Data-Efficient Reinforcement Learning in Continuous State-Action Gaussian-POMDPs

Authored by:

Carl Edward Rasmussen
Rowan McAllister

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 with 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 Paper Body

The Probabilistic Inference and Learning for COntrol (PILCO) [5] 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 for 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. Consider 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 system and controller actions. Since the dynamics are not known exactly, the simulation in PILCO had to 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

simulate distributions of possible trajectories of the physical state of the system. This was achieved using an analytical approximation based on momentmatching and Gaussian state distributions. In 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. A complication is that the belief state is itself a probability distribution, necessitating simulating distributions over distributions. This allows our algorithm to not only apply filtering during execution, but also anticipate the effects 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 POMDPs 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 of our proposed algorithm handling observation noise better than competing algorithms.

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. [13]), as long as the dynamics are approximately linear within the region covered by the belief distribution. Otherwise, the EKF?s first order Taylor expansion approximation breaks down. Larger nonlinearities warrant the unscented Kalman filter (UKF) ? a deterministic sampling technique to estimate moments ? or particle methods [7, 12]. However, if moments can be computed analytically and exactly, momentmatching methods are preferred. Moment-matching using distributions from the exponential family (e.g. Gaussians) is equivalent to optimising the Kullback-Leibler divergence KL(p——q) between the true distribution p and an approximate distribution q. In such cases, momentmatching is less susceptible to model

bias than the EKF due to its conservative predictions [4]. 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. Model-free methods can solve many tasks but require thousands of trials to solve the cartpole swing-up task [8], opposed to model-based methods like PILCO which requires about six. 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 the state, reducing the learning task to a POMDP planning task [6, 12, 14]. Finite state-action space tasks can be similarly solved, perhaps using Dirichlet parameters to model the finitely-many state-action-state transitions [10]. However, such solutions are not suitable for continuous-state ?black-box? problems with no prior dynamics knowledge. The original PILCO framework does not assume task-specific prior dynamics knowledge (only that the prior is vague, encoding only time-independent dynamics and smoothness on some unknown scale) yet assumes full state observability, failing under moderate sensor noise. One proposed solution is to filter observations during policy execution [4]. However, without also predicting system trajectories w.r.t. the filtering process, a policy is merely optimised for unfiltered control, not filtered control. The mismatch between unfiltered-prediction and filtered-execution restricts PILCO?s ability to take full advantage of filtering. Dallaire et al. [3] optimise a policy using a more realistic filtered-prediction. However, the method neglects model uncertainty by using the maximum a posteriori (MAP) model. Unlike the method of Deisenroth and Peters [4] which gives a full probabilistic treatment of the dynamics predictions, work by Dallaire et al. [3] 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 that realistic and probabilistic prediction enable our method to outperform the aforementioned methods. 2

Algorithm 1 PILCO 1: Define policy?s functional form: ? : zt ? ? ? ut . 2: Initialise policy parameters ? randomly. 3: repeat 4: Execute policy, record data. 5: Learn dynamics model p(f). 6: Predict state trajectories from 0) to p(XT). Pp(X T 7: Evaluate policy: J(?) = t=0 ? t Et , Et = EX [cost(Xt)—?]. 8: Improve policy: ? ? argmin? J(?). 9: until policy parameters ? converge

3

The PILCO algorithm

PILCO is a model-based policy-search RL algorithm, summarised by Algorithm 1. It applies to continuous-state, continuous-action, continuous-observation and discrete-time control tasks. After the policy is executed, the additional data is recorded to train a probabilistic dynamics model. The probabilistic dynamics model is then used to predict one-step system dynamics (from one timestep to

the next). This allows PILCO to probabilistically predict multi-step system trajectories over an 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, where the cost function is assumed to be known. 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 fewer local optima [9]. This process then repeats a small number of iterations before converging to a locally optimal policy. We now discuss details of each step in Algorithm 1 below, with policy evaluation and improvement discussed Appendix B. 3.1

