Control Adaptation via Meta-Learning Dynamics

James Harrison*,1, Apoorva Sharma*,1, Robert Dyro1, Xinrui Wang1, Roberto Calandra2, Marco Pavone1

Abstract— **Deep neural networks are an increasingly popular approach for dynamics learning for robotic systems, for use planning and control. When these models are applied on systems different from those on which they were trained, model mismatch can lead to degraded performance, instability, and loss of control. These issues can be alleviated by performing online adaptation of the model as state transition data from the true system is observed during operation. However, standard neural network models are ill-suited to the data-efficient and computationally efficient adaption that is necessary for adaptive control. In this work, we present Control Adaptation via Meta-Learning Dynamics (CAMeLiD), an approach for nonlinear adaptive trajectory optimization and control which leverages recent advances in meta-learning to enable the efficient adaptation of neural network models. Our approach extends the ALPaCA meta-learning model, and enables flexible and adaptive learning-based dynamics models. We pair this dynamics model with a model predictive controller that is designed for efficient operation with neural network dynamics models, and show that CAMeLiD outperforms state-of-the-art meta-learning-based adaptive control models on two continuous control tasks.**

I. Introduction

In recent years, deep neural networks are increasingly being used within model-based robotic control and planning, serving as data-driven approximations of system dynamics [1], [2], [3], [4]. While hyper-parametric neural network models are attractive as universal approximators that can fit arbitrary nonlinear dynamics, these models typically require large amounts of data to train, which can be expensive or tedious to obtain on a single robot of interest. Approaches to alleviating this problem include transferring from simulation [5], or large-scale data collection across multiple robots [6]. However, both involve training a model on data from a source that is different from the target application and thus are prone to issues arising from model mismatch.

This problem of model mismatch can be addressed through an adaptive control approach, wherein the state transitions observed during operation on the target system are used to adjust the model to better fit the true system dynamics. For this purpose, we not only desire a regression model that is *high-capacity*, in that it is able to capture nonlinear dynamics; but also one that is *data-efficient*, so that it can extract the maximal information about the true dynamics from the observed data. Further, in many robotic systems, it is critical to adapt quickly, during operation, so the adaptation should be *computationally efficient*. Typical neural network models fall short on these latter desiderata, as they require large amounts of data and several epochs of gradient descent steps to fit new data.

Recently, *meta-learning* has emerged as a promising approach to addressing these issues and has seen success in few-shot image classification and reinforcement learning, where deep networks are adapted from limited amounts of data at test-time [7]. In this work, we present a novel approach to nonlinear adaptive planning and control using a Bayesian meta-learned dynamics model. Our approach, Control Adaptation via Meta-Learning Dynamics (CAMeLiD), combines a model-predictive control scheme with the ALPaCA meta-learning model proposed in [8]. ALPaCA learns network weights such that it can efficiently adapt at test time through analytic, Bayesian recursive least-squares updates to the last-layer weights of the network.

Contributions: The contributions of this work are as follows. First, we extend the ALPaCA model with a multi-head architecture and demonstrate that this design allows better modeling relative to the original ALPaCA model. Next, we present CAMeLiD as a system-level framework for meta-learning for adaptive control and demonstrate that this outperforms state-of-the-art for meta-learning adaptive control. Finally, we conclude with a discussion of the general utility of the Bayesian modeling approach presented herein for the robotic control and planning context.

II. Related Work

Model learning is a core component of many robotic systems [9], and can be performed using a variety of strategies—system identification typically implies learning the model before acting [10], adaptive control implies learning and acting simultaneously over the course of one episode [11], and model-based reinforcement learning often implies interleaving learning and acting in an episodic fashion [12]. In this work, we consider a combination of these perspectives, in which prior experience allows a robotic system to learn strong priors, which are then adapted online over the course of one episode.

Adaptive control via Gaussian process modeling: Similarly to this paper, a body of work starting in the 1990s and the early 2000s has investigated adaptive control using Gaussian process models and radial basis function (RBF) networks. Murray-Smith et al. [13], [14] investigated adaptive control using non-parametric GP models in both the greedy (one-step ahead) and multi-step setting. They took a "cautious" or risk-sensitive approach, in which the optimization objective was the mean squared deviation from

*These authors contributed equally. Order determined randomly.

1Stanford University, Stanford, CA.
{jharrison, apoorva, rdyro, xinrui, pavone}@stanford.edu.
2Facebook AI Research, Menlo Park, CA. rcalandra@fb.com.

a reference trajectory, plus a variance penalty. Their multi-step optimization approach relied on gradient descent on the cost function, which is expensive with kernel GPs. Indeed, it is unlikely that this approach could be run online for anything but low-dimensional systems, or with more than a very small amount of training data. In contrast, our approach relies on a parametric GP and associated derivatives, for which model evaluation consists simply of evaluating a small neural network. In comparing non-parametric models to parametric, [14] claim that the relative advantage of non-parametric models is a representation of the model that depends on local data density. However, our model learns a structured representation of uncertainty via meta-learning. These learned correlations are capable of expressing uncertainty that is based on local data density if they are warranted for the class of systems under investigation, but are also able to learn richer correlations.

The model predictive control-based approach was extended to include constraints in [15], but computational efficiency was not addressed and the online optimization problem is constrained and nonconvex. More recently, Chowdhary et al. [16] leveraged GPs for model-reference adaptive control (MRAC) and achieved good online performance. However, this approach relies on selectively discarding data to reduce the computational expense of inference. In contrast, our parametric approach is capable of incorporating arbitrarily large amounts of data via recursive least squares, which results in constant complexity for inference and conditioning. As such, our proposed method has substantial computational efficiency advantages over kernel GPs and is better suited to online adaptive control. Note that application of GPs in adaptive control is a large subfield within the adaptive control literature, and there are many approaches that we have not surveyed here.

In addition to work on non-parametric GPs in adaptive control, there is a substantial body of work utilizing neural networks and parametric GP models. Early work on adaptive control with neural network models was presented in [17], in which a RBF network was used for direct adaptive control in the continuous time setting. Indeed, use of nonlinear basis functions (such as RBF networks) with linear adaptation has been one of the dominant approaches to both direct and indirect adaptive control in recent years [18], [19], [20]. Particularly relevant to our work, [21] use neural network dynamics models and perform Bayesian recursive least squares on the last layer to perform adaptive control. We apply this idea within the meta-learning setting, and thus empirically learn a prior that is optimized over the offline training data. Moreover, whereas their control law is a simple heuristic, we perform adaptive planning within the model predictive control loop.

Meta-learning for adaptive control: Closely related to our work is the literature framing model-based reinforcement learning as a meta-learning task [22], [23]. In [22], the authors investigated use of recurrent and gradient-based meta-learning approaches for the purpose of intra-episode dynamics adaptation. While this work is similar to the approach presented here, they use sampling-based MPC as opposed to a gradient-based control strategy. In addition, their approach is based on performing gradient descent on their dynamics neural network (trained using MAML [7]). A key advantage of MAML is that online adaptation is optimizing the full network via gradient descent, so in the limit of infinite data it can adapt to fit any arbitrary system at test time. However, to keep the evaluation of the model efficient at runtime, [22] propose only keeping a sliding window of past observations. They then condition their original prior on only these data by taking a few gradient steps, so the system never enters the regime where this generality of MAML is useful. In contrast, our approach restricts online adaptation to the last layer of network. While this limits the flexibility of our online adaption, it enables efficient recursive update rules that directly solve the optimization problem associated with posterior computation, summarizing all the observed data without needing to store it and process it when making predictions. We compare to [22] directly in our experiments and show that our approach is more effective in several domains.

