, y_i\}_{i=1}^n$ the distribution of $\{y_i\}_{i=1}^n$ is defined by the covariance given by a kernel function $k(x_i, x_j)$.

We use a variant of Gaussian Processes that uses a nonzero mean function. With $f : X \to Y$ as the mean function, the prediction for test input x becomes the following:

$$\mathbb{E}[y] = f(x) + k_{xX}^\top K^{-1}(Y - f(X))$$
$$var(y) = k_{xx} - k_{xX}^\top K^{-1} k_{xX}$$

where k_{xX} is the covariance between test input x and training set X, K is the covariance matrix of among the elements of X. In this formulation, the GP provides a posterior distribution given $f(x)$ and observations.

Using the simulator as the mean function for a Gaussian Process allows us to combine the simulator and real-world observations smoothly. Near previous observations, the simulator's prediction is corrected to match the target. When the test input is far from the previous observations, the predictions resort to the simulator. See Figure 2 for an illustration.

As we have both $\mathbf{x}$ and $\mathbf{u}$ as the input, we define $\widetilde{\mathbf{x}} = [\mathbf{x}^\top \mathbf{u}^\top]^\top$ to be the input and $\delta\mathbf{x}$ to be the output, and use $\mathbf{f}_{\text{sim}}$ as the mean function. Then, the GP serves to correct the error, $\delta\mathbf{x} - \mathbf{f}_{\text{sim}}\,\delta t$. We use the ARD (Automatic Relevance Determination) squared exponential kernel function:

$$k(\widetilde{\mathbf{x}}_i, \widetilde{\mathbf{x}}_j) = \sigma_f^2 \exp(-(\widetilde{\mathbf{x}}_i - \widetilde{\mathbf{x}}_j)^\top \Lambda^{-1} (\widetilde{\mathbf{x}}_i - \widetilde{\mathbf{x}}_j))$$

where σ_f^2 is the signal variance and Λ controls the characteristic length of each input dimension. These hyperparameters are optimized to maximize log-likelihood of data using conventional optimization methods. For multiple dimensions of Y, we train one GP per dimension and treat each dimension to be independent.

3.2 Robust-ILQG

In order to address both uncertainty and stochasticity in computing an optimal policy, we propose a new algorithm, which we refer to as Robust-Iterative Linear Quadratic Gaussian Control (Robust-ILQG). Robust-ILQG is an extension to the original ILQG [22], a variant of Differential Dynamic Programming (DDP) for stochastic systems. It makes second-order approximation of the value function to the second order and first-order approximation of the dynamics. The main difference between ILQG and Robust-ILQG is that the latter takes into account uncertainty in dynamics, as we will see in the following derivation.

We use a discretized version of a continuous system, with a fixed step size δt. From the equation

$$\begin{aligned} \Delta\mathbf{x} &= \mathbf{f}(\mathbf{x}, \mathbf{u})\delta t + \mathsf{F}(\mathbf{x}, \mathbf{u})\sqrt{\delta t}\boldsymbol{\xi}_{\mathsf{F}} \\ &= (\mathbf{f}_{\text{sim}} + \hat{\epsilon})\delta t + \mathsf{F}\,\boldsymbol{\xi}_{\mathsf{F}}\sqrt{\delta t} + \mathsf{G}\,\boldsymbol{\xi}_{\mathsf{G}}\sqrt{\delta t} \end{aligned} \tag{4}$$

The state $\mathbf{x}'$ at the next timestep can be written as

$$\mathbf{x}' = \mathbf{x} + (\mathbf{f}_{\text{sim}} + \hat{\epsilon})\delta t + \mathsf{F}\,\boldsymbol{\xi}_{\mathsf{F}}\sqrt{\delta t} + \mathsf{G}\,\boldsymbol{\xi}_{\mathsf{G}}\sqrt{\delta t} \tag{5}$$

Given a trajectory $\{\mathbf{x}_0, \mathbf{u}_0, \cdots \mathbf{x}_T\}$, the total cost is given as

$$J(\mathbf{x}_0) = \mathbb{E}\left[l_T(\mathbf{x}_T) + \sum_{i=0}^{T-1} l(\mathbf{x}_i, \mathbf{u}_i)\right] \tag{6}$$

where l_T is the final cost and l is the running cost. Our objective is to find a deterministic policy $\pi : \mathbf{X} \to \mathbf{U}$ that minimizes this total cost.