Execution phase

Once a policy is initialised, PILCO can execute the system (Algorithm 1, line 4). Let the latent state iid of the system at time t be xt ? RD , which is noisily observed as zt = xt + t , where t ? N (0, ?). The policy ?, parameterised by ?, takes observation zt as input, and outputs a control action ut = ?(zt , ?) ? RF . Applying action ut to the dynamical system in state xt , results in a new system state xt+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 - the result of optimising the policy on the erroneous model. 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 : x ?t ? xt+1 , where x ?t = [x¿ t , ut] . PILCO models the dynamics with D independent Gaussian process (GP) priors, one for each dynamics output variable: fa : x ?t ? xat+1 , where a ? [1, D] is the a?th dynamics output, and f a ? GP(?¿ ?, k a (? xi , x ?j)). ax 1 ¿ Note we implement PILCO with a linear mean function , ?a x ?, where ?a are additional hyperparameters trained by optimising the marginal likelihood [11, Section 2.7]. The covariance function 2 2 2 k is squared exponential, with length scales ?a = diag([l a,1 , ..., la,D+F]), and signal variance sa :

1 a 2 ¿ ?1 k (? xi , x ?j) = sa exp ? 2 (? xi ? x ?j) ?a (? xi ? x ?j) . 3.3

Learning dynamics from noisy observations

The original PILCO algorithm ignored sensor noise when training each GP by assuming each observation zt to be the latent state xt . However, this approximation breaks down under significant noise. More complex training schemes are required for each GP that correctly treat each training 1

The original PILCO [5] instead uses a zero mean function, and instead predicts relative changes in state.

3

datum xt as latent, yet noisily-observed as zt . We resort to GP state space model methods, specifically the ?Direct method? [9, section 3.5]. The Direct method infers the marginal likelihood p(z1:N) approximately using moment-matching in a single forward-pass. Doing so, it specifically exploits the time series structure that generated observations z1:N . We use the Direct method to set the GP?s training data {x1:N , u1:N } and observation noise variance ? 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

Prediction phase

In contrast to the execution phase, 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 ? t and Xt+1 , all approximated as now a random variable, distinguished with capitals: Xt , Zt , Ut , X jointly Gaussian. These variables interact both in execution and prediction according to Figure 1. To ? t is uncertain PILCO uses the iterated law of expectation and variance: predict Xt+1 now that X ? t) = N (?xt+1 = E ? [Ef [f (X ? t)]], ?xt+1 = V ? [Ef [f (X ? t)]] + E ? [Vf [f (X ? t)]]). (1) p(Xt+1 —X X X X After a one-step prediction from X0 to X1 , PILCO repeats the process from X1 to X2 , and up to XT , resulting in a multi-step prediction whose joint we refer to as a distribution over state-trajectories.

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 (using PILCO?s source code http://mlg.eng. cam.ac.uk/pilco/) two subroutines to extend PILCO from MDPs to a special-case of POMDPs (specifically where the partial observability has the form of additive Gaussian noise on the unobserved state X). First, we filter observations during system execution (Algorithm 1, line 4), detailed in Section 4.1. Second, we predict belief -trajectories instead of state-trajectories (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 zt . 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 zt . Our method additionally maintains a belief state b ? 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 seen Figure 3. Note both m and V are functions of previous observations z1:t . Now, during the (probabilistic) 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

Bt—t?1
Xt+1
Xt
Bt—t
f
Bt+1—t f
? Zt
?
Ut
Zt+1

Figure 1: The original (unfiltered) PILCO, as a probabilistic graphical model. At each timestep, the latent system Xt is observed noisily as Zt which is inputted directly into policy function ? to decide action Ut . Finally, the latent system will evolve to Xt+1 , according to the unknown, nonlinear dynamics function f of the previous state Xt and action Ut .

Zt
Ut
Zt+1

Figure 2: Our method (PILCO extended with Bayesian filtering). Our prior belief Bt—t?1 (over latent system Xt), generates observation Zt . The prior belief Bt—t?1 then combines with observation Zt resulting in posterior belief Bt—t (the update step). Then, the mean posterior belief E[Bt—t] is inputted into policy function ? to decide action Ut . Finally, the next timestep?s prior belief Bt+1—t is predicted using dynamics model f (the prediction step).

4
m
V
B
?m
?m
M
V?
B

Figure 3: Belief in execution phase: a Gaussian random variable parameterised by mean m and variance V.