In [23], the authors instead use a GP dynamics model with a latent variable governing context. However, their approach is limited by the high computational expense associated with non-parametric GP regression, whereas we take a parametric approach that is more efficient to use online. Additionally, their approach is focused on adaptation over the course of multiple episodes, whereas our approach is focused on adaptive control over the course of one episode.

Overall, our approach leverages a meta-learned dynamics model that can be viewed as a parametric Gaussian process at test-time. While this limits the degree to which we can adapt to *arbitrary* systems at runtime as compared to non-parametric GP or meta-learning approaches that optimize the entire network at test time, in return we enjoy improved computational complexity and efficient online posterior inference through the analytic recursive least-squares updates. In our experiments, we demonstrate that this can enable better system-level performance of the nonlinear adaptive control framework.

III. Problem Statement

We aim to control an unknown system, from which we have observed trajectory data with varying (unobservable) latent parameters. For example, if we have a delivery quadrotor, we may observe flight data for the vehicle under different loading conditions. By using efficient adaptive models that are capable of incorporating prior information, we wish to adaptively plan for and control this system, even without having knowledge about the exact payload being transported. Formally, we consider a discrete-time dynamical system of the form

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}) + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \Sigma_\epsilon), \tag{1}$$

where $\boldsymbol{f}(\cdot, \cdot; \cdot)$ is a continuous nonlinear function, $\boldsymbol{x} \in \mathcal{X}$ denotes the state, and $\boldsymbol{u} \in \mathcal{U}$ is the action. The dynamics are parameterized by latent parameters $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$. We assume the cost function, $\text{Cost}(\boldsymbol{x}, \boldsymbol{u})$ is known. We assume interaction with the system is episodic, and $\boldsymbol{\theta}$ is sampled randomly at the beginning of each episode and is fixed over the episode. Critically, however, we do not assume direct knowledge of $\boldsymbol{f}$, $\boldsymbol{\theta}$, or $p(\boldsymbol{\theta})$. Instead, we additionally assume access to trajectory data $\mathcal{D} = \{D_\tau(\boldsymbol{\theta}_j)\}_{j=1}^{J}$, where $D_\tau(\boldsymbol{\theta}_j) = \{(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1})\}_{t=0}^{\tau}$. From this data, we wish to learn an approximate model for both the dynamics function $\boldsymbol{f}$, as well as an approximate prior over $\boldsymbol{\theta}$. This setting is similar to the standard system identification setting, although $\boldsymbol{\theta}$ is not explicitly known to the system designer as it may be in system identification. Our approach only requires samples from the system dynamics $\boldsymbol{f}$; it may be used with systems for which analytical models are not available.

Given this dataset $\mathcal{D}$, an episode horizon T, an initial state distribution $p(\boldsymbol{x}_0)$, the system-level control problem can be formulated as

$$\min_{\boldsymbol{\pi}_{0:T} \in \Pi} \mathbb{E}_{\boldsymbol{\theta} \sim p(\boldsymbol{\theta}), \boldsymbol{x}_0 \sim p(\boldsymbol{x}_0)} \left[\sum_{t=0}^{T} \text{Cost}\left(\boldsymbol{x}_t, \boldsymbol{\pi}_t(\boldsymbol{x}_t)\right) \right], \tag{2}$$

and subject to the dynamics of Eq. 1, where we write $\boldsymbol{\pi}_t$ to denote a time-dependent policy, and Π to denote the set of possible policies. While this notation is more commonly used in the model-free literature, we take a model-based approach in this paper. The problem we have presented can also be seen as a generalization of the standard adaptive control problem [11] to the “multi-task” case [24]. Alternatively, it may be interpreted as the meta-reinforcement learning problem [25], [26], [27] in which the cost function is known. This problem setting is a Bayes-adaptive Markov Decision Process (BAMDP), a specific form of POMDP in which the unobserved variables are not time-varying [28].

IV. Background

ALPaCA [8] is an efficient Bayesian approach to function regression in a meta-learning/multi-task learning setting. In this formulation, function regression is performed by using observed data to update a posterior belief over a family of possible functions. As a Bayesian approach, ALPaCA maintains a belief over functions, and updates this belief via Bayes' Rule as samples of the function are observed. ALPaCA approximates this true distribution over functions by maintaining a distribution over the parameters of a neural network model $K\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{w})$, where $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{w})$ represents a feed-forward neural network with weights $\boldsymbol{w}$ and output dimension n_ϕ, and K is a $n_x \times n_\phi$ matrix which can be thought of as the linear last-layer weights of the network. ALPaCA only maintains a delta distribution over $\boldsymbol{w}$, but a full matrix-normal distribution on the last-layer weights $K \sim \mathcal{MN}(\bar{K}_0, \Sigma_\epsilon, \Lambda_0^{-1})$. This particular form reduces the task of online function regression to Bayesian linear regression in the feature space defined by $\boldsymbol{\phi}$.

Given a set of observed transition tuples $D_{\text{context}} = \{(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau, \boldsymbol{x}_{\tau+1})\}_{\tau=0}^{t-1}$ from a single fixed dynamical system $\boldsymbol{f}(\cdot; \boldsymbol{\theta})$, we can compute a posterior over the last-layer weights K, given by $p(K \mid X_t, \Phi_{t-1}) = \mathcal{MN}(\bar{K}_t, \Sigma_\epsilon, \Lambda_t^{-1},)$, where

$$\Lambda_t = \Phi_{t-1}^T \Phi_{t-1} + \Lambda_0 \,, \tag{3}$$

$$\bar{K}_t^T = \Lambda_t^{-1}(\Phi_{t-1} X_t + \Lambda_0 \bar{K}_0^T) \,, \tag{4}$$

with $X_t^T = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t] \in \mathbb{R}^{n_x \times t}$ and $\Phi_{t-1}^T = [\boldsymbol{\phi}(\boldsymbol{x}_0, \boldsymbol{u}_0), \ldots, \boldsymbol{\phi}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})] \in \mathbb{R}^{n_\phi \times t}$. We assume a Gaussian error model, giving rise to a Gaussian posterior predictive density $\hat{\boldsymbol{x}}_{t+1} \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1})$ where

$$\boldsymbol{\mu}_{t+1} = \bar{K}_t \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{w}) \,, \tag{5}$$

$$\Sigma_{t+1} = (1 + \boldsymbol{\phi}^T(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{w}) \Lambda_t^{-1} \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{w})) \Sigma_\epsilon \,. \tag{6}$$

A natural measure of the quality of this model is how well its posterior predictive density models future transition tuples drawn from the same system. If we have a set of query points $D_{\text{query}} = \{(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau, \boldsymbol{x}_{\tau+1})\}_{\tau=t}^{T}$ from the same system, we can evaluate the negative log likelihood of these points under the posterior predictive density (Eq. 5 and 6) to obtain a loss function. As the posterior predictive density parameters are computed via analytic and differentiable operations, we can optimize the weights $\boldsymbol{w}$ and the prior parameters $\bar{K}_0, \Lambda_0^{-1}$ via stochastic gradient descent. We train on a meta-dataset of datasets $\mathcal{D} = \{(D_{\text{context}}^{(i)}, D_{\text{query}}^{(i)})\}_{i=1}^{M}$, where the transitions in $D_{\text{context}}^{(i)}$ and $D_{\text{query}}^{(i)}$ are sampled from the dynamical system $\boldsymbol{f}(\cdot; \boldsymbol{\theta}_i)$, and $\boldsymbol{\theta}_i$ is assumed to be representative samples from the prior $p(\boldsymbol{\theta})$. In this way, the model is able to capture the structure of the dynamical system in the neural network features $\boldsymbol{\phi}$, and express the effects of the parameter uncertainty through the prior on K.