We consider the local variation of value function with respect to change in $\mathbf{x}$ and $\mathbf{u}$. The Bellman equation gives us the following:

$$V(\mathbf{x} + \delta\mathbf{x}) = l(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u}^*) + \mathbb{E}\left[V'(\mathbf{x}' + \delta\mathbf{x}'^*)\right] \tag{7}$$

where V is the value function at a particular timestep and V' is the value function at the next timestep. $\delta\mathbf{x}'^*$ refers to the variation of the next state given an optimal control change $\delta\mathbf{u}^*$.

Analogous to ILQR which does not have stochastic components, we can analytically derive the local optimal improvement $\delta\mathbf{x}^*$ by taking first order approximation of dynamics and second order approximation of the value function.

The local deviation of the next state from its nominal (deterministic) state can be written as the following:

$$\delta \mathbf{x}' = \mathbf{A}\delta\mathbf{x} + \mathbf{B}\delta\mathbf{u} + \mathsf{C}\boldsymbol{\xi}_{\mathsf{F}} + \mathsf{D}\boldsymbol{\xi}_{\mathsf{G}} \tag{8}$$

where $\mathbf{A}, \mathbf{B}$ linearizes the deterministic component of dynamics at the current time step, defined as

$$\begin{aligned}\mathbf{A} &= \mathbf{I} + (\mathbf{f}_{\text{sim}\,\mathbf{x}} + \hat{\epsilon}_{\mathbf{x}})\delta t \\ \mathbf{B} &= (\mathbf{f}_{\text{sim}\,\mathbf{u}} + \hat{\epsilon}_{\mathbf{u}})\delta t\end{aligned}$$

and C, D captures the deviation from nominal (deterministic) state caused by stochastic terms

$$\begin{aligned}\mathsf{C} &= (\mathsf{F} + \mathsf{F}_{\mathbf{x}} \otimes \delta\mathbf{x} + \mathsf{F}_{\mathbf{u}} \otimes \delta\mathbf{u})\sqrt{\delta t} \\ \mathsf{D} &= (\mathsf{G} + \mathsf{G}_{\mathbf{x}} \otimes \delta\mathbf{x} + \mathsf{G}_{\mathbf{u}} \otimes \delta\mathbf{u})\sqrt{\delta t}.\end{aligned}$$

where $\otimes$ denotes tensor-vector multiplication as defined in Section 6 such that $\mathsf{F}_{\mathbf{x}} \otimes \delta\mathbf{x}$, $\mathsf{G}_{\mathbf{x}} \otimes \delta\mathbf{x}$ are matrices of size $n \times p$ and $\mathsf{F}_{\mathbf{u}} \otimes \delta\mathbf{u}$, $\mathsf{G}_{\mathbf{u}} \otimes \delta\mathbf{u}$ are of size $n \times m$.

Assuming that the variation of value at the next timestep can be written in a quadratic form,

$$V'(\mathbf{x}' + \delta\mathbf{x}') = s' + \mathbf{s}'^{\top}\delta\mathbf{x}' + \frac{1}{2}\delta\mathbf{x}'^{\top}\mathbf{S}'\delta\mathbf{x}' \tag{9}$$

the local variation of cost-to-go at the current time step can be written as [4]

$$V(\mathbf{x} + \delta\mathbf{x}) = l(\mathbf{x}, \mathbf{u}) + \mathbf{l}_{\mathbf{x}}^{\top}\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^{\top}\mathbf{l}_{\mathbf{xx}}\delta\mathbf{x} + \mathbf{l}_{\mathbf{u}}^{\top}\delta\mathbf{u} + \frac{1}{2}\delta\mathbf{u}^{\top}\mathbf{l}_{\mathbf{uu}}\delta\mathbf{u} + \mathbb{E}[V'(\mathbf{x}' + \delta\mathbf{x}')]$$

Note that, when taking expectation for $\mathbf{s}^{\top}\delta\mathbf{x}'$, the stochasticity and uncertainty terms disappear as the means of $\boldsymbol{\xi}_{\mathsf{F}}$ and $\boldsymbol{\xi}_{\mathsf{G}}$ are zero, while for the second-order terms such as $\delta\mathbf{x}'^{\top}(\cdot)\delta\mathbf{x}'$, their covariances must be taken into account.[5].