Figure 4: Belief in prediction phase: a Gaussian random variable with random mean M and nonrandom variance V? , where M is itself a Gaussian random variable parameterised by mean ?m and variance ?m .

functions of the now-random observations, the belief parameters must be random also, distinguished as M and V 0 . Given the belief?s distribution parameters are now random, the belief is hierarchicallyrandom, denoted B ?

N (M, V 0) seen Figure 4. Our framework allows us to consider multiple possible future belief-states analytically during policy evaluation. Intuitively, our framework is an 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 requires no sampling. By restricting our beliefs as (parametric) Gaussian, we can tractably encode a distribution over beliefs by a distribution over belief-parameters. 4.1

Execution phase with a filter

When an actual filter is applied, it starts with three pieces of information: mt—t?1 , Vt—t?1 and a noisy observation of the system zt (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 bt—t?1 = Xt —z1:t?1 , u1:t?1 ? N (mt—t?1 , Vt—t?1) with observational likelihood p(zt) = N (Xt , ?) using Bayes rule to yield posterior belief bt—t = Xt —z1:t , u1:t?1 : bt—t ? N (mt—t , Vt—t),

mt—t = Wm mt—t?1 + Wz zt , ?1

Vt—t = Wm Vt—t?1 ,

(2)

?1

with weight matrices Wm = ? (Vt—t?1 +?) and Wz = Vt—t?1 (Vt—t?1 +?) computed from the standard result Gaussian conditioning. The policy ? instead uses updated belief-mean mt—t (smoother and better-informed than zt) to decide the action: ut = ?(mt—t , ?). Thus, the joint distribution over the updated (random) belief and the (non-random) action is

bt—t mt—t Vt—t 0 . . . ?bt—t = ? ? N m ? t—t = , Vt—t = . (3) ut ut 0 0 Next, the filtering ?prediction step? computes the predictive-distribution of bt+1—t = p(xt+1 —z1:t , u1:t) from the output of dynamics model f given random input ?bt—t . The distribution f (?bt—t) is nonGaussian yet has analytically computable moments [5]. For tractability, we approximate bt+1—t as Gaussian-distributed using moment-matching: bt+1—t ? N (mt+1—t , Vt+1—t),

mat+1—t = E?bt—t [f a (?bt—t)],

ab Vt+1—t = C?bt—t [f a (?bt—t), f b (?bt—t)], (4)

ab where a and b refer to the a?th and b?th dynamics output. Both mat+1—t and Vt+1—t are derived in Appendix D. The process then repeats using the predictive belief (4) as the prior belief in the following timestep. This completes the specification of the system in execution.

4.2

Prediction phase with a filter

During the prediction phase, we compute the probabilistic behaviour of the filtered system via an analytic distribution of belief states (Figure 4). We begin with a prior belief at time t = 0 before any observations are recorded (symbolised by ??1?), setting the prior Gaussian belief to have a distribution equal 5

to the known initial Gaussian state distribution: B0—?1 ? N (M0—?1 , V?0—?1), where M0—?1 ? N (?x0 , 0) and V?0—?1 = ?x0 . Note the variance of M0—?1 is zero, corresponding to a single prior belief at the beginning of the prediction phase. We probabilistically predict the yet-unobserved observation Zt using our belief distribution Bt—t?1 and the known additive Gaussian observation noise t as per Figure 2. Since we restrict both the belief mean M and observation Z to being Gaussian random variables, we can express their joint distribution:

m m

?t—t?1 ?t—t?1 ?m Mt—t?1 t—t?1 ?N , , (5) m m z ?t—t?1 ?t—t?1 ?t Zt

? where ?zt = ?m t—t?1 + Vt—t?1 + ? .

The filtering ?update step? combines prior belief Bt—t?1 with observation Zt using the same logic as (2), the only difference being Zt is now random. Since the updated posterior belief mean Mt—t is a (deterministic) function of random Zt , then Mt—t is necessarily random (with non-zero variance unlike M0—?1). Their relationship, Mt—t = Wm Mt—t?1 + Wz Zt , results in the updated hierarchical belief posterior:

m Bt—t ? N Mt—t , V?t—t , where Mt—t ? N ?m , ? (6) t—t t—t , ?m t—t

m m = Wm ?m t—t?1 + Wz ?t—t?1 = ?t—t?1 , ¿ W m ?m t—t?1 Wm

?m t—t

=

V?t—t

= Wm V?t—t?1 .

