Data-Efficient Reinforcement Learning in Continuous State-Action POMDPs

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present a data-efficient reinforcement learning method for continuous state-action systems under significant observation noise. Data-efficient solutions under small noise exist, such as PILCO which learns the cartpole swing-up task in 30s. PILCO evaluates policies by planning state-trajectories using a dynamics model. However, PILCO applies policies to the observed state, therefore planning in *observation*-space. We extend PILCO with filtering to instead plan in *belief*-space, consistent partially observable Markov decisions process (POMDP) planning. This enables data-efficient learning under significant observation noise, outperforming more naive methods such as *post-hoc* application of a filter to policies optimised by the original (unfiltered) PILCO algorithm. We test our method on the cartpole swing-up task, which involves nonlinear dynamics and requires nonlinear control.

1 Introduction

The Probabilistic Inference and Learning for COntrol (PILCO) [3] framework is a reinforcement learning algorithm, which uses Gaussian Processes (GPs) to learn the dynamics in continuous state spaces. The method has shown to be highly *efficient* in the sense that it can learn with only very few interactions with the real system. However, a serious limitation of PILCO is that it assumes that the observation noise level is small. There are two main reasons which make this assumption necessary. Firstly, the dynamics are learnt from the noisy observations, but learning the transition model in this way doesn't correctly account of the noise in the observations. If the noise is assumed small, then this will be a good approximation to the real transition function. Secondly, PILCO uses the noisy observation directly to calculate the action, which is problematic if the observation noise is substantial. Imagine a policy controlling an unstable system, where high gain feed-back is necessary for good performance. Observation noise is *amplified* when the noisy input is fed directly to the high gain controller, which in turn injects noise back into the state, creating cycles of increasing variance and instability.

In this paper we extend PILCO to address these two shortcomings, enabling PILCO to be used in situations with substantial observation noise. The first issue is addressed using the so-called *direct* method for training the transition model, see section 3.3. The second problem can be tackled by *filtering* the observations. One way to look at this is that PILCO does planning in observation space, rather than in belief space. In this paper we extend PILCO to allow filtering of the state, by combining the previous state distribution with the dynamics model and the observation using Bayes rule. Note, that this is easily done when the controller is being applied, but to gain the full benefit, we have to also take the filter into account when optimising the policy.

PILCO trains its policy through minimising the expected predicted loss when *simulating* the the system and controller actions. Since the dynamics are not known exactly, the simulation in PILCO had to simulate *distributions* of possible trajectories of the physical state of the system. This was achieved using an analytical approximation based on moment-matching and Gaussian state distributions. In

Submitted to 29th Conference on Neural Information Processing Systems (NIPS 2016). Do not distribute.

this paper we thus need to augment the simulation over physical states to include the state of the filter, an *information state* or *belief state*. This is a bit more complicated as the belief state itself is a probability distribution, we will now have to simulate distributions over distributions. This will allow the algorithm both to apply filtering during control but also to anticipate the effect of filtering during training, thereby learning a better policy.

We will first give a brief outline of related work in section 2 and the original PILCO algorithm in section 3, including the proposed use of the 'direct method' for training dynamics from noisy observations in section 3.3. In section 4 will derive the algorithm for POMDP training or planning in belief space. Note an assumption is that we observe noisy versions of the state variables. We do not handle more general cases where other unobserved states are also learnt nor learn any other mapping from the state space to observations other than additive Gaussian noise. In the final sections we show experimental results showing that the proposed algorithm handles observation noise better than competing algorithms, and close with conclusions and discussion.

2 Related Work

Implementing a filter is straightforward when the system dynamics are *known* and *linear*, referred to as Kalman filtering. For known nonlinear systems, the extended Kalman filter (EKF) is often adequate (e.g. [10]), as long as the dynamics are *locally linear*, meaning approximately linear within the region covered by the belief distribution. Otherwise, the EKF's first order Taylor expansion approximation breaks down. Greater nonlinearities usually warrant the unscented Kalman filter (UKF) or particle methods [5, 8]. The UKF uses a deterministic sampling technique to estimate moments. However, if moments can be computed analytically and exactly, moment-matching methods are preferred. Moment-matching using distributions from the exponential family (e.g. Gaussians) is equivalent to optimising the Kullback-Leibler divergence $\mathrm{KL}(p||q)$ between the true distribution p and an approximate distribution q. In such cases, moment-matching is less susceptible to model bias than the EKF due to its conservative predictions [2]. Unfortunately, the literature does not provide a continuous state-action method that is both data efficient and resistant to noise when the dynamics are *unknown* and *locally nonlinear*. Sometimes the dynamics are partially-known, with known functional form yet unknown parameters. Such 'grey-box' problems have the aesthetic solution of incorporating the unknown dynamics parameters into state, reducing the learning task into a POMDP planning task [4, 11, 9]. Finite state-action space tasks can be similarly solved, e.g. only two Dirichlet parameters are required to model the probability of each of the finitely-many state-action-state transitions [7]. However, such solutions are not suitable for continuous-state 'black-box' problems with no prior dynamics knowledge. The original PILCO does not assume any prior dynamics knowledge, yet assumes full state observability and fails under moderate sensor noise. One proposed solution is to filter observations during policy execution [2]. However, without also predicting system trajectories w.r.t. the filtering process, the above method merely optimises policies for unfiltered control, not for filtered control. The mismatch between unfiltered-prediction and filtered-execution restricts PILCO's ability to take full advantage of filtering. Dallaire et al. [1] optimise a policy using a more realistic filtered-prediction. However, the method neglects model uncertainty by only using the maximum a posteriori (MAP) model. Unlike the method of Deisenroth and Peters [2] which gives a full probabilistic treatment of the dynamics predictions, work by Dallaire et al. [1] is therefore highly susceptible to model error, hampering data-efficiency.

We instead predict system trajectories using closed loop filtered control precisely because we execute closed loop filtered control. The resulting policies are thus optimised for the specific case in which they are used. Doing so, our method retains the same data-efficiency properties of PILCO whilst applicable to tasks with high observation noise. To evaluate our method, we use the benchmark cartpole swing-up task with noisy sensors. We show realistic and probabilistic prediction (to consider uncertainty) helps our method outperform aforementioned methods.

3 The PILCO Algorithm

PILCO is a model-based policy-search RL algorithm. It applies to continuous-state, continuous-action, continuous-observation and discrete-time control tasks. A probabilistic dynamics model is used to predict one-step system dynamics (from one timestep to the next). This allows PILCO to probabilistically predict multi-step system trajectories over arbitrary time horizon T, by repeatedly using the predictive dynamics model's output at one timestep, as the (uncertain) input in the following

timestep. For tractability PILCO uses moment-matching to keep the latent state distribution Gaussian. The result is an analytic distribution of state-trajectories, approximated as a joint Gaussian distribution over T states. The policy is evaluated as the expected total cost of the trajectories. Next, the policy is improved using local gradient-based optimisation, searching over policy-parameter space. A distinct advantage of moment-matched prediction for policy search instead of particle methods is smoother policy gradients and less local optima [6]. Finally, the policy is executed, generating additional data to re-train the dynamics model. The whole process then repeats until policy convergence. For the remainder of this section we discuss, step by step, PILCO summarised by Algorithm 1. The user first defines a parametric policy π function (Algorithm 1, line 1) and then initialises the policy parameters ψ randomly (line 2) since we begin without any data.