V. A More Expressive Meta-Learning Model

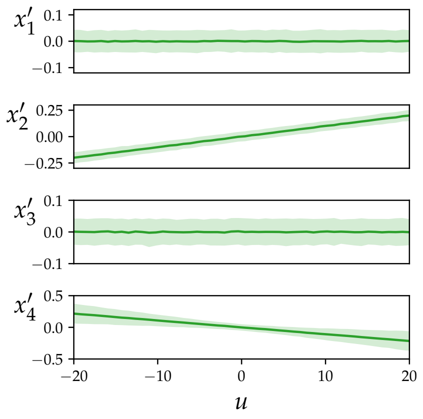
While ALPaCA provides a heteroskedastic posterior predictive model, the posterior predictive covariance is restricted to be a scalar multiple of the error covariance Σ_ϵ. However, in many dynamical systems, the shape of the uncertainty in the transition function is often different across different dimensions. For example, systems where both position and velocities are observed, the position at the next timestep is only a function of the current velocity, while the next velocity may depend on the uncertain latent parameters, and thus have a different shape of uncertainty. Quantitative results are available in the appendix. To correct this, we propose making the modeling assumption that the uncertainty of each output dimension of f is independent. This allows us to use a separate, single-variable Bayesian linear regression model for each output dimension. These can be throught of as distinct “heads” of the model, hence we refer to it as a multi-head ALPaCA model. Formally, we can write the model and prior as

$$\boldsymbol{x}_{t+1} = K \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon} \,, \tag{7}$$

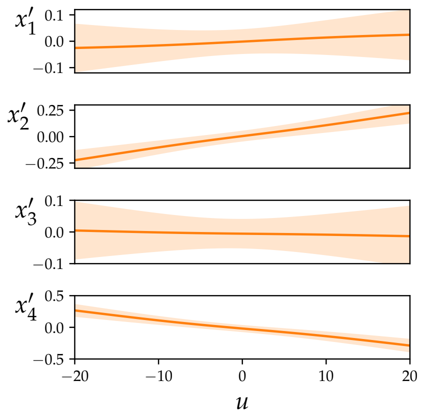
$$\boldsymbol{k}_i \sim \mathcal{N}(\bar{\boldsymbol{k}}_{i,0}, \sigma_i^2 \Lambda_{i,0}^{-1}) \,, \qquad \forall i = 1, \ldots, n_x \,, \tag{8}$$

$$\epsilon_i \sim \mathcal{N}(0, \sigma_i^2) \,, \qquad \forall i = 1, \ldots, n_x \,, \tag{9}$$

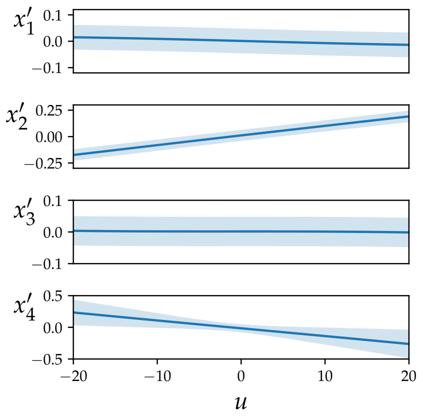
where each row $\boldsymbol{k}_i$ of the weight matrix K is drawn from an independent Gaussian distribution, with mean $\bar{\boldsymbol{k}}_{i,0}$ and covariance $\sigma_i^2 \Lambda_{i,0}^{-1}$, and each output dimension is corrupted with independent additive Gaussian noise.



(a) True System



(b) Single Prior [8]



(c) Multi-Head (Ours)

Fig. 1: Mean and 95% confidence intervals of, from left to right, the true transition function $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{0}, \boldsymbol{u}; \boldsymbol{\theta})$ with $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$ for the Cart-Pole system, the approximation of the ALPaCA model with a single prior on the last layer, and our variation using approximation from the multi-head ALPaCA model. Multi-head ALPaCA more accurately models the underlying function.

These independence assumptions allow computing the posterior distribution on each $\boldsymbol{k}_i$ separately. Concretely, given D_{context}, the posterior distributions for each output dimension i are given by $\mathcal{N}(\bar{\boldsymbol{k}}_{i,t}, \sigma_i^2 \Lambda_{i,t}^{-1})$, where

$$\Lambda_{i,t} = \boldsymbol{\phi}_{t-1}^T \Phi_{t-1} + \Lambda_{i,0}, \tag{10}$$

$$\bar{\boldsymbol{k}}_{i,t} = \Lambda_{i,t}^{-1}(\Phi_{t-1} X_{i,t} + \Lambda_{i,0} \bar{\boldsymbol{k}}_{i,0}), \tag{11}$$

$\Phi_t \in \mathbb{R}^{n_\phi \times t}$ is unchanged from before, and $X_{i,t} \in \mathbb{R}^t$ the vector containing dimension i of the next state for each of the t transition tuples in D_{context}. The posterior predictive for this model remains of the same form as in Eq. 5 and 6, where now the covariance is given by a diagonal matrix Σ_t with elements

$$(\Sigma_{t+1})_{ii} = \left(1 + \boldsymbol{\phi}^T(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{w}) \Lambda_{i,t}^{-1} \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{w})\right) \sigma_i^2 \,. \tag{12}$$

In the adaptive control context where context data is observed sequentially as the agent interacts with the true system, it is useful to leverage recursive updates of the posterior, as detailed in [8]. In our improved model, parameters of the posterior at time t can be computed given observed transition $(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}, \boldsymbol{x}_t)$ and the values of those parameters at the previous timestep. For each state dimension i, this update rule is

$$\begin{aligned} \Lambda_{i,t}^{-1} &= \Lambda_{i,t-1}^{-1} - \frac{1}{1 + \boldsymbol{\phi}_{t-1}^T \Lambda_{i,t-1}^{-1} \boldsymbol{\phi}_{t-1}} \Lambda_{i,t-1}^{-1} \boldsymbol{\phi}_{t-1} \boldsymbol{\phi}_{t-1}^T \Lambda_{i,t-1}^{-1} \,, \\ \boldsymbol{q}_{i,t} &= (\boldsymbol{x}_t)_i \boldsymbol{\phi}_{t-1} + \boldsymbol{q}_{i,t-1} \,, \\ \bar{\boldsymbol{k}}_{i,t} &= \Lambda_{i,t}^{-1} \boldsymbol{q}_{i,t} \,, \end{aligned} \tag{13}$$

with each $\boldsymbol{q}_i \in \mathbb{R}^{n_\phi}$ is initialized to be $\boldsymbol{q}_i = \Lambda_0 \bar{\boldsymbol{k}}_{i,0}$, and $\boldsymbol{\phi}_{t-1}$ is shorthand for $\boldsymbol{\phi}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})$.

This choice of variance model corresponds to a structural assumption on the covariance of $\text{vec}(K)$. The approach taken in [8] was to model K with a matrix-normal distribution, which corresponds to $\text{vec}(K) \sim \mathcal{N}(\text{vec}(\bar{K}), \Sigma_\epsilon \otimes \Lambda^{-1})$. Note that the full (unstructured) covariance of $\text{vec}(K)$ contains $n_\phi^2 n_x^2$ parameters. For a moderately large system (say, $n_x =$ 12) with 128 basis functions, this would require learning more than 2.3 million matrix entries. In contrast, the matrix normal factorization requires learning only n_ϕ^2 entries, which in this case corresponds to 0.7% of the parameters. However, this relies on Λ^{-1} being shared for each output dimension, which is typically overly restrictive. The approach we present here corresponds to a block-diagonal covariance, wherein $\text{vec}(K) \sim \mathcal{N}(\text{vec}(\bar{K}), \text{diag}([\Lambda_1^{-1}, \ldots, \Lambda_{n_x}^{-1}]))$. This factorization requires n_x RLS updates, and maintains $n_x n_\phi^2$ parameters. In comparison to a fully dense covariance matrix, we only sacrifice the ability to handle off-diagonal elements in the noise covariance, which are often ignored in the modeling process anyway. Moreover, this choice of covariance factorization allows the RLS update to be performed via n_x independent updates with the precision matrices for each dimension, substantially reducing computational complexity. An extended discussion of the covariance factorization is presented in the extended version of this paper, available at [29].