+

¿ Wm ? m t—t?1 Wz

+

(7) ¿ W z ?m t—t?1 Wm

+

Wz ?zt Wz¿ ,

(8) (9)

The policy now has a random input Mt—t , thus the control output must also be random (even though ? is a deterministic function): Ut = ?(Mt—t , ?), which we implement by overloading the policy function: m u u mu (?ut , ?ut , Ctmu) = ?(?m t—t , ?t—t , ?), where ?t is the output mean, ?t the output variance and Ct . ?1 input-output covariance with premultiplied inverse input variance, Ctmu = (?m CM [Mt—t , Ut]. t—t) Making a moment-matched approximation yields a joint Gaussian:

m

mu ?m ?m ?t—t Mt—t . t—t t—t Ct m ? . m ? . ? Mt—t = ? N ?t—t = , ?t—t = . (10) Ut (Ctmu)¿ ?m ?ut ?ut t—t m Finally, we probabilistically predict the belief-mean Mt+1—t ? N (?m t+1—t , ?t+1—t) and the expected 0 belief-variance V?t+1—t = EM? t—t [Vt+1—t]. To do this we use a novel generalisation of Gaussian process moment matching with uncertain inputs by Candela et al. [1] generalised to hierarchically-uncertain inputs detailed in Appendix E. We have now discussed the one-step prediction of the filtered system,

from Bt—t?1 to Bt+1—t . Using this process repeatedly, from initial belief B0—?1 we one-step predict to B1—0 , then to B2—1 , up to BT —T ?1 .

5

Experiments

We test our algorithm on the cartpole swing-up problem (shown in Appendix A), a benchmark for comparing controllers of nonlinear dynamical systems. We experiment using a physics simulator by solving the differential equations of the system. Each episode begins with the pendulum hanging downwards. The goal is then to swing the pendulum upright, thereafter continuing to balance it. The use a cart mass of mc = 0.5kg. A zero-order hold controller applies horizontal forces to the cart within range [?10, 10]N. The policy is a linear combination of 100 radial basis functions. Friction resists the cart?s motion with damping coefficient b = 0.1Ns/m. Connected to the cart is a pole of length l = 0.2m and mass mp = 0.5kg located at its endpoint, which swings due to gravity?s acceleration g = 9.82m/s2 . An inexpensive camera observes the system. Frame rates of $10 webcams are typically 30Hz at maximum resolution, thus the time discretisation is ?t = 1/30s. The state x comprises ? ¿ . We both randomlythe cart position, pendulum angle, and their time derivatives x = [xc , ?, x? c , ?] initialise the system and set the initial belief of the system according to B0—?1 ? N (M0—?1 , V0—?1) 1/2 where M0—?1 ? ?([0, ?, 0, 0]¿) and V0—?1 = diag([0.2m, 0.2rad, 0.2m/s, 0.2rad/s]). The camera?s 0.03 1/2 noise standard deviation is: (?) = diag([0.03m, 0.03rad, 0.03 ?t m/s, ?t rad/s]), noting 0.03rad ? 1.7? . We use the 0.03 terms since using a camera we cannot observe velocities directly but can ?t estimate them with finite differences. Each episode has a two second time horizon (60 timesteps). The

cost function we impose is 1 ? exp ? 12 d2 /?c2 where ?c = 0.25m and d2 is the squared Euclidean distance between the pendulum?s end point and its goal.

6

We compare four algorithms: 1) PILCO by Deisenroth and Rasmussen [5] as a baseline (unfiltered execution, and unfiltered full-prediction); 2) the method by Dallaire et al. [3] (filtered execution, and filtered MAP-prediction); 3) the method by Deisenroth and Peters [4] (filtered execution, and unfiltered full-prediction); and lastly 4) our method (filtered execution, and filtered full-prediction). For clear comparison we first control for data and dynamics models, where 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 first experiment are 1) the method of system prediction and 2) the method of system execution. Each policy is then optimised from the same initialisation using their respective prediction methods, before comparing performances. Afterwards, we experiment allowing each algorithm to collect its own data, and also experiment with various noise level.

