Reinforcement Learning with Reference Tracking Control in Continuous State Spaces

Joseph Hall, Carl Edward Rasmussen and Jan Maciejowski

Abstract— **The contribution described in this paper is an algorithm for learning nonlinear, reference tracking, control policies given no prior knowledge of the dynamical system and limited interaction with the system through the learning process. Concepts from the field of reinforcement learning, Bayesian statistics and classical control have been brought together in the formulation of this algorithm which can be viewed as a form of indirect self tuning regulator. On the task of reference tracking using a simulated inverted pendulum it was shown to yield generally improved performance on the best controller derived from the standard linear quadratic method using only 30 s of total interaction with the system. Finally, the algorithm was shown to work on the simulated double pendulum proving its ability to solve nontrivial control tasks.**

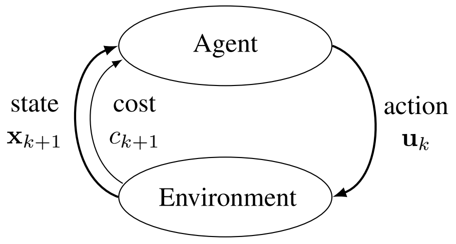

Fig. 1. The reinforcement learning framework of an agent interacting with its environment (adapted from [9]). The agent applies an action $\mathbf{u}$ at time k. It can then observe the resulting state $\mathbf{x}_{k+1}$ of the environment and the associated cost c_{k+1} (which may be a function of the next action $\mathbf{u}_{k+1}$ as well as the state $\mathbf{x}_{k+1}$).

I. INTRODUCTION

Reinforcement learning is a framework by which a control policy can be found for a system with unknown dynamics. However, there is no formal method for incorporating known information about the system into such algorithms. Such a method would be useful in the case of using an unknown system to track a known reference trajectory (or one generated according to a known stochastic process). It is this problem that forms the motivation for this paper.

The fundamental elements of the (fully observable) reinforcement learning problem are shown in Figure 1. The agent applies actions (inputs) $\mathbf{u} \in \mathcal{A}$ to an unknown, or partially unknown, environment at every timestep. This causes a change in the state $\mathbf{x} \in \mathcal{S}$ of the environment. The agent applies actions based on a policy $\pi : \mathcal{S} \to \mathcal{A}$ and the state changes according to the transition dynamics $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. At each timestep a stage cost $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is generated which defines the objectives of the agent in its environment. The goal of the agent is to find a policy π^* that minimises the expected (possibly discounted) future sum of stage costs.

There are two approaches to solving the reinforcement learning problem: *model-based*, where a model of the system dynamics is trained through data obtained through interaction and a policy is found via internal simulation; or *model-free*, where a policy is learned directly from system interaction. Model-free methods, such as $\mathcal{Q}$-learning [10] have received much attention in the literature because convergence to an optimal policy can be proven. However, such methods often require many thousands of trials before finding a good policy. Model-based methods make efficient use of interaction data by "summarising" the experience in the internal model. The problem is that of model-bias: the policy may be optimal with respect to the learned model but inadequate on the actual system. This problem was addressed by the algorithm proposed by [7] where a distribution over dynamics models is defined using a probabilistic modelling technique known as Gaussian processes. Furthermore, unlike the vast majority of reinforcement learning literature, it deals naturally with continuous state and input spaces.

The algorithm uses data to train a Gaussian process model of the dynamics under a Bayesian framework. The reinforcement learning problem is then solved by optimisation and internal simulation by propagation of the uncertainty. The inclusion of model uncertainty in the predictions has resulted in unprecedented learning speed-ups over competing algorithms. For example, the inverted pendulum swing-up and balance task was solved with only 17 s of interaction compared to 125 s achieved by [6] or $16,000$ s by [4]. Recent work has demonstrated the capability of the algorithm to solve the extremely challenging control problem of balancing a robotic unicycle in a fixed location in simulation.