Experimental validation: We trained both the model from [8] as well as a model with the dimension specific priors on a cart-pole system with randomized parameters (details of the randomization are provided in the experiments section). Apart from the last layer structure, the models were identical. Fig. 1 visualizes each dimension of the next state when different control actions are applied to the system initially at $\boldsymbol{x}_0 = \boldsymbol{0}$. We see that the uncertainty in the parameters influences the transition in different ways for each dimension. It is evident that by having a separate prior for each output dimension, the model is better able to capture the true uncertainty structure in the dynamics. We observed that this added flexibility improved training stability, and led to improved quantitative performance as well: on the cart-pole system the multi-head ALPaCA obtains a NLL of -24.8, while the baseline ALPaCA only attains -22.67. We find that the multi-head alpaca model outperforms the single head version with only a minor increase in computational complexity, and so we use the multi-head ALPaCA model for all of our experiments, and henceforth use ALPaCA to refer to this multi-head version.

VI. Control Adaptation via Meta-Learning Dynamics

Our proposed framework for meta-learning-enabled nonlinear adaptive planning and control consists of two phases. First, *offline*, we use a meta-dataset of transitions drawn from a distribution of dynamical systems to optimize an adaptive dynamics model. Then, *online*, we use this adaptive model in a model predictive control framework, while adapting the model as data is observed. The high level operation of both of these phases is outlined in Alg. 1; in the following sections, we discuss each phase in more detail.

Offline Meta-learning of Dynamics: To efficiently adapt as transition data is observed online, it is necessary to leverage prior knowledge about the dynamical system expected to be seen in the online phase. In this work, we assume that this prior knowledge comes in the form of a meta-dataset of datasets as discussed in the previous section $\mathcal{D} = \{(D^{(i)}_{\text{context}}, D^{(i)}_{\text{query}})\}_{i=1}^{M}$, drawn from the distribution of dynamical systems we expect to see online. This meta-dataset can be gathered in different ways: for example, it could be constructed from past interactions with dynamical systems of the same class, or generated by randomizing parameters of a simulator in physically realistic ways. In our experiments, we sample from the meta-dataset by first sampling the parameters $\boldsymbol{\theta}$ of the simulator from a prior distribution $p(\boldsymbol{\theta})$, and then sampling state transitions from the simulator. More details on the data collection process for each of our experiments is available in the appendix. Given this meta-dataset, the model is trained by minimizing the negative log likelihood of D_{query} under the posterior predictive density when conditioned on D_{context}, as in [8].

Online Adaptive Model Predictive Control: At test time, the learned adaptive model can be applied within a model-predictive control framework to control an uncertain system. At each timestep, the agent takes an action and uses the observed transition to update the posterior parameters through the recursive updates in Eq. 13. To compute which action to take, the agent uses the current model estimate to optimize a finite horizon control strategy. Formally, at time t, the agent optimizes for a horizon H using the current model estimate $\bar{K}_t$ such that

$$\begin{aligned} \min_{\pi\in\Pi} \quad & \sum_{\tau=0}^{H} \text{Cost}(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau)\,, \\ \text{subject to} \quad & \boldsymbol{x}_{\tau+1} = \bar{K}_t \boldsymbol{\phi}(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau; \boldsymbol{w})\,, \\ & \boldsymbol{u}_\tau = \boldsymbol{\pi}_\tau(\boldsymbol{x}_\tau)\,, \\ & \boldsymbol{u}_\tau \in \mathcal{U}\,. \end{aligned} \tag{14}$$

In this work, we solve this nonlinear optimal control problem using a simple sequential convex programming-based model predictive control (SCP MPC) algorithm. This controller sequentially linearizes the dynamics around a nominal trajectory, and uses these linearized dynamics to solve a local control problem. To ensure that the linearized dynamics are reasonable, we enforce penalties on the state and control deviation from the nominal trajectory. Following the solution to this penalized local problem, the dynamics are relinearized and the problem is re-solved, and this is repeated until convergence. More discussion of this control scheme is available in the appendix. The agent takes h steps from the resulting policy before recomputing a policy with the updated model. Since the ALPaCA dynamics model is pretrained to adapt quickly to fit the true observed transitions, the computed policies become increasingly effective on the true dynamical system.

Algorithm 1 CAMeLiD

```
Require: training data $\mathcal{D}$, replanning frequency $h$, MPC horizon $H$
    OFFLINE META-LEARNING:
 1: Train ALPaCA model to get $\bar{K}_0, \{\Lambda_{i,0}\}_{i=1}^{n_x}, \boldsymbol{w}$
    ONLINE ADAPTIVE MPC:
 2: Initialize $\boldsymbol{q}_{i,0} \leftarrow \Lambda_{i,0}\bar{\boldsymbol{k}}_{i,0}$ for all $i = 1, \dots, n_x$
 3: Get initial state $x_0$
 4: for $t = 0$ to $T$ do
 5:     if $t$ mod $h = 0$ then
 6:         Compute rows of $\bar{K}_t$ (13)
 7:         Compute policy $\pi_t, \dots, \pi_{t+H}$ according to $\bar{K}_t$ by solving (14)
 8:     end if
 9:     Apply control $\boldsymbol{u}_t = \boldsymbol{\pi}_t(\boldsymbol{x}_t)$
10:     Observe next state $\boldsymbol{x}_{t+1}$
11:     Compute $\Lambda^{-1}_{i,t+1}, \boldsymbol{q}_{i,t+1}$ via recursive updates (13)
12: end for
```

The MPC approach taken herein has several similarities to modern implementations of the iterative Linear Quadratic Regulator (iLQR) framework [30], with several substantial advantages. First, iLQR requires a sequential forward pass through the dynamics, which is often prohibitively slow to perform using neural network dynamics. In our approach, the dynamics evaluations (specifically, computation of the Jacobians) can be performed in batch, substantially improving computational performance (and in fact enabling real time operation). Compared to iLQR, our control scheme uses off-the-shelf solvers which are better able to handle control and state constraints. In our experiments, enforcing state constraints enabled the agent to operate only in regions of the state space in which training data had been observed. The dynamics network may have a very poor model of the dynamics far from training data, and thus without the state constraints control performance can degrade substantially. While iLQR results in a nominal action sequence as well as a feedback policy, we simply perform one iteration of time-varying LQR after the termination of our MPC optimization.

VII. Experimental Results

A. Experimental Settings

We evaluate CAMeLiD on two benchmark systems for nonlinear systems, the 4 degree of freedom Cart-Pole system, and a 12 degree of freedom Quadrotor model. We now detail these two systems and the baselines that we use as comparisons for our approach. More in-depth details of the cost function, dynamics, and randomization can be found in

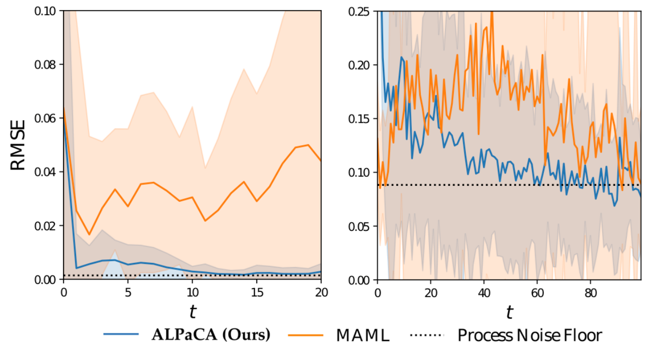


Fig. 2: Mean and 95% confidence of the RMSE of the prediction of the next state at each timestep of an episode. For both the cart-pole (**left**) and the quadrotor (**right**) tasks our approach substantially outperforms the MAML baseline by providing more accurate predictions. Moreover, the ALPaCA model adapts extremely rapidly and the decrease in RMSE is nearly monotonic.

the appendix. Due to space constraints, we report only our central performance results. More experimental results are available in [29].