6 6.1

Results and analysis Results using a common dataset

We now compare algorithm performance, both predictive (Figure 5) and

empirical (Figure 6). First, we analyse predictive costs per timestep (Figure 5). Since predictions are probabilistic, the costs have distributions, with the exception of Dallaire et al. [3] 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 [5] and Deisenroth and Peters [4], whose prediction methods are identical). During the first 10 timesteps, we note identical performance with maximum cost due to the non-zero time required to 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 11). This higher gain is advantageous here, corresponding to a more reactive system which is more likely to catch a falling pendulum. Finally, we note Dallaire et al. [3] predict very high performance. Without balancing the costs across multiple possible trajectories, the method instead optimises a sequence of deterministic states to near perfection. To compare the predictive results against the empirical, we used 100 executions of each algorithm (Figure 6). First, we notice a stark difference between predictive and executed performances from Dallaire et al. [3], 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 [4], 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. 6.2

Results of full reinforcement learning task

In the previous experiment we used a common dataset to compare each algorithm, to isolate and focus on how well each algorithm makes use of data, rather than also considering the different ways each algorithm collects different data. Here, we remove the constraint of a common dataset, and test the full reinforcement learning task by allowing each algorithm to collect its own data over repeated trials of the cart-pole task. Each algorithm is allowed 15 trials (episodes), repeated 10 times with different random seeds. For a particular re-

run experiment and episode number, an algorithm?s predicted loss is unchanged when repeatedly computed, yet the empirical loss differs due to random initial states, observation noise, and process noise. We therefore average the empirical results over 100 random executions of the controller at each episode and seed.
7
1
1 Deisenroth 2011 Dallaire 2009 Deisenroth 2012 Our Method
0.8
0.8 0.6
0.4
0.4
0.2
0.2
Cost
0.6
0
0
10
20
30 Timestep
40
50
0
60

Figure 5: Predictive cost 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).

0
10
20
30 Timestep
40
50
60

Figure 6: Empirical cost 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 5.

60
40
40
Loss
60
20
20 Deisenroth 2011 Dallaire 2009 Deisenroth 2012 Our Method
0

1
2
3
4
5
6
Deisenroth 2011 Dallaire 2009 Deisenroth 2012 Our Method
0
7 8 9 10 11 12 13 14 Episode
Loss

Figure 7: Predictive loss per episode. Error bars show ?1 standard error of the mean predicted loss given 10 repeats of each algorithm.

1
60
50
50
40
40
30
30 k k k k k
10 0
1
2
= = = = =
3
1 2 4 8 16
4
3
4
5
6
7 8 9 10 11 12 13 14 Episode

Figure 8: Empirical loss per episode. Error bars show ?1 standard error of the mean empirical loss given 10 repeats of each algorithm. In each repeat we computed the mean empirical loss using 100 independent executions of the controller.

60
20
2
20 10 5
6
0
7 8 9 10 11 12 13 14 Episode

Figure 9: Empirical loss of Deisenroth 2011 for various noise levels. The error bars show ?1 standard deviation of the empirical loss distribution based on 100 repeats of the same learned controller, per noise level.

1
2
3
4
5
6
7 8 9 10 11 12 13 14 Episode

Figure 10: Empirical loss of Filtered PILCO for various noise levels. The error bars show ?1 standard deviation of the empirical loss distribution based on 100 repeats of the same learned controller, per noise level.

8

The predictive loss (cumulative cost) distributions of each algorithm are shown Figure 7. Perhaps the most striking difference between the full reinforcement learning predictions and those made with a controlled dataset (Figure 5) is that Dallaire does not predict it will perform well. The quality of the data collected by Dallaire within the first 15 episodes is not sufficient to predict good performance. Our Filtered PILCO method accurately predicts its own strong performance and additionally outperforms the competing algorithm seen in Figure 8. Of interest is how each algorithm performs equally poorly during the first four episodes, with Filtered PILCO?s performance breaking away and learning the task well by the seventh trial. Such a learning rate was similar to the original PILCO experiment with the noise-free cartpole. 6.3