3.1 System Execution Phase

With a policy now defined, PILCO is ready to *execute* the system (Algorithm 1, line 4). Let the latent state of the system at time t be $x_t \in \mathbb{R}^D$, which is noisily observed as $z_t = x_t + \epsilon_t$, where $\epsilon_t \overset{iid}{\sim} \mathcal{N}(0, \Sigma^\epsilon)$. The policy π, parameterised by ψ, takes observation z_t as input, and outputs a control action $u_t = \pi(z_t, \psi) \in \mathbb{R}^F$. Applying action u_t to the dynamical system in state x_t, results in a new system state x_{t+1}. Repeating until horizon T results in a new single state-trajectory of data.

3.2 Learning Dynamics

To learn the unknown dynamics (Algorithm 1, line 5), any probabilistic model flexible enough to capture the complexity of the dynamics can be used. Bayesian nonparametric models are particularly suited given their resistance to overfitting and underfitting respectively. Overfitting otherwise leads to model bias, and underfitting limits the complexity of the system this method can learn to control. In a nonparametric model no prior dynamics knowledge is required, not even knowledge of *how complex* the unknown dynamics might be since the model's complexity grows with the available data. We define the latent dynamics $f : \tilde{x}_t \to x_{t+1}$, where $\tilde{x}_t \doteq [x_t^\top, u_t^\top]^\top$. PILCO models the dynamics with D independent Gaussian process (GP) priors, one for each dynamics output variable: $f^a : \tilde{x}_t \to x_{t+1}^a$, where $a \in [1, D]$ is the a'th dynamics output, and $f^a \sim \mathcal{GP}(\phi_a^\top \tilde{x}, k(\tilde{x}_i, \tilde{x}_j))$. Note we implement PILCO with a linear mean function[1] $\phi_a^\top \tilde{x}$. The covariance function k is squared exponential, with length scales $\Lambda_a = \text{diag}([l_{a,1}^2, ..., l_{a,D+F}^2])$, and signal variance s_a^2: $k^a(\tilde{x}_i, \tilde{x}_j) = s_a^2 \exp\big(-\frac{1}{2}(\tilde{x}_i - \tilde{x}_j)^\top \Lambda_a^{-1}(\tilde{x}_i - \tilde{x}_j)\big)$.

3.3 Learning Dynamics from Noisy Observations

The original PILCO algorithm ignored sensor noise when training each GP by assuming each observation z_t to be the latent state x_t. However, this approximation breaks down under significant noise. More complex training schemes are required for each GP that correctly treat each training datum x_t as latent, yet noisily-observed as z_t. We resort to GP state space model methods, specifically the 'direct method' [6, section 3.5]. The direct method infers the marginal likelihood $p(z_{1:N})$ approximately using moment-matching and a single forward-pass. Doing so, it specifically exploits the time series structure that generated observations $z_{1:N}$. We use the direct method to set the GP's training data $\{x_{1:N}, u_{1:N}\}$ and observation noise variance Σ^ϵ to the inducing point parameters and noise parameters that optimise the marginal likelihood. In this paper we use the superior Direct method to train GPs, both in our extended version of PILCO presented section 4, and in our implementation of the original PILCO algorithm for fair comparison in the experiments.

3.4 System Prediction Phase

In contrast to executions, PILCO also *predicts* analytic distributions of state-trajectories (Algorithm 1, line 6) for policy evaluation. PILCO does this offline, between the online system executions. Predicted control is identical to executed control except each aforementioned quantity is instead now a random variable, distinguished with capitals: X_t, Z_t, U_t, $\tilde{X}_t$ and X_{t+1}, all approximated as jointly Gaussian. These variables interact both in execution and prediction according to Figure 1. To predict X_{t+1} now that $\tilde{X}_t$ is uncertain PILCO uses the iterated law of expectation and variance:

$$p(X_{t+1}|\tilde{X}_t) = \mathcal{N}(\mu_{t+1}^x = \mathbb{E}_{\tilde{X}}[\mathbb{E}_f[f(\tilde{X}_t)]], \quad \Sigma_{t+1}^x = \mathbb{V}_{\tilde{X}}[\mathbb{E}_f[f(\tilde{X}_t)]] + \mathbb{E}_{\tilde{X}}[\mathbb{V}_f[f(\tilde{X}_t)]]). \quad (1)$$

After a one-step prediction from X_0 to X_1, PILCO repeats the process from X_1 to X_2, and up to X_T, resulting in a multi-step prediction whose joint we refer to as a distribution over state-trajectories.

[1] The original PILCO instead uses a zero mean function, and instead predicts relative changes in state.

Algorithm 1 PILCO

1: *Define* policy's functional form: $\pi : z_t \times \psi \rightarrow u_t$.
2: *Initialise* policy parameters ψ randomly.
3: **repeat**
4: *Execute* system, record data.
5: *Learn* dynamics model $p(f)$.
6: *Predict* state trajectories from $p(X_0)$ to $p(X_T)$.
7: *Evaluate* policy: $J(\psi) = \sum_{t=0}^{T} \gamma^t \mathcal{E}_t, \qquad \mathcal{E}_t = \mathbb{E}_X[\text{cost}(X_t)|\psi]$.
8: *Improve* policy: $\psi \leftarrow \text{argmin}_\psi J(\psi)$.
9: **until** policy parameters ψ converge

3.5 Policy Evaluation and Improvement

To evaluate the policy π (or more specifically, the policy parameters ψ), PILCO computes the loss $J(\psi)$ by applying a cost function to the marginal state distribution at each timestep (see Algorithm 1, line 7). After policy evaluation, PILCO optimises the policy using the analytic gradients of J. A BFGS optimisation method searches for the set of policy parameters ψ that minimise the total cost $J(\psi)$ using gradients information $\mathrm{d}J/\mathrm{d}\psi$ (Algorithm 1, line 8). To compute $\mathrm{d}J/\mathrm{d}\psi$ we require derivatives $\mathrm{d}\mathcal{E}_t/\mathrm{d}\psi$ at each time t to chain together, detailed in [3].

4 Our Method: PILCO Extended with Bayesian Filtering

Here we describe the novel aspects of our method. Our method uses the same high-level algorithm as PILCO (Algorithm 1). However, we modify[2] two subroutines to extend PILCO from MDPs to a special-case of POMDPs (specifically where the partial observability has the form of Gaussian noise on the latent state). First, we filter observations during system execution (Algorithm 1, line 4), detailed section 4.1. Second, we predict *belief*-trajectories (instead of state-trajectories) through the filter in addition to PILCO's dynamics model (line 6), detailed section 4.2. Filtering maintains a belief posterior of the latent system state. The belief is conditioned on, not just the most recent observation, but all previous observations (Figure 2). Such additional conditioning has the benefit of providing a less-noisy and more-informed input to the policy: the filtered belief-mean instead of the raw observation z_t. Our implementation continues PILCO's distinction between *executing* the system (resulting in a single real belief-trajectory) and *predicting* the system's responses (which in our case yields an analytic distribution of multiple possible future belief-trajectories). During the execution phase, the system reads specific observations z_t. Our method additionally maintains a belief state $b \sim \mathcal{N}(m, V)$ by filtering observations. This belief state b can be treated as a random variable with a distribution parameterised by belief-mean m and belief-certainty V. Note both m and V are functions of previous observations $z_{1:t}$. Now, during the (probabilistic) system prediction phase, future observations are instead *random* variables (since they have not been observed yet), distinguished as Z. Since the belief parameters m and V are functions of the now-random observations, the belief parameters must be random also, distinguished as M and V'. Given the belief's distribution parameters are now random, the belief is *hierarchically*-random, denoted $B \sim \mathcal{N}(M, V')$. Our framework allows us to consider multiple possible future belief-states *analytically* during policy evaluation. Intuitively, this framework is the analytical analogue of POMDP policy evaluation using particle methods. In particle methods, each particle is associated with a distinct belief, due to each conditioning on independent samples of future observations. A particle distribution thus defines a distribution over beliefs. Our method is the analytical analogue of this particle distribution. And by restricting our beliefs as (parametric) Gaussian, we can tractably encode a distribution over beliefs by a distribution over belief-parameters.