Expanding above equation with the definitions of partial derivatives,

$$\begin{aligned}V(\mathbf{x} + \delta\mathbf{x}) = l + s' + \mathbf{s}'^{\top}(\mathbf{A}\delta\mathbf{x} + \mathbf{B}\delta\mathbf{u}) + \mathbf{l}_{\mathbf{x}}\delta\mathbf{x} + \mathbf{l}_{\mathbf{u}}\delta\mathbf{u} \\ + \frac{1}{2}\Big((\mathbf{A}\delta\mathbf{x} + \mathbf{B}\delta\mathbf{u})^{\top}\mathbf{S}'(A\delta\mathbf{x} + B\delta\mathbf{u}) + \delta\mathbf{x}^{\top}\mathbf{l}_{\mathbf{xx}}\delta\mathbf{x} + \delta\mathbf{u}^{\top}\mathbf{l}_{\mathbf{uu}}\delta\mathbf{u}\Big) \\ + \frac{1}{2}\big(tr(\mathsf{C}\mathsf{C}^{\top}\mathbf{S}') + tr(\mathsf{D}\mathsf{D}^{\top}\mathbf{S}')\big)\delta t\end{aligned} \tag{10}$$

where the trace terms can be re-written as the following:

$$\begin{aligned}tr(\mathsf{C}\mathsf{C}^{\top}\mathbf{S}') &= tr\Big(\sum_i (\mathsf{F}^{(i)} + \mathsf{F}_{\mathbf{x}}^{(i)} + \mathsf{F}_{\mathbf{u}}^{(i)})(\mathsf{F}^{(i)} + \mathsf{F}_{\mathbf{x}}^{(i)} + \mathsf{F}_{\mathbf{u}}^{(i)})^{\top}\mathbf{S}'\Big) \\ &= \sum_i (\mathsf{F}^{(i)} + \mathsf{F}_{\mathbf{x}}^{(i)} + \mathsf{F}_{\mathbf{u}}^{(i)})^{\top}\mathbf{S}'(\mathsf{F}^{(i)} + \mathsf{F}_{\mathbf{x}}^{(i)} + \mathsf{F}_{\mathbf{u}}^{(i)})\end{aligned}$$

where the superscripts (i) denote the ith column of the corresponding matrix and subscripts denote their partial derivatives.

[4] We assume that $\mathbf{l}_{\mathbf{ux}}$ is zero.

[5] e.g., for a randomvariable $\xi \sim \mathcal{N}(0, \Sigma)$, $\mathbb{E}[\xi^{\top}S\xi] = tr(\Sigma S)$