Cart-Pole: The Cart-Pole system is a classic benchmark nonlinear system for control in which a pole is attached as a pendulum onto a cart which can move horizontally. This system has one control input, the force applied to the cart. The key parameters of the nonlinear dynamics are the mass of the cart, the mass of the pole, and the length of the pole. In our experiments we consider the setting where each is varied simultaneously. The system starts with the pole upright, and the cart at position $x = -3$. The task is to move the cart to a position of $x = 0$ while keeping the pole upright, and this is encoded through a simple quadratic cost centered at the goal state.

Quadrotor: To investigate the performance of the framework on a more complex system we consider a quadrotor model, a standard benchmark problem in control and reinforcement learning [31], [32], [33]. We consider a quadrotor delivery scenario, in which a quadrotor has to carry an unknown mass attached at an unknown position on its frame. The model also has process noise on the velocities, simulating the effects of wind. We test on a simple translation task, by choosing a quadratic cost function centered around a goal position.

Baselines: We evaluate CAMeLiD, which combines the ALPaCA adaptive dynamics model with our SCP-based MPC controller, against a variety of alternative models. First, to investigate the efficacy of ALPaCA (with our modifications), we compare against the MAML dynamics model used in [34], as well as a baseline, non-adaptive neural network model trained to minimize NLL on all the transition data in $\mathcal{D}$. All the neural network models use the same architecture, and are trained on the same data. For each dynamics model, we also compare the MPC controller against MPPI [2], a derivative free Monte Carlo-based MPC algorithm. The combination of the MAML dynamics model and the MPPI controller is the GrBAL framework proposed in [22].

TABLE I: Mean and 95% confidence of the cost attained on different systems with different modes of randomization. CAMeLiD, a combination of ALPaCA and the MPC controller, consistently demonstrates superior system-level performance. Details of the distributions over system parameters for both domains can be found in the appendix.

Model	**Cart-Pole**		**Quadrotor**	
CAMeLiD	**14.66**	±1.43	**41.30**	±11.23
MAML/MPC	58.71	±11.77	79.67	±12.92
Baseline/MPC	81.76	±16.73	177.09	±45.71
ALPaCA/MPPI	136.32	±2.74	84.52	±4.04
GrBAL [22]	514.92	±62.25	60.74	±1.14
Baseline/MPPI	141.55	±25.33	81.83	±2.12

We evaluate all six model-controller combinations on the two simulated control systems under different forms of randomization to investigate what forms of uncertainty this framework can address. For each system $\boldsymbol{f}(\cdot;\boldsymbol{\theta})$ with a particular distribution of parameters $p(\boldsymbol{\theta})$, we first train each of the dynamics models from data generated from this distribution. After training the model, we evaluate each model-controller combination by simulating rollouts on 25 systems sampled from the same distribution $p(\boldsymbol{\theta})$. From these simulated rollouts, we report the average realized cost in Table I.

B. *Dynamics Meta-Learning*

We first investigate the model adaptation in isolation: at every timestep of simulation, we log the RMSE between the predicted next state and the actual observed next state. Fig. 2 compares ALPaCA against MAML in terms of these prediction RMSE values for the both systems (using our MPC controller). The results indicate that the RMSE predictions made by ALPaCA quickly decrease and approach the minimum achievable for the amount of process noise present in each system—meaning that the model is able to almost perfectly predict the next state after just 10 transitions (for the cart-pole system). In contrast, the MAML model does not significantly improve over time in terms of RMSE prediction of the next state. This gap in performance may be explained by the fact that online, the MAML dynamics model is only taking a gradient step on all the parameters of the network. As such, the performance of MAML can be highly sensitive to hyperparameters such as the step-size to use for this online gradient step. In contrast, ALPaCA requires no such hyperparameter as it exactly updates the last layer weights to the analytic optimal through the recursive least-squares updates. Similar results are obtained on the quadrotor task, where we observe that ALPaCA adapts more efficiently and stably than MAML, with the prediction RMSE here converging to the lower bound of the process noise RMSE after observing roughly 50 transition samples. Due to the large amount of process noise present, the observed RMSE has large variance (due in part to the relatively small number of trials).

C. *Adaptive Control Performance*

Accurate predictions do not always translate into precise control. Hence, we now proceed with evaluating the control performance of CAMeLiD on the cart-pole and quadrotor tasks. The results reported in Table I show that CAMeLiD outperforms the other models in terms of realized cost for all forms of randomization. Notably, we observe that the MPC controller consistently outperforms MPPI in this setting. We suspect that this is because MPPI approximates the optimal sequence of actions via Monte-Carlo sampling. As such, the quality of the optimized trajectories depends on hyperparameters such as the number of action samples taken and the form of the action distribution, which may be hard to tune. Furthermore, as this task involves control around an unstable equilibrium of the system, sampling low-cost action sequences is low-probability.

VIII. Discussion and Extensions

While ALPaCA models are performing Bayesian regression, and thus maintaining a full posterior over the next state, we have not utilized this distribution in our control scheme. Within adaptive control, our approach is *certainty equivalent*, as opposed to an approach which incorporates uncertainty, which is called *cautious*. Optimally, one would aim to incorporate both the uncertainty of predictions and the knowledge that information will be gained in the future. This approach, referred to as *dual control* [35], is typically computationally intractable. The relative merits of *certainty-equivalent* versus *cautious* adaptive control are still an open topic of research [36]. Cautious adaptive control tends to be overly conservative as it incorporates model-uncertainty without capturing information gain, especially over long horizons.

During this work we implemented two cautious adaptive control schemes: an iLQR-based approach that incorporates uncertainty, similar to [37], as well as probabilistic differential dynamic programming (PDDP) [38]. The former relies on re-sampling the model at each time, whereas the latter adds the uncertainty to the state vector and approximately propagates it forward in time. As such, they represent substantially different approaches to uncertainty-aware control. We repeatedly found that both methods performed worse than certainty-equivalent versions of iLQR and MPC, which we believe represents an important negative result.

Beyond cautious adaptive control schemes, there are numerous ways in which the probabilistic predictions generated by our model may be leveraged, which we leave for future work. First, while incorporating the model uncertainty into the cost resulted in degraded performance, this model uncertainty may be used for probabilistic safety guarantees, via e.g. chance-constrained programming approaches [39]. Another possible application of the probabilistic modeling is Bayesian model testing/fault detection. Because we maintain a posterior distributions over outcomes, this may be compared to our prior model, and if the probability is large that there exists a mismatch between our posterior and the incoming data, we may e.g. reset to our prior. This approach could effectively perform adaptive control for systems with discontinuous changes in dynamics. A similar approach was taken in [34]. However, because they are using MAML [7] models, their probability evaluations were implicitly assuming isotropic unit variance (as they were using the 2-norm of the prediction error as a surrogate for likelihood) that did not concentrate as more data was observed online. Finally, in the reinforcement-learning setting in which episodes are performed sequentially, uncertainty may be used for exploration via e.g. Thompson sampling [40] or optimism [41].