4.1 Filtered-System Execution Phase

When an actual filter is applied, it starts with three pieces of information: $m_{t|t-1}$, $V_{t|t-1}$ and a noisy observation of the system z_t (the dual subscript means belief of the latent physical state x at time t given all observations up until time $t-1$ inclusive). The filtering 'update step' combines prior belief $b_{t|t-1} = p(x_t|z_{1:t-1}, u_{1:t-1}) \sim \mathcal{N}(m_{t|t-1}, V_{t|t-1})$ with observational likelihood $p(x_t) = \mathcal{N}(z_t, \Sigma^\epsilon)$

[2]We implement our method by modifying PILCO's source code: `http://mlg.eng.cam.ac.uk/pilco/`.

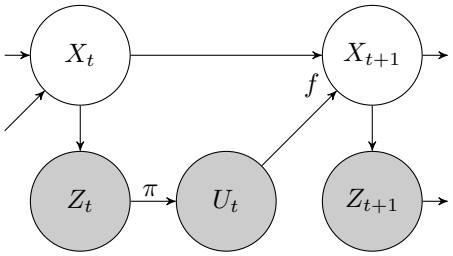

Figure 1: **The original (unfiltered) PILCO, as a probabilistic graphical model.** At each timestep, the latent system X_t is observed noisily as Z_t which is inputted directly into policy function π to decide action U_t. Finally, the latent system the will evolve to X_{t+1}, according to the unknown, nonlinear dynamics function f of the previous state X_t and action U_t.

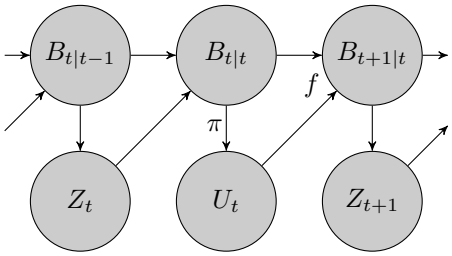

Figure 2: **Our method (PILCO extended with Bayesian filtering)**. Our prior belief $B_{t|t-1}$ (over latent system X_t), generates observation Z_t. The prior belief $B_{t|t-1}$ then combines with observation Z_t resulting in posterior belief $B_{t|t}$ (the update step). Then, the mean posterior belief $\mathbb{E}[B_{t|t}]$ is inputted into policy function π to decide action U_t. Finally, the next timestep's prior belief $B_{t+1|t}$ is predicted using dynamics model f (the prediction step).

using Bayes rule to yield posterior belief $b_{t|t} = p(x_t|z_{1:t}, u_{1:t-1})$:

$$b_{t|t} \sim \mathcal{N}(m_{t|t}, V_{t|t}), \qquad m_{t|t} = W_m m_{t|t-1} + W_z z_t, \qquad V_{t|t} = W_m V_{t|t-1}, \tag{2}$$

with weight matrices $W_m = \Sigma^\epsilon (V_{t|t-1} + \Sigma^\epsilon)^{-1}$ and $W_z = V_{t|t-1}(V_{t|t-1} + \Sigma^\epsilon)^{-1}$ computed from the standard result of a product of two Gaussians. The policy π instead uses updated belief-mean $m_{t|t}$ (smoother and better-informed than z_t) to decide the action: $u_t = \pi(m_{t|t}, \psi)$. Thus, the joint distribution over the updated (random) belief and the (non-random) action is

$$\tilde{b}_{t|t} \doteq \begin{bmatrix} b_{t|t} \\ u_t \end{bmatrix} \sim \mathcal{N}\left(\tilde{m}_{t|t} \doteq \begin{bmatrix} m_{t|t} \\ u_t \end{bmatrix}, \tilde{V}_{t|t} \doteq \begin{bmatrix} V_{t|t} & 0 \\ 0 & 0 \end{bmatrix}\right). \tag{3}$$

Next, the filtering 'prediction step' computes the predictive-distribution of $b_{t+1|t} = p(x_{t+1}|z_{1:t}, u_{1:t})$ from the output of dynamics model f given random input $\tilde{b}_{t|t}$. The distribution $f(\tilde{b}_{t|t})$ is non-Gaussian yet has analytically computable moments [3]. For tractability, we approximate $b_{t+1|t}$ as Gaussian-distributed using moment-matching:

$$b_{t+1|t} \sim \mathcal{N}(m_{t+1|t}, V_{t+1|t}), \qquad m^a_{t+1|t} = \mathbb{E}_{\tilde{b}_{t|t}}[f^a(\tilde{b}_{t|t})], \qquad V^{ab}_{t+1|t} = \mathbb{C}_{\tilde{b}_{t|t}}[f^a(\tilde{b}_{t|t}), f^b(\tilde{b}_{t|t})], \tag{4}$$

where a and b refer to the a'th and b'th dynamics output. Both $m^a_{t+1|t}$ and $V^{ab}_{t+1|t}$ are derived in Appendix C. The process then repeats using the predictive belief (Eq. 4) as the prior belief in the following timestep. This completes the specification of the system in execution.

4.2 Filtered-System Prediction Phase

In *system prediction*, we compute the probabilistic behaviour of the filtered system via an analytic distribution of possible beliefs. A distribution over beliefs b is in principle a distribution over its parameters m and V. To distinguish m, V and b as now being *random* and *hierarchically-random* respectively, we capitalise them: M, V' and B. As an approximation we do not consider random-variance V', but instead consider the non-random $V = \mathbb{E}[V']$ (which has a fixed value, for a given timestep). Restricting M to being Gaussian distributed then we begin system prediction with the joint:

$$\begin{bmatrix} M_{t|t-1} \\ Z_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu^m_{t|t-1} \\ \mu^x_t \end{bmatrix}, \begin{bmatrix} \Sigma^m_{t|t-1} & 0 \\ 0 & \Sigma^z_t \end{bmatrix}\right), \tag{5}$$

where $\Sigma^z_t = \Sigma^m_{t|t-1} + V_{t|t-1} + \Sigma^\epsilon$. The updated belief posterior is also Gaussian,

$$M_{t|t} \sim \mathcal{N}\left(\mu^m_{t|t} = \mu^m_{t|t-1}, \quad \Sigma^m_{t|t} = W_m \Sigma^m_{t|t-1} W_m^\top + W_z \Sigma^z_t W_z^\top\right). \tag{6}$$

The policy now has a random input $M_{t|t}$, thus the control output must also be random (even though π is a deterministic function): $U_t = \pi(M_{t|t}, \psi)$, which we implement by overloading the policy function: $(\mu^u_t, \Sigma^u_t, C^{mu}_t) = \pi(\mu^m_{t|t}, \Sigma^m_{t|t}, \psi)$, where μ^u_t is the output mean, Σ^u_t the output variance and C^{mu}_t input-output covariance with premultiplied inverse input variance, $C^{mu}_t \doteq (\Sigma^m_{t|t})^{-1}\mathbb{C}_M[M_{t|t}, U_t]$.