Then, combining the like terms together, (10) becomes

$$
\begin{aligned}
&= \underbrace{l + s' + \frac{1}{2}\delta t \sum_i \mathsf{F}^{(i)\top}\,\mathbf{S}'\,\mathsf{F}^{(i)} + \frac{1}{2}\delta t \sum_j \mathsf{G}^{(j)\top}\,\mathbf{S}'\,\mathsf{G}^{(j)}}_{q} \\
&+ \underbrace{\left(\mathbf{s}'^{\top}\mathbf{A} + \mathbf{l}_{\mathbf{x}}^{\top} + \delta t \sum_i \mathsf{F}^{(i)\top}\,\mathbf{S}'\,\mathsf{F}_{\mathbf{x}}^{(i)} + \delta t \sum_j \mathsf{G}^{(j)\top}\,\mathbf{S}'\,\mathsf{G}_{\mathbf{x}}^{(j)}\right)}_{\mathbf{q}^{\top}} \delta\mathbf{x} \\
&+ \frac{1}{2}\delta\mathbf{x}^{\top} \underbrace{\left(\mathbf{A}^{\top}\mathbf{S}'\mathbf{A} + \mathbf{l}_{\mathbf{xx}} + \delta t \sum_i \mathsf{F}_{\mathbf{x}}^{(i)\top}\,\mathbf{S}'\,\mathsf{F}_{\mathbf{x}}^{(i)} + \delta t \sum_j \mathsf{G}_{\mathbf{x}}^{(j)\top}\,\mathbf{S}'\,\mathsf{G}_{\mathbf{x}}^{(j)}\right)}_{\mathbf{Q}} \delta\mathbf{x} \\
&+ \underbrace{\left(\mathbf{s}'^{\top}\mathbf{B} + \mathbf{l}_{\mathbf{u}}^{\top} + \delta t \sum_i \mathsf{F}^{(i)\top}\,\mathbf{S}'\,\mathsf{F}_{\mathbf{u}}^{(i)} + \delta t \sum_j \mathsf{G}^{(j)\top}\,\mathbf{S}'\,\mathsf{G}_{\mathbf{u}}^{(j)}\right)}_{\mathbf{g}^{\top}} \delta\mathbf{u} \\
&+ \delta\mathbf{x}^{\top} \underbrace{\left(\mathbf{A}^{\top}\mathbf{S}'\mathbf{B} + \delta t \sum_i \mathsf{F}_{\mathbf{x}}^{(i)\top}\,\mathbf{S}'\,\mathsf{F}_{\mathbf{u}}^{(i)} + \delta t \sum_j \mathsf{G}_{\mathbf{x}}^{(j)\top}\,\mathbf{S}'\,\mathsf{G}_{\mathbf{u}}^{(j)}\right)}_{\mathbf{G}^{\top}} \delta\mathbf{u} \\
&+ \frac{1}{2}\delta\mathbf{u}^{\top} \underbrace{\left(\mathbf{B}^{\top}\mathbf{S}'\mathbf{B} + \mathbf{l}_{\mathbf{uu}} + \delta t \sum_i \mathsf{F}_{\mathbf{u}}^{(i)\top}\,\mathbf{S}'\,\mathsf{F}_{\mathbf{u}}^{(i)} + \delta t \sum_j \mathsf{G}_{\mathbf{u}}^{(j)\top}\,\mathbf{S}'\,\mathsf{G}_{\mathbf{u}}^{(j)}\right)}_{\mathbf{H}} \delta\mathbf{u}
\end{aligned}
$$

Minimizing this with respect to $\delta\mathbf{u}$ gives the optimal $\delta\mathbf{u}^*$:

$$\delta\mathbf{u}^* = -\mathbf{H}^{-1}\mathbf{g} - \mathbf{H}^{-1}\mathbf{G}\delta\mathbf{x},$$

Plugging this $\delta\mathbf{u}^*$ back into (10), we get a quadratic form of $V(\mathbf{x} + \delta\mathbf{x})$:

$$V(\mathbf{x} + \delta\mathbf{x}) = s + \delta\mathbf{x}^{\top}\mathbf{s} + \frac{1}{2}\delta\mathbf{x}^{\top}\mathbf{S}\delta\mathbf{x}$$

with

$$
\begin{aligned}
\mathbf{S} &= \mathbf{Q} - \mathbf{G}^{\top}\mathbf{H}^{-1}\mathbf{G} \\
\mathbf{s} &= \mathbf{q} - \mathbf{G}^{\top}\mathbf{H}^{-\top}\mathbf{g} \\
s &= q - \frac{1}{2}\mathbf{g}^{\top}\mathbf{H}^{-\top}\mathbf{g}
\end{aligned}
$$

Note that, in the absence of uncertainty terms, this is equivalent to iLQG as introduced in [22], and further, in the absence of both uncertainty and stochsaticity, this is equivelent to iLQR [9].

To make a local improvement of the nominal trajectory, we perform a backward pass to update the value function and the optimal $\delta\mathbf{u}^*$ for each timestep, starting with $\mathbf{S}_T = \mathbf{l}_{\mathbf{xx}}^{(T)}$, $\mathbf{s}_T = \mathbf{l}_{\mathbf{x}}^{(T)}$, $s_T = l_T$.