IX. Conclusions

A critical capability for truly autonomous systems is to quickly adapt to new environments and conditions. In this paper, we frame this adaptation from a model-based control perspective, as adapting on-the-fly a forward dynamics model which is simultaneously used for control. To do so, we propose CAMeLiD: a novel approach which leverage advances in meta-learning to efficiently and robustly adapt a neural network-based dynamics model within an adaptive control framework. Experimental results demonstrate that CAMeLiD outperform existing methods both in prediction accuracy, and in control performance.

References

[1] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Neural Information Processing Systems (NeurIPS)*, 2018.

[2] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[3] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[4] S. Zhou, M. K. Helwa, and A. P. Schoellig, "Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking," *IEEE Conference on Decision and Control (CDC)*, 2017.

[5] J. Harrison, A. Garg, B. Ivanovic, Y. Zhu, S. Savarese, L. Fei-Fei, and M. Pavone, "AdaPT: Zero-shot adaptive policy transfer for stochastic dynamical systems," *International Symposium on Robotics Research (ISRR)*, 2017.

[6] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *International Journal of Robotics Research*, 2018.

[7] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *International Conference on Machine Learning (ICML)*, 2017.

[8] J. Harrison, A. Sharma, and M. Pavone, "Meta-learning priors for efficient online Bayesian regression," *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2018.

[9] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, 2011.

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

[11] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[12] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics." *Foundations and Trends in Robotics*, 2013.

[13] R. Murray-Smith and D. Sbarbaro, "Nonlinear adaptive control using non-parametric gaussian process prior models," *IFAC World Congress*, 2002.

[14] R. Murray-Smith, D. Sbarbaro, C. E. Rasmussen, and A. Girard, "Adaptive, cautious, predictive control with gaussian process priors," *IFAC World Congress*, 2003.

[15] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, "Gaussian process model based predictive control," *American Control Conference (ACC)*, 2004.

[16] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, "Bayesian nonparametric adaptive control using gaussian processes," *IEEE transactions on neural networks and learning systems*, 2014.

[17] R. M. Sanner and J.-J. Slotine, "Gaussian networks for direct adaptive control," *IEEE Transactions on neural networks*, 1992.

[18] D. Wang and J. Huang, "Neural network-based adaptive dynamic surface control for a class of uncertain nonlinear systems in strict-feedback form," *IEEE Transactions on Neural Networks*, 2005.

[19] M. M. Polycarpou, "Stable adaptive neural control scheme for nonlinear systems," *IEEE Transactions on Automatic Control*, 1996.

[20] S. S. Ge, C. C. Hang, T. H. Lee, and T. Zhang, *Stable adaptive neural network control*. Springer, 2013.

[21] S. Fabri and V. Kadirkamanathan, "Dual adaptive control of nonlinear stochastic systems using neural networks," *Automatica*, 1998.

[22] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *International Conference on Learning Representations (ICLR)*, 2019.

[23] S. Sæmundsson, K. Hofmann, and M. P. Deisenroth, "Meta reinforcement learning with latent variable gaussian processes," *Uncertainty in Artificial Intelligence (UAI)*, 2018.

[24] A. Argyriou, T. Evgeniou, and M. Pontil, "Multi-task feature learning," *Neural Information Processing Systems (NeurIPS)*, 2007.

[25] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," *Neural Information Processing Systems (NeurIPS)*, 2018.

[26] N. Schweighofer and K. Doya, "Meta-learning in reinforcement learning," *Neural Networks*, 2003.

[27] B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever, "Some considerations on learning to explore via meta-reinforcement learning," *arXiv:1803.01118*, 2018.

[28] M. O. Duff, "Optimal learning: Computational procedures for bayes-adaptive markov decision processes," Ph.D. dissertation, University of Massachusetts at Amherst, 2002.

[29] J. Harrison, A. Sharma, R. Dyro, X. Wang, R. Calandra, and M. Pavone, "Control adaptation via meta-learning dynamics (extended version)," 2019. [Online]. Available: http://asl.stanford.edu/wp-content/papercite-data/pdf/Harrison.Sharma.ea.ICRA20.pdf

[30] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," *American Control Conference (ACC)*, 2005.

[31] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[32] B. Ivanovic, J. Harrison, A. Sharma, M. Chen, and M. Pavone, "BaRC: Backward reachability curriculum for robotic reinforcement learning," *arXiv:1806.06161*, 2018.

[33] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice," *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[34] A. Nagabandi, C. Finn, and S. Levine, "Deep online learning via meta-learning: Continual adaptation for model-based rl," *arXiv:1812.07671*, 2019.

[35] A. Feldbaum, "Dual control theory. I," *Avtomatika i Telemekhanika*, 1960.

[36] J. Harrison, "Lecture Notes on Optimal and Learning-based Control," 2019, Stanford University.

[37] G. Lee, S. S. Srinivasa, and M. T. Mason, "GP-iLQG: Data-driven robust optimal control for uncertain nonlinear dynamical systems," *arXiv:1705.05344*, 2017.

[38] Y. Pan and E. Theodorou, "Probabilistic differential dynamic programming," *Neural Information Processing Systems (NeurIPS)*, 2014.

[39] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, 2011.

[40] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen *et al.*, "A tutorial on Thompson sampling," *Foundations and Trends in Machine Learning*, 2018.

[41] T. M. Moldovan, S. Levine, M. I. Jordan, and P. Abbeel, "Optimism-driven exploration for nonlinear systems," *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[42] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[43] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *arXiv: 1711.08013*, 2017.

[44] R. Tedrake, "Underactuated robotics: Learning, planning, and control for efficient and agile machines: Course notes for mit 6.832."

[45] R. Beard, "Quadrotor dynamics and control rev 0.1," 2008.

[46] H. Huang, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering," in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 3277–3282.

Appendix

A. Output Layer Factorization

In this section we will explicitly show that the block-diagonal covariance structure results in the RLS updates presented in Section V. We will first show that the RLS updates of the block-diagonal system preserve block-diagonality. Then we will show the mean and covariance RLS updates are equivalent to those previously presented. We will rewrite Eq. 7 as

$$\boldsymbol{x}_{t+1} = \hat{\Phi}\hat{\boldsymbol{k}} + \boldsymbol{\epsilon} \tag{15}$$

where $\hat{\boldsymbol{k}} = \text{vec}(K) = [\boldsymbol{k}_1, \dots, \boldsymbol{k}_{n_x}]^T$ where $\boldsymbol{k}_i$ is the i'th row of K, and

$$\hat{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t)^T & \boldsymbol{0}^T & & \\ \boldsymbol{0}^T & \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t)^T & & \\ & & \ddots & \\ & & & \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{u}_t)^T \end{bmatrix}. \tag{16}$$

One can verify that this is equivalent to Eq. 7. We will structure our prior as $\hat{\boldsymbol{k}}_0 = \mathcal{N}(\hat{\bar{\boldsymbol{k}}}_0, (\Sigma_\epsilon \otimes I)\hat{\Lambda}_0^{-1})$, with

$$\hat{\Lambda}_0^{-1} = \begin{bmatrix} \Lambda_{1,0}^{-1} & & \\ & \ddots & \\ & & \Lambda_{n_x,0}^{-1} \end{bmatrix}, \tag{17}$$

where the off-diagonal blocks are zeros. For matrix input, the RLS covariance update is

$$\hat{\Lambda}_t^{-1} = \hat{\Lambda}_{t-1}^{-1} - \hat{\Lambda}_{t-1}^{-1}\hat{\Phi}_{t-1}^T(I + \hat{\Phi}_{t-1}\hat{\Lambda}_{t-1}\hat{\Phi}_{t-1}^T)^{-1}\hat{\Phi}_{t-1}\hat{\Lambda}_{t-1}^{-1}. \tag{18}$$

Note that

$$(I + \hat{\Phi}_{t-1}\hat{\Lambda}_{t-1}\hat{\Phi}_{t-1}^T)^{-1} = \text{diag}((1 + \boldsymbol{\phi}_{t-1}^T\Lambda_{i,t-1}^{-1}\boldsymbol{\phi}_{t-1})^{-1}) \tag{19}$$

and

$$\hat{\Lambda}_{t-1}^{-1}\hat{\Phi}_{t-1}^T = \begin{bmatrix} \Lambda_{1,t-1}^{-1}\boldsymbol{\phi}_{t-1} & & \\ & \ddots & \\ & & \Lambda_{n_x,t-1}^{-1}\boldsymbol{\phi}_{t-1} \end{bmatrix}, \tag{20}$$

with zero off-diagonals. One may see, then, that all terms in Eq. 18 are block-diagonal, and so we recover the n_x independent RLS updates presented previously.