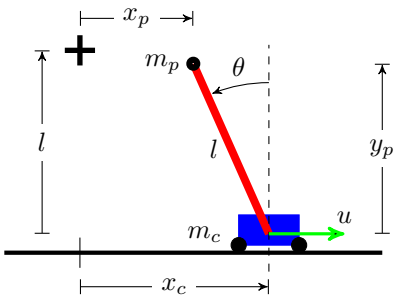

Figure 3: **The cartpole swing-up task.** A pendulum of length l is attached to a cart by a frictionless pivot. The cart has mass m_c and position x_c. The pendulum's endpoint has mass m_p and position (x_p, y_p), with angle θ from vertical. The cart begins at position $x_c = 0$ and pendulum hanging down: $\theta = \pi$. The goal is to accelerate the cart by applying horizontal force u_t at each timestep t to invert then stabilise the pendulum's endpoint at the goal (black cross), i.e. to maintain $x_c = 0$ and $\theta = 0$.

Making a moment-matched approximation yields a joint Gaussian:

$$\tilde{M}_{t|t} \doteq \begin{bmatrix} M_{t|t} \\ U_t \end{bmatrix} \sim \mathcal{N}\left(\mu^{\tilde{m}}_{t|t} \doteq \begin{bmatrix} \mu^m_{t|t} \\ \mu^u_t \end{bmatrix}, \; \Sigma^{\tilde{m}}_{t|t} \doteq \begin{bmatrix} \Sigma^m_{t|t} & \Sigma^m_{t|t} C^{mu}_t \\ (C^{mu}_t)^\top \Sigma^m_{t|t} & \Sigma^u_t \end{bmatrix}\right) \tag{7}$$

Finally, we probabilistically predict the belief-mean distribution $p(M_{t+1|t})$ and the expected belief-variance $\bar{V}_{t+1|t} = \mathbb{E}_{\tilde{M}_{t|t}}[V_{t+1|t}]$, both detailed in Appendix D. We have now discussed the one-step prediction of the filtered system, from $B_{t|t-1}$ to $B_{t+1|t}$. Using this process repeatedly, from initial belief $B_{0|0}$ we one-step predict to $B_{1|1}$, then to $B_{2|2}$ etc., up to $B_{T|T}$.

4.3 Policy Evaluation and Improvement

To evaluate a policy we again apply the loss function J (Algorithm 1, line 7) to the multi-step prediction (section 4.2). The policy is again optimised using the analytic gradients of J. Since J now is a function of beliefs, we additionally consider the gradients of $B_{t|t-1}$ w.r.t. ψ. As the belief is distributed by $B_{t|t-1} \sim \mathcal{N}(M_{t|t-1}, V_{t|t-1}) \sim \mathcal{N}(\mathcal{N}(\mu^m_{t|t-1}, \Sigma^m_{t|t-1}), V_{t|t-1})$, we use partial derivatives of $\mu^m_{t|t-1}$, $\Sigma^m_{t|t-1}$ and $V_{t|t-1}$ w.r.t. each other and w.r.t ψ, detailed in Appendix A.

5 Experiments

We test our algorithm on the cartpole swing-up problem (Figure 3), a benchmark for comparing controllers of nonlinear dynamical systems. We experiment using a physics simulator by solving the differential equations of the system. The pendulum begins each episode hanging downwards with the goal of swinging it up and stabilising it. The use a cart mass of $m_c = 0.5$kg. A zero-order hold controller applies horizontal forces to the cart within range $[-10, 10]$N. The policy is a mixture of 100 radial basis functions. Friction resists the cart's motion with damping coefficient $b = 0.1$Ns/m. Connected to the cart is a pole of length $l = 0.2$m and mass $m_p = 0.5$kg located at its endpoint, which swings due to gravity's acceleration $g = 9.82$m/s^2. An inexpensive camera observes the system. Frame rates of \$10 webcams are typically 30Hz at maximum resolution, thus the time discretisation is $\Delta t = 1/30s$. The state x comprises the cart position, pendulum angle, and their time derivatives $x = [x_c, \theta, \dot{x}_c, \dot{\theta}]^\top$. The cartpole's motion is described with the differential equation:

$$\dot{x}^\top = \left[\dot{x}_c, \dot{\theta}, \frac{-2m_p l\dot{\theta}^2 s + 3m_p g s c + 4u - 4b\dot{x}_c}{4(m_c+m_p) - 3m_p c^2}, \frac{-3m_p l\dot{\theta}^2 s c + 6(m_c+m_p) g s + 6(u - b\dot{x}_c)c}{4l(m_c+m_p) - 3m_p l c^2}\right], \tag{8}$$

using shorthand $s = \sin\theta$ and $c = \cos\theta$. We both randomly-initialise the system and set the initial belief of the system according to $B_{0|0} \sim \mathcal{N}(M_{0|0}, V_{0|0})$ where $M_{0|0} \sim \delta([0, \pi, 0, 0]^\top)$ and $V^{1/2}_{0|0} = \text{diag}([0.2\text{m}, 0.2\text{rad}, 0.2\text{m/s}, 0.2\text{rad/s}])$. The camera's noise standard deviation is: $(\Sigma^\epsilon)^{1/2} = \text{diag}([0.03\text{m}, 0.03\text{rad}, \frac{0.03}{\Delta t}\text{m/s}, \frac{0.03}{\Delta t}\text{rad/s}])$, noting $0.03\text{rad} \approx 1.7°$. We use the $\frac{0.03}{\Delta t}$ terms since using a camera we cannot observe velocities directly but can estimate them with finite differences.Each episode has a two second time horizon (60 timesteps). The cost function we impose is $1 - \exp\left(-\frac{1}{2}d^2/\sigma_c^2\right)$ where $\sigma_c = 0.25m$ and d^2 is the squared Euclidean distance between the pendulum's end point (x_p, y_p) and its goal $(0, l)$. I.e. $d^2 = x_p^2 + (l - y_p)^2 = (x_c - l\sin\theta)^2 + (l - l\cos\theta)^2$.

We compare four algorithms: 1) PILCO by Deisenroth and Rasmussen [3] as a baseline (unfiltered execution, and unfiltered full-prediction); 2) the method by Dallaire et al. [1] (filtered execution, and filtered MAP-prediction); 3) the method by Deisenroth and Peters [2] (filtered execution, and unfiltered full-prediction); and lastly 4) our method (filtered execution, and filtered full-prediction). For clear comparison we opted for a tightly controlled experiment. We control for data and dynamics models, i.e. each algorithm has access to the exact same data and exact same dynamics model. The reason is to eliminate variance in performance caused by different algorithms choosing different actions. We generate a single dataset by running the baseline PILCO algorithm for 11 episodes (totalling 22 seconds of system interaction). The independent variables of our experiment are 1) the method of system prediction and 2) the method of system execution. We then optimise each policy from the same initialisation using their respective prediction methods. Finally, we measure and compare their performances in both prediction and execution.

6 Results and Analysis

6.1 Predictive Performance

We now compare algorithm performance, both predictive (Figure 4) and empirical (Figure 5). First, we analyse predictive costs per timestep (Figure 4). Since predictions are probabilistic, the costs have distributions, with the exception of Dallaire et al. [1] which predicts MAP trajectories and therefore has deterministic cost. Even though we plot distributed costs, policies are optimised w.r.t. expected total cost only. Using the same dynamics, the different prediction methods optimise different policies (with the exception of Deisenroth and Rasmussen [3] and Deisenroth and Peters [2], whose prediction methods are identical). During the first 10 timesteps, we note identical performance with maximum cost due to the non-zero time required physically swing the pendulum up near the goal. Performances thereafter diverge. Since we predict w.r.t. a filtering process, less noise is predicted to be injected into the policy, and the optimiser can thus afford higher gain parameters w.r.t. the pole at balance point. If we linearise our policy around the goal point, our policy has a gain of -81.7N/rad w.r.t. pendulum angle, a larger-magnitude than both Deisenroth method gains of -39.1N/rad (negative values refer to *left* forces in Figure 3). Being afforded higher gains our policy is more reactive and more likely to catch a falling pendulum. Finally, we note Dallaire et al. [1] predict very high performance. Without balancing the costs across multiple possible trajectories, the method instead optimises a sequence of deterministic states to near perfection.