During a forward pass, the nominal trajectory is updated by applying $\delta\mathbf{u} = -\alpha\mathbf{H}^{-1}\mathbf{g} - \mathbf{H}^{-1}\mathbf{G}\delta\mathbf{x}$ to the deterministic part of the system. We use backtracking line-search to find α that minimizes the total cost.

For a long horizon task, Model Predictive Contol can be used with Robust-ILQG, which is to run Robust-ILQG for a short horizon repeatedly. We use this approach in our second experiment in Section 4.2.

4 Experiments and Analysis

We consider two simulated tasks: cart-pole swing-up and quadrotor control. For each task, one set of model parameters was used as the "simulator", and another set of model parameters was used

as the "real-world." Multiplicative noise was added to dynamics and Gaussian noise was added to observation. We compare GP-ILQG's performance with three optimal control algorithms: (1) ILQG using the "real-world" model, (2) ILQG using the incorrect "simulator" model, (3) Probabilistic DDP (PDDP) [13], which is a variant of ILQG that relies only on data. For both GP-ILQG and PDDP, the same set of data and same GP implementation were used at each iteration.

As data gets large, computing Gaussian Process becomes computationally expensive, and thus it is common to use a subset of the data for computing the mean and covariance [17]. In our experiments, we keep all observations and uniformly sub-sample 300 data points at each iteration[6]. GP hyperparameters are optimized with the Gaussian Process for Machine Learning Toolbox [18]. If the learner's prediction error for a validation set is higher than that of its previous learner, we re-sample and re-train.

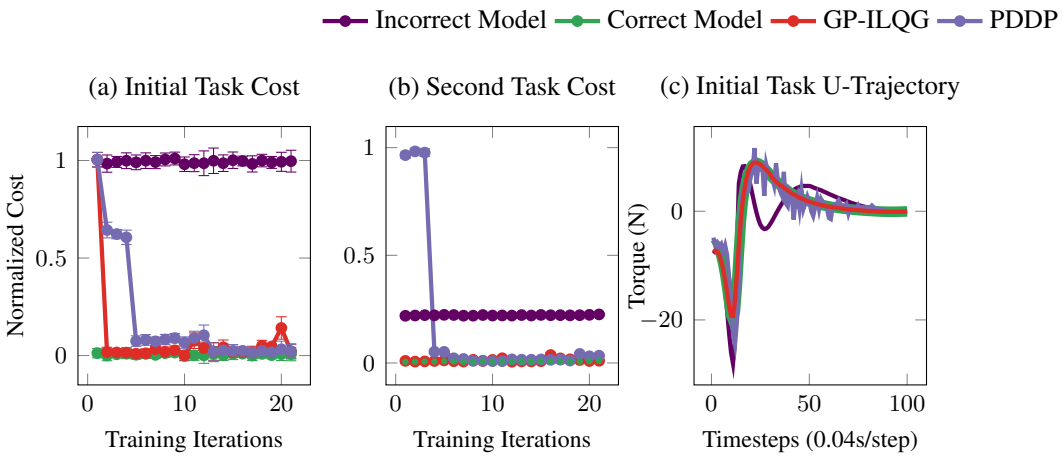

Figure 3: Cartpole swing-up.

4.1 Cart-Pole Swing-Up

In the cart-pole problem, the state is defined as $[x, \dot{x}, \theta, \dot{\theta}]$, where x is the position of the cart along the $x-$axis and θ is the angle of the pole from the vertical downright position. Control input is the x-directional force (N).

The model parameters for the "simulator" and "real-world" model are given in Section 6. The real-world model has a 30% longer pole.

We run two tasks in this experiment. In the first task, the initial state is $[0, \pi/4, 0, 0]$. Figure 3(a) is the normalized cost for the first task. While both GP-ILQG and PDDP converges to the optimal performance, GP-ILQG is converges much quickly, within the first 2 iterations.