The mean update may be written as

$$\hat{\boldsymbol{q}}_t = \hat{\boldsymbol{q}}_{t-1} + \hat{\Phi}_{t-1}^T\boldsymbol{x}_t \tag{21}$$

where $\hat{\boldsymbol{q}}$ is again stacked $\boldsymbol{q}_i$. The final RLS update is then

$$\hat{\bar{\boldsymbol{k}}}_t = \hat{\Lambda}_t^{-1}\hat{\boldsymbol{q}}_t. \tag{22}$$

Again, these updates are equivalent to those presented earlier. However, if one were to naïvely implement the matrix equations presented here, considerable computation would be wasted on the zero off-diagonal terms. The n_x separate RLS updates presented previously avoid this unnecessary computation.

Finally, while we have discussed choice of a structured $\hat{\Lambda}^{-1}$, one may note that choice of a diagonal Σ_ϵ also results in the predictive covariance being equivalent. Thus, the structure we have proposed falls out naturally as a consequence of this diagonal process noise assumption, which is very common in controls engineering.

B. Model Training

For all models, the neural network approximation of the dynamics is a 3-hidden layer network with and $\tanh$ activations. For the cartpole, the hidden layer sizes were 32, while for the quadrotor experiments, the first two hidden layers had size 128 while the last had size 64. For the ALPaCA models, this means the dimension of $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{w})$, n_ϕ is 32 and 64 for the cartpole and quadrotor respectively, and in each case, the last linear layer corresponds to the matrix K.

The models were all trained on a meta-dataset of 5000 datasets, where each dataset contained 40 transitions sampled randomly from the simulator. We monitored model performance on a separate, similarly generated validation meta-training set to inform model architecture and training duration. Moreover, we emphasize that CAMeLiD is operating in an off-policy sense—the generating distribution of the data is invariant to our control actions. An important direction of future work is characterizing performing in the on-policy setting.

C. SCP MPC Details

The SCP MPC controller solves the optimal control problem given the nonlinear dynamics model by sequential linearizations around a trajectory. At each iteration the dynamic constraints between states and actions are enforced using a first order truncated Taylor approximation. After each iteration, the trajectory is re-linearized at the previous solution. Unlike iLQR, the iteration involves no forward pass through the dynamics. This results in all model calls being batched over the control horizon and avoids the overhead of tens of (non-batched) model calls per iteration. The generated trajectory is dynamically feasible under the non-linear model only in the limit of zero state deviation from previous iteration. We terminated the solver when the maximum state deviation ℓ-2 norm drops below 10^{-3}.

We use state and control deviation penalties from the linearization point to ensure the quality of the first order dynamics approximation. The state deviation penalty was useful for damping the trajectory optimization iteration process. While the penalty remains a hyperparameter, we pick its value to be 1, scaled by average Frobenius norm of the state cost matrices. The control deviation penalty was set to 0. We found that using penalty deviation worked better than explicit trust region constraints because of the added trajectory iteration damping and a lack of the need for handling infeasible iterations. Our penalty on the deviation from the nominal trajectory is analogous to the adaptive regularization of the quadratic term of the approximate Q function in the iLQR implementation of [42].

If the state and control cost is kept quadratic, the optimization problem with linearized dynamics is a quadratic program (QP) which allows us to incorporate control and state box constraints and can be solved efficiently. In the sequential setting, the costs could also be quadratized, but we only use quadratic costs in this work. We use OSQP, a fast operator splitting solver [43]. Control constraints are useful in explicitly incorporating system control limitations and state constraints ensure that models are not being queried outside of the trained state space which is important when using control with function approximation models. In contrast, standard iLQR provides no such guarantees.

D. Cart-Pole

Dynamics and Randomization: The state of the Cart-Pole system is $\boldsymbol{x} = [x, \dot{x}, \theta, \dot{\theta}]$, the position and velocity of the cart and the angular position and velocity of the pole. We use the Cart-Pole dynamics as defined in [44]. Our simulator forward integrates the dynamics simply via forward Euler integration, to improve performance. The uncertain parameters $\boldsymbol{\theta}$ are the cart mass m_c, the pole mass m_p, and the pole length ℓ_p. The distribution of parameters $p(\boldsymbol{\theta})$ we use are:

$$m_c \sim \mathrm{Unif}(7.5, 12.5) \tag{23}$$
$$m_p \sim \mathrm{Unif}(1.5, 2.5) \tag{24}$$
$$\ell_p \sim \mathrm{Unif}(0.5, 1.5) \tag{25}$$

When only one parameter is randomized, the others are held fixed at the mean value.

Cost Function: To define the task of translating the cart to a goal position of $x = 0$ while keeping the pole upright $\theta = \pi$, we use the following quadratic cost function

$$\begin{aligned}\mathrm{Cost}(\boldsymbol{x}, \boldsymbol{u}) = 0.05x^2 + 0.001\dot{x}^2 + 0.005(\theta - \pi)^2 \\ + 0.01\dot{\theta}^2 + 0.001u^2\end{aligned}$$

Controller Hyperparameters: For both controllers, we use a replan rate of $h = 1$, a control horizon of $H = 30$, and simulate an episode of length $T = 80$. For MPPI, we use 5000 action sequence samples, with actions sampled from a normal distribution with mean 0 and standard deviation 7.

Data Collection: To collect data, we sampled a set of system parameters, and sampled a number of transitions from each. In particular, we sampled 10000 different $\boldsymbol{\theta}$ and 40 transitions per system. These transitions were sampled iid from the state space, and the actions were sampled iid from the action space. We chose to sample a large number of systems to avoid overfitting to a small number of dynamics models. This meta-overfitting is potentially problematic for data gathered from real experiments, as it is possible that the system will not learn general knowledge about the latent parameters, but simply identify the active latent parameters from the small set of training parameters. This type of overfitting is not currently well understood, and while we do not address it herein, we believe understanding this effect is a promising avenue of future work. Additionally, we train on transitions sampled iid from the state space, whereas the model is conditioned on data that are highly correlated at test time, as they are generated via a controller acting on the system. We did not notice any performance degredation as a result, but precisely characterizing the performance differences induced by on-policy versus off-policy training is also an important direction of future work.

E. Quadrotor

Dynamics and Randomization: Following [45], quadrotor dynamics are defined in terms of 12 states, 6 of which are in the quad's body frame of reference. The summary of the states is found in Table II.