6.2 Empirical Performance

To compare the predictive results against the empirical, we used 100 executions of each algorithm (Figure 5). First, we notice a stark difference between predictive and executed performances from Dallaire et al. [1], due to neglecting model uncertainty, suffering model bias. In contrast, the other methods consider uncertainty and have relatively unbiased predictions, judging by the similarity between predictive-vs-empirical performances. Deisenroth's methods, which differ only in execution, illustrate that filtering during execution-only can be better than no filtering at all. However, the major benefit comes when the policy is evaluated from multi-step predictions of a filtered system. Opposed to Deisenroth and Peters [2], our method's predictions reflect reality closer because we both predict and execute system trajectories using closed loop filtering control.

To test statistical significance of empirical cost differences given 100 executions, we use a Wilcoxon rank-sum test at each time step. Excluding time steps ranging $t = [0, 29]$ (whose costs are similar), the minimum z-score over timesteps $t = [30, 60]$ that our method has superior average-cost than each other methods follows: Deisenroth 2011 $\min(z) = 4.99$, Dallaire 2009's $\min(z) = 8.08$, Deisenroth 2012's $\min(z) = 3.51$. Since the minimum $\min(z) = 3.51$, we have $p > 99.9\%$ certainty our method's average empirical cost is superior than each other method.

6.3 Training Time Complexity

Training the GP dynamics model involved $N = 660$ data, $M = 50$ inducing points under a sparse GP FITC $P = 100$ policy RBF centroids, $D = 4$ state dimensions, $F = 1$ action dimensions, and $T = 60$ timestep horizon. To train the dynamics model scales $\mathcal{O}(DNM^2)$. Policy optimisation (with 300 steps, each of which require trajectory prediction with gradients) is the most intense part: our method and both Deisenroth's methods scale $\mathcal{O}(M^2D^2(D+F)^2T + P^2D^2F^2T)$, whilst Dallaire's only scales $\mathcal{O}(MD(D+F)T + PDFT)$. Worst case we require $M = \mathcal{O}(\exp(D+F))$ inducing

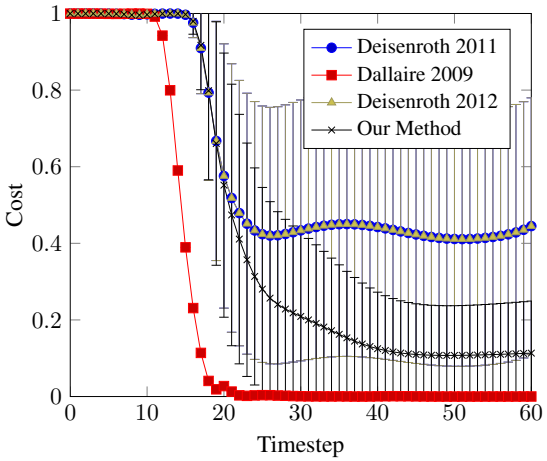

Figure 4: **Predictive costs per timestep.** The error bars show ± 1 standard deviation. Each algorithm has access to the same data set (generated by baseline Deisenroth 2011) and dynamics model. Algorithms differ in their multi-step prediction methods (except Deisenroth's algorithms whose predictions overlap).

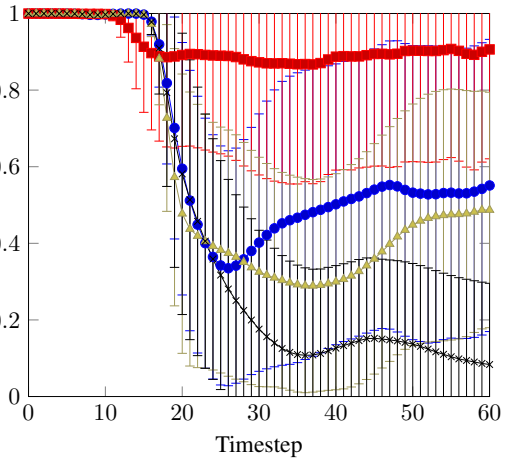

Figure 5: **Empirical costs per timestep**. We generate empirical cost distributions from 100 executions per algorithm. Error bars show ± 1 standard deviation. The plot colours and shapes correspond to the legend in Figure 4.

points to capture dynamics, the average case is unknown. Total training time was four hours to train the original PILCO method with an additional one hour to re-optimise the policy.

7 Conclusion and Future Work

In this paper, we extended the original PILCO algorithm [3] to filter observations, both during system execution and multi-step probabilistic prediction required for policy evaluation. The extended framework enables learning in *partially-observed* environments (POMDPs) whilst retaining PILCO's data-efficiency property. We demonstrated successful application to a benchmark control problem, the noisily-observed cartpole swing-up. Our algorithm learned a good policy under significant observation noise in less than 30 seconds of system interaction. Importantly, our algorithm evaluates policies with predictions that are faithful to reality. We predict w.r.t. closed loop filtered control precisely because we execute closed loop filtered control, unlike other RL algorithms.

We showed experimentally that *faithful* and *probabilistic* predictions give greater performance gains than otherwise. For clear comparison we constrained each algorithm to use the same dynamics dataset rather than each interacting with the system to generate their own. Thus, we cannot currently claim superior data-collection abilities of our method, only superior data-usage. In future work we wish to relax this experimental constraint, to test a difference in data-collection abilities. However, the extra variance in empirical performance (caused by selection of different data) means a much larger number of experiments will be required to detect if such *additional* performance gains exist.

Several more challenges remain for future work. Firstly the assumption of zero variance of the belief-variance could be relaxed. A relaxation allows distributed trajectories to more accurately consider belief states having various degrees of certainty (belief-variance). E.g. system trajectories have larger belief-variance when passing though data-sparse regions of state-space, and smaller belief-variance in data-dense regions. Secondly, the policy could be a function of the full belief distribution (mean and variance) rather than just the mean. Such flexibility could help the policy make more 'cautious' actions when more uncertain about the state. Thirdly, the framework could be extended to active learning. Currently, the framework is a passive learner, greedily optimising the total cost-means and ignoring cost-variance information which could otherwise better inform exploration, increasing data-efficiency further.

References

[1] Dallaire, P., Besse, C., Ross, S., and Chaib-Draa, B. (2009). Bayesian reinforcement learning in continuous POMDPs with Gaussian processes. In *International Conference on Intelligent Robots and Systems (IROS 2009)*, pages 2604–2609. IEEE.

[2] Deisenroth, M. and Peters, J. (2012). Solving nonlinear continuous state-action-observation POMDPs for mechanical systems with Gaussian noise. In *European Workshop on Reinforcement Learning (EWRL 2012)*.

[3] Deisenroth, M. and Rasmussen, C. (2011). PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML 2011)*, pages 465–472, New York, NY, USA.

[4] Duff, M. (2002). *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, Department of Computer Science, University of Massachusetts Amherst.

[5] Ko, J. and Fox, D. (2009). GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90.

[6] McHutchon, A. (2014). *Nonlinear modelling and control using Gaussian processes*. PhD thesis, Department of Engineering, University of Cambridge.

[7] Poupart, P., Vlassis, N., Hoey, J., and Regan, K. (2006). An analytic solution to discrete Bayesian reinforcement learning. *Proceedings of the 23rd international conference on Machine learning*, pages 697–704.

[8] Ross, S., Chaib-Draa, B., and Pineau, J. (2008a). Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *International Conference on Robotics and Automation (ICRA 2008)*, pages 2845–2851. IEEE.