The difference between GP-ILQG and PDDP is more noticeable in the second task (Figure 3(b)), which starts from a different initial state $[0, -\pi/4, 0, 0]$. Both GP-ILQG and PDDP use the learner used in the previous task, but the initial cost for PDDP is significantly higher than GP-ILQG. We believe that this is because both algorithms explore an unexplored region in the first few iterations. While GP-ILQG relies on the simulator's inaccurate model in the unexplored region, PDDP has no information to make meaningful advancement until enough data is collected.

What is more noticeable is the improved performance of GP-ILQG over the simulator-based ILQG. The latter's suboptimal policy results in significantly higher cost in general. Figure 3(c) shows the control sequences generated by the final policies of the four algorithms. GP-ILQG's control sequence is almost identical to the optimal sequence, and PDDP closely follows the optimal sequence as well. However, the simulator-based ILQG's control sequence is quite different due to its incorrect model.

[6]For better results, more advanced techniques such as Sparse Pseudo-input GP [19] or Sparse Spectral Gaussian Process [16] can be used.

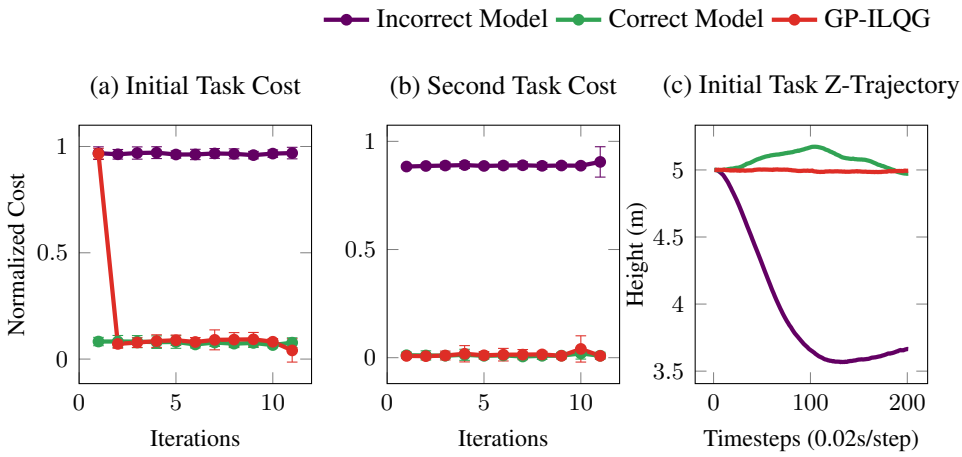

Figure 4: Quadrotor control

4.2 Quadrotor

We use the quadrotor model introduced in [23]. The model has 12-dimensional state, $\mathbf{x} = [\mathbf{p}, \mathbf{v}, \mathbf{r}, \mathbf{w}]^\top$, where $\mathbf{p}$ (m) and $\mathbf{v}$ (m/s) refers to the quadrotor's position and velocity in 3D space, $\mathbf{r}$ is orientation (rotation about axis $\mathbf{r}$ by angle $\|r\|$), and $\mathbf{w}$ (rad/s) is angular velocity. It has 4 control inputs, $\mathbf{u} = [u_1, u_2, u_3, u_4]^\top$, which represent the force (N) exerted by the four rotors. The dynamics is given as the following:

$$
\begin{aligned}
\dot{\mathbf{p}} &= \mathbf{v} \\
\dot{\mathbf{v}} &= -g\mathbf{e_3} + (\sum u_i)\exp([\mathbf{r}]\mathbf{e}_3 - k_v\mathbf{v})/m \\
\dot{\mathbf{r}} &= \mathbf{w} + \frac{1}{2}[\mathbf{r}]\mathbf{w} + (1 - \frac{1}{2}\|\mathbf{r}\|/\tan(\frac{1}{2}\|r\|)[\mathbf{r}]^2/\|\mathbf{r}\|^2 \\
\dot{\mathbf{w}} &= J^{-1}(\rho(u_2 - u_4)\mathbf{e}_1 + \rho(u_3 - u_1)\mathbf{e}_2 \\
&\quad + k_m(u_1 - u_2 + u_3 - u_4)\mathbf{e}_3 - [\mathbf{w}]J\mathbf{w})
\end{aligned}
$$

where $\mathbf{e}_i$ are the standard basis vectors, $g = 9.8m/s^2$ is gravity, k_v is the drag coefficient of rotors, m (kg) is the mass, J (kg m^2) is the moment of inertia matrix, and ρ (m) is the distance between the center of mass and the center of rotors, and k_m is a constant relating the force of rotors to its torque. $[\cdot]$ refers to the skew-symmetric cross product. The model parameters used are included in Section 6. The real-world model is 40% heavier than the simulator's model.