The dynamics are given by the following relations

$$\begin{bmatrix}\dot{p}_n\\ \dot{p}_e\\ \dot{h}\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & -1\end{bmatrix} R_{123}(\phi,\theta,\psi)^{-1}\begin{bmatrix}u\\ v\\ w\end{bmatrix}$$

$$\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{bmatrix} = -\begin{bmatrix}p\\ q\\ r\end{bmatrix}\times\begin{bmatrix}u\\ v\\ w\end{bmatrix} + \begin{bmatrix}0\\ 0\\ -\frac{T_{\text{total}}}{m}\end{bmatrix} + $$

$$+ R_{123}(\phi,\theta,\psi)\begin{bmatrix}0\\ 0\\ g\end{bmatrix} - C_{d,v}\begin{bmatrix}u^2\\ v^2\\ w^2\end{bmatrix} \tag{26}$$

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta)\\ 0 & \cos(\phi) & -\sin(\phi)\\ 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta)\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix}$$

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = J^{-1}\left(-\begin{bmatrix}0 & -r & q\\ r & 0 & -p\\ -q & p & 0\end{bmatrix} J \begin{bmatrix}p\\ q\\ r\end{bmatrix} + \begin{bmatrix}\tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix}\right)$$

Position derivatives follow from body frame velocity mapping into the inertial frame and reversal of the z-axis to allow for positive up height. Body velocity derivatives are expressed in terms of the Coriolis component, thrust, decomposition of gravitational acceleration into the body frame and translational drag. Euler angle orientation derivatives are computed by considering instantaneous axis rotation rates. Body rates are derived from conservation of angular momentum in a rotating, body, frame of reference.

Moments around the geometric center of the quadrotor are

State	Interpretation
p_n	position north; inertial frame; x coordinate
p_e	position east; inertial frame; y coordinate
h	height above ground; inertial frame; negative z coordinate
u	x axis velocity; body frame
v	y axis velocity; body frame
w	z axis velocity; body frame
ϕ	orientation Euler angle; rotation around 1st axis
θ	orientation Euler angle; rotation around 2nd axis
ψ	orientation Euler angle; rotation around 3rd axis
p	body rotation rate around body x axis
q	body rotation rate around body y axis
r	body rotation rate around body z axis

TABLE II: Quadrotor Model States.

given by

$$\begin{bmatrix}\tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix} = \begin{bmatrix}0 & -l_r & 0 & l_l\\ l_f & 0 & -l_b & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}T_f(\delta_f,\dots)\\ T_r(\delta_r,\dots)\\ T_b(\delta_b,\dots)\\ T_l(\delta_l,\dots)\end{bmatrix} + $$

$$+\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ -k_m & k_m & -k_m & k_m\end{bmatrix}\begin{bmatrix}\delta_f\\ \delta_r\\ \delta_b\\ \delta_l\end{bmatrix} + \tag{27}$$

$$+ r_{\text{cm}} \times \left(R_{123}(\phi,\theta,\psi)\begin{bmatrix}0\\ 0\\ mg\end{bmatrix}\right) -$$

$$- C_{d,\omega}\begin{bmatrix}p^2\\ q^2\\ r^2\end{bmatrix}$$

where $\delta_{(\cdot)}$ is a thrust command and k_m is the proportionality constant between thrust signal and rotor momentum. $T_{(\cdot)}(\delta_{(\cdot)})$, thrust, has a dependence on system thrust command, but also quadrotor state and parameters. Specifically, we modeled the translational aerodynamic effects on thrust for angles of attack less than 20° (typical hover behavior occurs at angles of attack around 90°).

Following [46], angle of attack and free stream velocity is

$$\alpha = \tan^{-1}\left(\frac{-w'}{\sqrt{u'^2 + v'^2}}\right) \tag{28}$$

$$v_\infty = \sqrt{u'^2 + v'^2 + w'^2} \tag{29}$$

The effects were taken on a per motor basis and a separate free stream velocity and effective angle of attack were calculated for each one. u', v', w' are the local motor velocities given by body velocity, local body rates and motor position away from the geometric center. We calculate rotor induced air velocity from the fixed point expression by bisection

$$v_i = \frac{T_h/(2\rho A)}{\sqrt{(v_\infty\cos(\alpha))^2 + (v_i - v_\infty\sin(\alpha))^2}} \tag{30}$$

where $T_h = k_f\delta.$ is the hover nominal commanded thrust. Constant motor power analysis gives an expression for the actual thrust

$$T = \begin{cases}\frac{T_h^{3/2}}{\sqrt{2\rho A}(v_i - v_\infty\sin(\alpha)} & \text{if } \alpha < 20^\circ\\ k_f\delta. & \text{otherwise}\end{cases} \tag{31}$$

Param.	Interpretation	Value/Distribution
g	Gravity	9.81
m	Mass of Quad	1
ℓ	Length of Arm	0.2
$C_{d,v}$	Coefficient of Translational Drag	0.25
$C_{d,\omega}$	Coefficient of Rotational Drag	0.02255
J_{xx}	Moment of Inertia (before added mass)	0.05
J_{yy}	Moment of Inertia (before added mass)	0.05
J_{zz}	Moment of Inertia (before added mass)	0.05
$T_{\min}$	Minimum Thrust	0
$T_{\max}$	Maximum Thrust	$2mg$
m_a	Added mass	$\text{Unif}(0, 0.2)$
$r_{\text{add},x}$	x-position of addded mass	$\text{Unif}(0.0, 0.05)$
$r_{\text{add},y}$	y-position of addded mass	$\text{Unif}(0.0, 0.05)$
$r_{\text{add},z}$	z-position of addded mass	0.01
k_f	thrust signal to thrust scaling	$2mg/4$
k_m	thrust signal to rotor momentum scale	0.125
ρ	air density	1.225
A	propeller area	$\text{Unif}(0.008, 0.126)$

TABLE III: Quadrotor Parameters and Associated Randomizations. $\text{Unif}(a, b)$ denotes that the parameter is sampled uniformly within the range $[a, b)$.

For numerical stability, translational corrected thrust was clipped between 0 and 1.5 of the nominal commanded thrust.

System parameters or their range is presented in Table III.

Cost Function: We use a quadratic cost function centered around the desired goal position, with zero velocities and zero angle:

$$\begin{aligned}\text{Cost}(\boldsymbol{x}, \boldsymbol{u}) =& (p_n - x_g)^2 + (p_e - y_g)^2 + (h - h_g)^2 \\ &+ \phi^2 + \theta^2 + \psi^2 + u^2 + v^2 + w^2 \\ &+ p^2 + q^2 + r^2 + (\delta_1 - 0.5)^2 + (\delta_2 - 0.5)^2 \\ &+ (\delta_3 - 0.5)^2 + (\delta_4 - 0.5)^2\end{aligned}$$

For our experiments, $x_g = 2.0, y_g = -1.0, h_g = 0.5$.

Controller Hyperparameters: For these experiments, we simulate for an episode length of $T = 100$. We also use a terminal cost in the model predictive control equal to $\text{Cost}_T = 10 * \text{Cost}(\boldsymbol{x}, 0.5 * \mathbf{1})$ As in the Cart-Pole experiments, we use a replan rate of $h = 1.$ and a planning horizon of $H = 30$. For MPPI, we use 2000 action samples, sampled with mean 0 and standard deviation 0.2.

Data Collection: Data collection for the quadrotor system was similar to for the cart-pole system. We sampled uniformly throughout the state space, and sampled random actions. However, we restricted the range of sampled actions from the full action space. Most actions in the action space result in the quadrotor immediately losing control. However, the vehicle will typically only be using actions in a small envelope around a nominal hover action. Therefore, we sample actions by adding small perturbations (uniformly sampled from $[-3N, 3N]$) to a nominal action that is used for each prop, sampled uniformly from $[0, 20N]$. We found that this sampling scheme was necessary to achieve good performance. While off-policy training may be effective, it requires careful choice of the dataset generative distribution, whereas this is performed automatically in on-policy training. The relative merits of on-policy versus off-policy is highly dependent on the system, as training an on-policy system from scratch on the highly unstable quadrotor dynamics is difficult and inefficient.