Results with various observation noises

Different observation noise levels were also tested, comparing PILCO (Figure 9) with Filtered PILCO (Figure 10). Both figures show a noise factors k, such that the observation noise is: ? 0.01 ? = k ? diag([0.01m, 0.01rad, 0.01 ?t m/s, ?t rad/s]). For reference, our previous experiments used a noise factor of k = 3. At low noise factor k = 1, both algorithms perform similarly-well, since observations are precise enough to control a system without a filter. As observations noise increases, the performance of unfiltered PILCO soon drops, whilst the Filtered PILCO can successfully control the system under higher noise levels (Figure 10). 6.4

Training time complexity

Training the GP dynamics model involved N = 660 data points, M = 50 inducing points under a sparse GP Fully Independent Training Conditional (FITC) [2], P = 100 policy RBF centroids, D = 4 state dimensions, F = 1 action dimensions, and T = 60 timestep horizon, with time complexity O(DN M 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 O(M 2 D2 (D + F)2 T + P 2 D2 F 2 T), whilst Dallaire?s only scales O(M D(D + F)T + P DF T). Worst case we require M = O(exp(D + F)) inducing 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 [5] to filter observations, both during system execution and multi-step probabilistic prediction required for policy evaluation. The extended framework enables learning in a special case of partially-observed MDP 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. We showed experimentally that faithful and probabilistic predictions improved performance with respect to the baselines. For clear comparison we first constrained each algorithm to use the same dynamics dataset to demonstrate superior data-usage of our algorithm. Afterwards we relaxed this constraint, and showed our algorithm was able to learn from fewer data. 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). For example, 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. A third challenge is handling non-Gaussian noise and unobserved state variables. For example, in real-life scenarios using a camera sensor for self-driving, observations are occasionally fully or partially occluded, or limited by weather conditions, where such occlusions and limitations change, opposed to assuming a fixed Gaussian addition noise. Lastly, experiments with a real robot would be important to show the usefulness in practice.

9

2 References

[1] Joaquin Candela, Agathe Girard, Jan Larsen, and Carl Rasmussen. Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting. In International Conference on Acoustics, Speech, and Signal Processing, volume 2, pages 701?704, 2003. [2] Lehel Csat? and Manfred Opper. Sparse on-line Gaussian processes. Neural Computation, 14(3):641?668, 2002. [3] Patrick Dallaire, Camille Besse, Stephane Ross, and Brahim Chaibdraa. Bayesian reinforcement learning in continuous POMDPs with Gaussian processes. In International Conference on Intelligent Robots and Systems, pages 2604?2609, 2009. [4] Marc Deisenroth and Jan Peters. Solving nonlinear continuous state-action-observation POMDPs for mechanical systems with Gaussian noise. In European Workshop on Reinforcement Learning, 2012. [5] Marc Deisenroth and Carl Rasmussen. PILCO: A model-based and data-efficient

approach to policy search. In International Conference on Machine Learning, pages 465?472, New York, NY, USA, 2011. [6] Michael Duff. Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes. PhD thesis, Department of Computer Science, University of Massachusetts Amherst, 2002. [7] Jonathan Ko and Dieter Fox. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. Autonomous Robots, 27(1):75?90, 2009. [8] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In arXiv preprint, arXiv 1509.02971, 2015. [9] Andrew McHutchon. Nonlinear modelling and control using Gaussian processes. PhD thesis, Department of Engineering, University of Cambridge, 2014. [10] Pascal Poupart, Nikos Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete Bayesian reinforcement learning. International Conference on Machine learning, pages 697?704, 2006. [11] Carl Rasmussen and Chris Williams. Gaussian Processes for Machine Learning. MIT Press, Cambridge, MA, USA, 1 2006. [12] Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In International Conference on Robotics and Automation, pages 2845?2851, 2008. [13] Jur van den Berg, Sachin Patil, and Ron Alterovitz. Efficient approximate value iteration for continuous Gaussian POMDPs. In Association for the Advancement of Artificial Intelligence, 2012. [14] Dustin Webb, Kyle Crandall, and Jur van den Berg. Online parameter estimation via real-time replanning of continuous Gaussian POMDPs. In International Conference Robotics and Automation, pages 5998?6005, 2014.

10