We evaluate the performance of two tasks. The quadrotor starts at a position near $(0m, 0m, 5m)$ with zero velocity. In the first task, the goal is to move the quadrotor forward to $(4m, 0m, 5m)$ in 4 seconds. The second task is to drive the quadrotor to $(2m, 1m, 7m)$ in 4 seconds. The cost function was set to track a straight line from the initial position to the goal, with higher cost for maintaining the height.

In this experiment, we were not able to run PDDP to convergence with the same data used in GP-ILQG. We believe that this arises from the same problem we saw in the second task of cart-pole: PDDP has insufficient data to infer the unexplored state-space. We note that the original PDDP algorithm requires random trajectories as its initial data set instead of random variations of a single nominal trajectory. While our experiment does not indicate that PDDP is incapable of this task[7], it highlights our algorithm's data efficiency. Even with the small set of task-specific data, GP-ILQG converges in the first two iterations in the initial task (Figure 4(a) and converges immediately to the optimal policy in the second task (Figure 4(b)). Figure 4(c) compares the trajectories generated by the three algorithms. It shows that while our algorithm closely tracks the desired height, the simulator's suboptimal controller fails to recover from the vertical drop due to its incorrect mass model.

[7] A similar experiment with quadrotor control was shown to be successful in [15].

5 Conclusion

In this paper, we proposed a novel algorithm that combines real-world data with a simulator's model to improve real-world performance of simulation-based optimal control. Our approach uses a Gaussian Process to correct a simulator's nonlinear model bias beyond the scope of its model space while incorporating the uncertainty of our estimate in computing a robust optimal control policy. Through simulated experiments, we have shown that our approach converges to the optimal performance within a few iterations and is capable of generalizing the learned dynamics for new tasks.

Although our algorithm is capable of correcting significant model errors, it is limited by the quality of the initial policy based on the simulator's incorrect model. For example, the simulator's model can be sufficiently different from the true model such that the initial policy results in catastrophic damage to the robot. Our algorithm is incapable of measuring this initial uncertainty, although it can be improved by providing an initial set of expert-generated trajectories. We leave this as a future research direction.

6 Appendix

6.1 Stochasticity and uncertainty terms

We define the partial derivatives for stochasticity and uncertainty terms as the following:

$$\mathsf{F}_{\mathbf{x}} \otimes \delta\mathbf{x} \triangleq \left[\mathsf{F}_{\mathbf{x}}^{(1)}\,\delta\mathbf{x}, ..., \mathsf{F}_{\mathbf{x}}^{(p)}\,\delta\mathbf{x}\right] \tag{11}$$

$$\mathsf{G}_{\mathbf{x}} \otimes \delta\mathbf{x} \triangleq \left[\mathsf{G}_{\mathbf{x}}^{(1)}\,\delta\mathbf{x}, ..., \mathsf{G}_{\mathbf{x}}^{(n)}\,\delta\mathbf{x}\right] \tag{12}$$

where $\mathsf{F}_{\mathbf{x}}^{(i)}, \mathsf{G}_{\mathbf{x}}^{(j)}$ refer to the partial derivatives of i, j-th columns of F and G with respect to $\mathbf{x}$.

6.2 Models used in experiments

We have used the following model parameters for the cart-pole and quadrotor experiments. "Simulator" refers to the incorrect model in the simulator. "Real World" is the model used during rollouts and policy evaluation; its model parameters are not revealed to our algorithm.

6.2.1 Cartpole

	Simulator	Real World
Cart Mass	1 kg	1 kg
Pole Mass	1 kg	1 kg
Pole Length	**1 m**	**1.3 m**

6.2.2 Quadrotor

k_v is a constant relating the velocity to an opposite force, caused by rotor drag and induced inflow. m (kg) is the mass, J (kg m^2) is the moment of inertia matrix, ρ (m) is the distance between the center of mass and the center of the rotors.