[9] Ross, S., Chaib-Draa, B., and Pineau, J. (2008b). Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2845–2851. IEEE.

[10] Van Den Berg, J., Patil, S., and Alterovitz, R. (2012). Efficient approximate value iteration for continuous gaussian pomdps. In *AAAI*.

[11] Webb, D. J., Crandall, K. L., and van den Berg, J. (2014). Online parameter estimation via real-time replanning of continuous gaussian pomdps. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 5998–6005. IEEE.

A Gradients for Policy Improvement

Let vec$(\cdot)$ be the 'unwrap operator' that reshapes matrices columnwise into vectors. We define a Markov filtered-system from the belief's parameters: $S_t = [M_{t|t-1}^\top,\ \mathrm{vec}(V_{t|t-1})^\top]^\top$. To predict system evolution, the state distribution is defined:

$$p(S_t) \sim \mathcal{N}\left(\mu_t^s = \begin{bmatrix} \mu_{t|t-1}^m \\ \mathrm{vec}(V_{t+1|t}) \end{bmatrix},\ \Sigma_t^s = \begin{bmatrix} \Sigma_{t|t-1}^m & 0 \\ 0 & 0 \end{bmatrix}\right). \tag{9}$$

To compute policy gradient $\mathrm{d}J/\mathrm{d}\psi$ we first require $\mathrm{d}\mathcal{E}_t/\mathrm{d}\psi$:

$$\begin{aligned} \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\theta} &= \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}p(S_t)}\frac{\mathrm{d}p(S_t)}{\mathrm{d}\theta} \\ &= \frac{\mathrm{d}\mathcal{E}_t}{\partial\mu_t^s}\frac{\partial\mu_t^s}{\mathrm{d}\theta} + \frac{\mathrm{d}\mathcal{E}_t}{\partial\Sigma_t^s}\frac{\partial\Sigma_t^s}{\mathrm{d}\theta}, \qquad \text{and} \end{aligned} \tag{10}$$

$$\frac{\mathrm{d}p(S_{t+1})}{\mathrm{d}\theta} = \frac{\mathrm{d}p(S_{t+1})}{\mathrm{d}p(S_t)}\frac{\mathrm{d}p(S_t)}{\mathrm{d}\theta} + \frac{\partial p(S_{t+1})}{\partial\theta}. \tag{11}$$

Application of the chain rule backwards from the state distribution at the horizon S_T, to S_t at arbitrary time t, is analogous to that detailed in PILCO [3], where we use S_t, μ_t^s and Σ_t^s in the place of x_t, μ_t and Σ_t respectively.

B Identities for Gaussian Process Prediction with Hierarchical Uncertain Inputs

The two functions

$$\begin{aligned} q(x, x', \Lambda, V) &\triangleq |\Lambda^{-1}V + I|^{-1/2} \exp\big(-\tfrac{1}{2}(x-x')[\Lambda+V]^{-1}(x-x')\big), \\ Q(x, x', \Lambda_a, \Lambda_b, V, \mu, \Sigma) &\triangleq c_1 \exp\big(-\tfrac{1}{2}(x-x')^\top[\Lambda_a+\Lambda_b+2V]^{-1}(x-x')\big) \\ &\quad \times \exp\big(-\tfrac{1}{2}(z-\mu)^\top\big[\big((\Lambda_a+V)^{-1}+(\Lambda_b+V)^{-1}\big)^{-1}+\Sigma\big]^{-1}(z-\mu)\big), \\ &= c_2\, q(x, \mu, \Lambda_a, V)\, q(\mu, x'\Lambda_b, V) \\ &\quad \times \exp\big(\tfrac{1}{2}\mathbf{r}^\top\big[(\Lambda_a+V)^{-1}+(\Lambda_b+V)^{-1}+\Sigma^{-1}\big]^{-1}\mathbf{r}\big), \end{aligned}$$

$$\text{where} \begin{cases} z &= (\Lambda_b+V)(\Lambda_a+\Lambda_b+2V)^{-1}x + (\Lambda_a+V)(\Lambda_a+\Lambda_b+2V)^{-1}x' \\ \mathbf{r} &= (\Lambda_a+V)^{-1}(x-\mu) + (\Lambda_b+V)^{-1}(x'-\mu) \\ c_1 &= \big|(\Lambda_a+V)(\Lambda_b+V) + (\Lambda_a+\Lambda_b+2V)\Sigma\big|^{-1/2}\big|\Lambda_a\Lambda_b\big|^{1/2} \\ c_2 &= \big|\big((\Lambda_a+V)^{-1}+(\Lambda_b+V)^{-1}\big)\Sigma + I\big|^{-1/2}, \end{cases} \tag{12}$$

have the following Gaussian integrals

$$\begin{aligned} \int q(x, t, \Lambda, V)\mathcal{N}(t|\mu, \Sigma)dt &= q(x, \mu, \Lambda, \Sigma+V), \\ \int q(x, t, \Lambda_a, V)\, q(t, x', \Lambda_b, V)\, \mathcal{N}(t|\mu, \Sigma)dt &= Q(x, x', \Lambda_a, \Lambda_b, V, \mu, \Sigma), \\ \int Q(x, x', \Lambda_a, \Lambda_b, 0, \mu, V)\mathcal{N}(\mu|\mathbf{m}, \Sigma)d\mu &= Q(x, x', \Lambda_a, \Lambda_b, 0, \mathbf{m}, \Sigma+V). \end{aligned} \tag{13}$$

We want to model data with E output coordinates, and use separate combinations of linear models and GPs to make predictions, $a = 1, \ldots, E$:

$$f_a(x^*) = f_a^* \sim \mathcal{N}\big(\theta_a^\top x^* + k_a(x^*, \mathbf{x})\beta_a,\ k_a(x^*, x^*) - k_a(x^*, \mathbf{x})(K_a + \Sigma_\varepsilon^a)^{-1}k_a(\mathbf{x}, x^*)\big),$$

where the E squared exponential covariance functions are

$$k_a(x, x') = s_a^2 q(x, x', \Lambda_a, 0), \ \text{ where } \ a = 1, \ldots, E, \tag{14}$$

and s_a^2 are the signal variances and Λ_a is a diagonal matrix of squared length scales for GP number a. The noise variances are Σ_ε^a. The inputs are $\mathbf{x}$ and the outputs y_a and we define $\beta_a = (K_a + \Sigma_\varepsilon^a)^{-1}(y_a - \theta_a^\top \mathbf{x})$, where K_a is the Gram matrix.

B.1 Derivatives

For symmetric Λ and V and Σ:

$$\begin{aligned}
\frac{\partial \ln q(x, x', \Lambda, V)}{\partial x} &= -(\Lambda + V)^{-1}(x - x') = -(\Lambda^{-1}V + I)^{-1}\Lambda^{-1}(x - x') \\
\frac{\partial \ln q(x, x', \Lambda, V)}{\partial x'} &= (\Lambda + V)^{-1}(x - x') \\
\frac{\partial \ln q(x, x', \Lambda, V)}{\partial V} &= -\frac{1}{2}(\Lambda + V)^{-1} + \frac{1}{2}(\Lambda + V)^{-1}(x - x')(x - x')^\top(\Lambda + V)^{-1}
\end{aligned} \tag{15}$$

Let $L = (\Lambda_a + V)^{-1} + (\Lambda_b + V)^{-1}$, $R = \Sigma L + I$, $Y = R^{-1}\Sigma = \left[L + \Sigma^{-1}\right]^{-1}$, $T : X \to XX^\top$:

$$\begin{aligned}
\partial Q(x, x', \Lambda_a, \Lambda_b, V, \mu, \Sigma) &= Q \circ \partial\Big(\ln c_2 + \ln q(x, \mu, \Lambda_a, V) + \ln q(\mu, x'\Lambda_b, V) + \frac{1}{2}y^\top Y y\Big) \\
\frac{1}{2}\frac{\partial\, y^\top Y y}{\partial \mu} &= y^\top Y \frac{\partial y}{\partial \mu} = -y^\top Y L \\
\frac{\partial \ln c_2}{\partial \Sigma} &= -\frac{1}{2}\frac{\partial \ln |L\Sigma + I|}{\partial \Sigma} = -\frac{1}{2}L^\top(L\Sigma + I)^{-\top} = -\frac{1}{2}LR^{-1} \\
\frac{\partial\, y^\top Y y}{\partial \Sigma} &= \Sigma^{-\top}Y^\top y y^\top Y^\top \Sigma^{-\top} = T(R^{-\top}y) \\
\frac{\partial \ln c_2}{\partial V} &= -\frac{1}{2}\frac{\partial \ln |L\Sigma + I|}{\partial V} = -\frac{1}{2}\frac{\partial \ln |\sum_i \left[(\Lambda_i + V)^{-1}\right]\Sigma + I|}{\partial V} \\
&= \frac{1}{2}\sum_i \Big[(\Lambda_i + V)^{-\top}\Big(\sum_j \left[(\Lambda_j + V)^{-1}\right]\Sigma + I\Big)^{-\top}\Sigma^\top(\Lambda_i + V)^{-\top}\Big] \\
&= \frac{1}{2}\sum_i \Big[(\Lambda_i + V)^{-1}Y(\Lambda_i + V)^{-1}\Big] \\
\frac{\partial\, y^\top Y y}{\partial V} = y^\top \frac{\partial Y}{\partial V}y + \frac{\partial y^\top}{\partial V}Yy + y^\top Y\frac{\partial y}{\partial V} &= \sum_i \Big[(\Lambda_i + V)^{-1}Y^\top y y^\top Y^\top (\Lambda_i + V)^{-1}\Big] \\
&\quad - \sum_i \Big[(\Lambda_i + V)^{-1}(x_{n_i} - \mu)(Yy)^\top(\Lambda_i + V)^{-1}\Big] \\
&\quad - \sum_i \Big[(\Lambda_i + V)^{-1}(y^\top Y)^\top(x_{n_i} - \mu)^\top(\Lambda_i + V)^{-1}\Big] \\
&= \sum_i \Big[T\Big((\Lambda_i + V)^{-1}(Yy - (x_{n_i} - \mu))\Big) - T\Big((\Lambda_i + V)^{-1}(x_{n_i} - \mu)\Big)\Big]
\end{aligned} \tag{16}$$

C Dynamics Predictions in System-Execution

Here we specify the predictive distribution $p(b_{t+1|t})$, whose moments are equal to the moments from dynamics model output f with uncertain input $\tilde{b}_{t|t} \sim \mathcal{N}(\tilde{m}_{t|t}, \tilde{V}_{t|t})$ similar to Deisenroth and Rasmussen [3]. Consider making predictions from $a = 1, \ldots, E$ GPs at $\tilde{b}_{t|t}$ with specification $p(\tilde{b}_{t|t}) \sim \mathcal{N}(\tilde{m}_{t|t}, \tilde{V}_{t|t})$. We have the following expressions for the predictive mean, variances and input-output covariances using the law of iterated expectations and variances:

$$
\begin{aligned}
b_{t+1|t} &\sim \mathcal{N}(m_{t+1|t}, V_{t+1|t}), && (17)\\
m^a_{t+1|t} &= \mathbb{E}_{\tilde{b}_{t|t}}[f^a(\tilde{b}_{t|t})] \\
&= \int \big(s_a^2 \beta_a^\top q(x_i, \tilde{b}_{t|t}, \Lambda_a, 0) + \phi_a^\top \tilde{b}_{t|t}\big)\mathcal{N}(\tilde{b}_{t|t}; \tilde{m}_{t|t}, \tilde{V}_{t|t}) d\tilde{b}_{t|t} \\
&= s_a^2 \beta_a^\top q^a + \phi_a^\top \tilde{m}_{t|t}, && (18)\\
C_a &\doteq \tilde{V}_{t|t}^{-1} \mathbb{C}_{\tilde{b}_{t|t}}[\tilde{b}_{t|t}, f^a(\tilde{b}_{t|t}) - \phi_a^\top \tilde{b}_{t|t}], \\
&= \tilde{V}_{t|t}^{-1} \int (\tilde{b}_{t|t} - \tilde{m}_{t|t}) s_a^2 \beta_a^\top q(\mathrm{x}, \tilde{b}_{t|t}, \Lambda_a, 0)\mathcal{N}(\tilde{b}_{t|t}; \tilde{m}_{t|t}, \tilde{V}_{t|t}) d\tilde{b}_{t|t} \\
&= s_a^2 (\Lambda_a + \tilde{V}_{t|t})^{-1} (\mathrm{x} - \tilde{m}_{t|t}) \beta_a q^a, && (19)\\
V^{ab}_{t+1|t} &= \mathbb{C}_{\tilde{b}_{t|t}}\Big[f^a(\tilde{b}_{t|t}),\ f^b(\tilde{b}_{t|t})\Big] \\
&= \mathbb{C}_{\tilde{b}_{t|t}}\Big[\mathbb{E}_f[f^a(\tilde{b}_{t|t}), \mathbb{E}_f[f^b(\tilde{b}_{t|t})\Big] + \mathbb{E}_{\tilde{b}_{t|t}}\Big[\mathbb{C}_f[f^a(\tilde{b}_{t|t}),\ f^b(\tilde{b}_{t|t})]\Big] \\
&= \mathbb{C}_{\tilde{b}_{t|t}}\Big[s_a^2 \beta_a^\top q(\mathrm{x}, \tilde{b}_{t|t}, \Lambda_a, 0) + \phi_a^\top \tilde{b}_{t|t},\ \ s_b^2 \beta_b^\top q(\mathrm{x}, \tilde{b}_{t|t}, \Lambda_b, 0) + \phi_b^\top \tilde{b}_{t|t}\Big] + \\
&\quad \delta_{ab} \mathbb{E}[s_a^2 - k_a(\tilde{b}_{t|t}, \mathrm{x})(K_a + \Sigma_\varepsilon^a)^{-1} k_a(\mathrm{x}, \tilde{b}_{t|t})] \\
&= s_a^2 s_b^2 \big[\beta_a^\top (Q^{ab} - q^a q^{b\top}) \beta_b + \\
&\quad \delta_{ab}\big(s_a^{-2} - \mathrm{tr}((K_a + \Sigma_\varepsilon^a)^{-1} Q^{aa})\big)\big] + C_a^\top \tilde{V}_{t|t} \phi_b + \phi_a^\top \tilde{V}_{t|t} C_b + \phi_a^\top \tilde{V}_{t|t} \phi_b, && (20)
\end{aligned}
$$

where

$$
\begin{aligned}
q_i^a &= q\big(\mathrm{x}_i, \tilde{m}_{t|t}, \Lambda_a, \tilde{V}_{t|t}\big), \\
Q_{ij}^{ab} &= Q\big(\mathrm{x}_i, \mathrm{x}_j, \Lambda_a, \Lambda_b, 0, \tilde{m}_{t|t}, \tilde{V}_{t|t}\big), \\
\beta_a &= (K_a + \Sigma^{\epsilon,a})^{-1}(y_a - \phi_a^\top \mathrm{x}),
\end{aligned}
$$

and training inputs are x, outputs are y_a (determined by the 'Direct method'), K_a is a Gram matrix.