	Simulator	Real World
k_v	0.15	0.15
k_m	0.025	0.025
m	**0.5**	**0.7**
J	0.05 I	0.05 I
ρ	0.17	0.17

References

[1] P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In W. W. Cohen and A. Moore, editors, *Proceedings of the 23th International Conference on*

Machine Learning (ICML-06), pages 1–8, 2006. URL `http://www.machinelearning.org/proceedings/icml2006/001_Using_Inaccurate_Mod.pdf`.

[2] C. G. Atkeson, A. W. Moorey, and S. Schaalz. Locally weighted learning. *Artif Intell Rev*, 11 (1-5):11–73, 1997.

[3] A. Boeing and T. Bräunl. Leveraging multiple simulators for crossing the reality gap. In *Control Automation Robotics & Vision (ICARCV), 2012 12th International Conference on*, pages 1113–1119. IEEE, 2012.

[4] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[5] D. Huh and E. Todorov. Real-time motor control using recurrent neural networks. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pages 42–49. IEEE, 2009.

[6] D. H. Jacobson and D. Q. Mayne. Differential dynamic programming. 1970.

[7] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[8] S. Levine and V. Koltun. Guided policy search. In *ICML (3)*, pages 1–9, 2013.

[9] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems.

[10] D. Mitrovic, S. Klanke, and S. Vijayakumar. Adaptive optimal feedback control with learned internal dynamics models. In *From Motor Learning to Interaction Learning in Robots*, pages 65–84. Springer, 2010.

[11] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 242–248. IEEE, 2016.

[12] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on neural networks*, 1(1):4–27, 1990.

[13] Y. Pan and E. Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems*, pages 1907–1915, 2014.

[14] Y. Pan and E. A. Theodorou. Data-driven differential dynamic programming using gaussian processes. In *American Control Conference (ACC), 2015*, pages 4467–4472. IEEE, 2015.

[15] Y. Pan, X. Yan, E. Theodorou, and B. Boots. Scalable reinforcement learning via trajectory optimization and approximate gaussian process regression. *NIPS Workshop on Advances in Approximate Bayesian Inference*, 2015.

[16] J. Quiñonero-Candela, C. E. Rasmussen, A. R. Figueiras-Vidal, et al. Sparse spectrum gaussian process regression. *Journal of Machine Learning Research*, 11(Jun):1865–1881, 2010.

[17] C. E. Rasmussen. Gaussian processes for machine learning. 2006.

[18] C. E. Rasmussen and H. Nickisch. Gaussian processes for machine learning (gpml) toolbox. *Journal of Machine Learning Research*, 11(Nov):3011–3015, 2010.

[19] E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in neural information processing systems*, 18:1257, 2006.

[20] A. A. Stoorvogel. The h-infinity control problem: A state space approach. 1993.

[21] J. Tan, Z. Xie, B. Boots, and C. K. Liu. Simulation-based design of dynamic controllers for humanoid balancing. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 2729–2736. IEEE, 2016.

[22] E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.

[23] J. van den Berg. Extended lqr: Locally-optimal feedback control for systems with non-linear dynamics and non-quadratic cost. In *Robotics Research*, pages 39–56. Springer, 2016.

[24] J. M. Wang, D. J. Fleet, and A. Hertzmann. Optimizing walking controllers for uncertain inputs and environments. In *ACM Transactions on Graphics (TOG)*, volume 29, page 73. ACM, 2010.

[25] A. Yamaguchi and C. G. Atkeson. Differential dynamic programming with temporally decomposed dynamics. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 696–703. IEEE, 2015.

[26] A. Yamaguchi and C. G. Atkeson. Neural networks and differential dynamic programming for reinforcement learning problems. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5434–5441. IEEE, 2016.

[27] J. C. Zagal, J. Ruiz-del Solar, and P. Vallejos. Back to reality: Crossing the reality gap in evolutionary robotics. In *IAV 2004 the 5th IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal*, 2004.

[28] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 528–535. IEEE, 2016.