D Dynamics Predictions in System-Prediction

Here we describe the prediction formulae for the random belief state in system-prediction. We again note, during execution, our belief distribution is specified by certain parameters, $b_{t|t} \sim \mathcal{N}(m_{t|t}, V_{t|t})$. By contrast, during system prediction, our belief distribution is specified by an uncertain belief-mean and certain belief-variance: $B_{t|t} \sim \mathcal{N}(M_{t|t}, V_{t|t}) \sim \mathcal{N}(\mathcal{N}(\mu^m_{t|t}, \Sigma^m_{t|t}), \bar{V}_{t|t})$, where we assumed a delta distribution on $\bar{V}_{t|t}$ for mathematical simplicity, i.e. $\mathrm{vec}(\bar{V}_{t|t}) \sim \mathcal{N}(\mathrm{vec}(\bar{V}_{t|t}), 0)$. Therefore we conduct GP prediction given hierarchically-uncertain inputs, outlining each output moment below. I.e. consider making predictions from $a = 1, \ldots, E$ GPs at $B_{t|t}$ with *hierarchical* specification

$$
p(B_{t|t}) \sim \mathcal{N}(M_{t|t}, V_{t|t}), \ \text{ and } \ M_{t|t} \sim \mathcal{N}(\mu^m_{t|t}, \Sigma^m_{t|t}), \tag{21}
$$

or equivalently the joint

$$
p\Big(\begin{bmatrix} B_{t|t} \\ M_{t|t} \end{bmatrix}\Big) \sim \mathcal{N}\Big(\begin{bmatrix} \mu^m_{t|t} \\ \mu^m_{t|t} \end{bmatrix}, \begin{bmatrix} \Sigma^m_{t|t} + \bar{V}_{t|t} & \Sigma^m_{t|t} \\ \Sigma^m_{t|t} & \Sigma^m_{t|t} \end{bmatrix}\Big). \tag{22}
$$

Mean of the Belief-Mean: dynamics prediction uses input $\tilde{M}_{t|t} \sim \mathcal{N}(\mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t})$, which is jointly distributed according to Eq. 7. Using the belief-mean $m^a_{t+1|t}$ definition (Eq. 18),

$$\begin{aligned}
\mu^{m,a}_{t+1|t} &= \mathbb{E}_{\tilde{M}_{t|t}}[M^a_{t+1|t}] \\
&= \int M^a_{t+1|t}\mathcal{N}(\tilde{M}_{t|t}; \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t})\mathrm{d}\tilde{M}_{t|t}, \\
&= s^2_a\beta^\top_a \int q(\mathrm{x}, \tilde{M}_{t|t}, \Lambda_a, V)\mathcal{N}(\tilde{M}_{t|t}; \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t})d\tilde{M}_{t|t} + \phi^\top_a \mu^{\tilde{m}}_{t|t} \\
&= s^2_a\beta^\top_a \hat{q}^a + \phi^\top_a \mu^{\tilde{m}}_{t|t}, \qquad (23) \\
\hat{q}^a_i &= q\Big(x_i, \mu^{\tilde{m}}_{t|t}, \Lambda_a, \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t}\Big). \qquad (24)
\end{aligned}$$

Input-Output Covariance: the expected input-output covariance belief term (Eq. 19) (equivalent to the input-output covariance of the belief-mean) is:

$$\begin{aligned}
\hat{C}_a &\doteq \tilde{V}^{-1}_{t|t}\mathbb{E}_{\tilde{M}_{t|t}}[\mathbb{C}_{B_{t|t}}[\tilde{B}_{t|t}, f(\tilde{B}_{t|t}) - \phi^\top_a \tilde{M}_{t|t}]], \quad \text{and similarly defined} \\
&\doteq (\Sigma^{\tilde{m}}_{t|t})^{-1}\mathbb{C}_{\tilde{M}_{t|t}}[\tilde{M}_{t|t}, \mathbb{E}_{B_{t|t}}[f(\tilde{B}_{t|t}) - \phi^\top_a \tilde{M}_{t|t}]], \\
&= (\Sigma^{\tilde{m}}_{t|t})^{-1}\int (\tilde{M}_{t|t} - \mu^{\tilde{m}}_{t|t})\mathbb{E}_{B_{t|t}}[f(\tilde{B}_{t|t})]\mathcal{N}(\tilde{M}_{t|t}; \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t})d\tilde{M}_{t|t} \\
&= (\Sigma^{\tilde{m}}_{t|t})^{-1}\int (\tilde{M}_{t|t} - \mu^{\tilde{m}}_{t|t})\big(s^2_a\beta^\top_a q(x_i, \tilde{M}_{t|t}, \Lambda_a, \tilde{V}_{t|t})\big)\mathcal{N}(\tilde{M}_{t|t}; \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t})d\tilde{M}_{t|t} \\
&= s^2_a(\Lambda_a + \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t})^{-1}(\mathrm{x} - \mu^{\tilde{m}}_{t|t})\beta_a\hat{q}^a_i. \qquad (25)
\end{aligned}$$

Variance of the Belief-Mean: the variance of randomised belief-mean (Eq 18) is:

$$\begin{aligned}
\Sigma^{m,ab}_{t+1|t} &= \mathbb{C}_{\tilde{M}_{t|t}}[M^a_{t+1|t}, M^b_{t+1|t}], \\
&= \int M^a_{t+1|t}M^b_{t+1|t}\mathcal{N}(\tilde{M}_{t|t}|\mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t})\mathrm{d}\tilde{M}_{t|t} - \mu^a_{m_{t+1|t}}\mu^b_{m_{t+1|t}}, \\
&= s^2_a s^2_b \beta^\top_a(\hat{Q}^{ab} - \hat{q}^a\hat{q}^{b\top})\beta_b + \hat{C}^\top_a\Sigma^{\tilde{m}}_{t|t}\phi_b + \phi^\top_a\Sigma^{\tilde{m}}_{t|t}\hat{C}_b + \phi^\top_a\Sigma^{\tilde{m}}_{t|t}\phi_b, \qquad (26) \\
\hat{Q}^{ab}_{ij} &= Q(\mathrm{x}_i, \mathrm{x}_j, \Lambda_a, \Lambda_b, \tilde{V}_{t|t}, \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t}). \qquad (27)
\end{aligned}$$

Mean of the Belief-Variance: using the belief-variance $V^{ab}_{t+1|t}$ definition (Eq. 20),

$$\begin{aligned}
\bar{V}^{ab}_{t+1|t} &= \mathbb{E}_{\tilde{M}_{t|t}}[V^{ab}_{t+1|t}] \\
&= \int V^{ab}_{t+1|t}\mathcal{N}(\tilde{M}_{t|t}|\mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t})\mathrm{d}\tilde{M}_{t|t} \\
&= s^2_a s^2_b\big[\beta^\top_a(\tilde{Q}^{ab} - \hat{Q}^{ab})\beta_b + \delta_{ab}\big(s_a^{-2} - \mathrm{tr}((K_a + \Sigma^a_\varepsilon)^{-1}\tilde{Q}^{aa})\big)\big] \\
&\quad + \hat{C}^\top_a\tilde{V}_{t|t}\phi_b + \phi^\top_a\tilde{V}_{t|t}\hat{C}_b + \phi^\top_a\tilde{V}_{t|t}\phi_b, \qquad (28) \\
\tilde{Q}^{ab}_{ij} &= Q(\mathrm{x}_i, \mathrm{x}_j, \Lambda_a, \Lambda_b, 0, \mu^{\tilde{m}}_{t|t}, \Sigma^{\tilde{m}}_{t|t} + \tilde{V}_{t|t}). \qquad (29)
\end{aligned}$$