However, this algorithm lacked the capability to perform learning of a reference tracking policy, given access to a preview of the future reference trajectory. The work of this paper seeks to address this by incorporating a model of the reference signal generator into the learning process. The proposed update to the algorithm was inspired by the canonical approach to optimal tracking control of linear systems given in [2]. The new algorithm can then be viewed as a form of indirect self tuning regulator (with offline parameter updates) where the identification stage is achieved using Gaussian processes and control design by direct policy

This work was supported by the EPSRC

J. Hall and J. Maciejowski are with the Control Group, Department of Engineering, Cambridge University, UK jah215@cam.ac.uk, jmm@eng.cam.ac.uk

C. Rasmussen is with the Machine Learning Group, Computational and Biological Learning Laboratory, Department of Engineering, Cambridge University, UK cer54@eng.cam.ac.uk

search using gradient descent methods.

The rest of the paper is laid out as follows. Section II outlines the theory of Gaussian processes for the modelling of dynamical systems and describes the canonical linear quadratic approach to reference tracking. An explanation of the updated learning algorithm for reference tracking in continuous state-spaces is given in Section III. Section IV contains simulation results from application of the algorithm to the inverted pendulum problem. Finally, the conclusions are summarised in Section V.

II. BACKGROUND THEORY

A. Gaussian Processes for Modelling of Dynamical Systems

Gaussian Process (GP) models can be used to learn the dynamic behaviour of arbitrary nonlinear systems described as follows

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k, \tag{1}$$

with state $\mathbf{x} \in \mathbb{R}^n$, input $\mathbf{u} \in \mathbb{R}^m$ and Gaussian noise term $\mathbf{w} \sim \mathcal{N}(0, \Sigma_{\mathbf{w}})$.

To introduce the general concept of training and prediction with a Gaussian process model, the learning of a noisy static map $y = h(\mathbf{x}) + \epsilon$, where ϵ is Gaussian distributed white noise, shall be considered. A GP can be viewed as an extension of the multivariate Gaussian distribution to infinitely long vectors, or functions. Formally, GPs are defined by [8] as *a collection of random variables, any finite number of which have a joint Gaussian distribution*. This can be expressed as

$$h(\mathbf{x}) \sim \mathcal{GP}\big(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big) \tag{2}$$

where $h(\cdot)$ is the random process, $m(\cdot)$ is the mean function, often set to zero, and $k(\cdot,\cdot)$ is the covariance function. It is important to note that the input into the random process is the state $\mathbf{x}$, not time. The parameters defining $k(\mathbf{x}, \mathbf{x}')$ are contained in the vector $\boldsymbol{\theta}$. A common choice of covariance function is the squared exponential

$$k(\mathbf{x}_i, \mathbf{x}_j) = \alpha^2 \exp\Big(-\tfrac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^\top \boldsymbol{\Lambda}^{-1}(\mathbf{x}_i - \mathbf{x}_j)\Big) \tag{3}$$

plus a noise term $\delta_{ij}\sigma_\epsilon^2$ where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1^2 \dots \lambda_n^2)$ and $\mathbf{x}_i$, $\mathbf{x}_j$ are two arbitrary points in the state space. In this case the hyperparameter vector is $\boldsymbol{\theta} = [\lambda_1 \dots \lambda_n, \alpha, \sigma_\epsilon]^\top$.

Before proceeding, the covariance matrix $K(X, X) \in \mathbb{R}^{d\times d}$, or simply K, will be defined as the matrix with elements $K_{ij} := k(\mathbf{x}_i, \mathbf{x}_j)$. Similarly, the vector $\mathbf{k}(\mathbf{x}) := \mathbf{k}(X, \mathbf{x}) \in \mathbb{R}^d$ has elements $\mathbf{k}_i = k(\mathbf{x}_i, \mathbf{x})$ and the scalar $k(\mathbf{x}) := k(\mathbf{x}, \mathbf{x})$.

1) Training: Given a set of training data $\mathcal{D} = \{X, \mathbf{y}\}$ of input vectors $X = [\mathbf{x}_1 \dots \mathbf{x}_d]^\top \in \mathbb{R}^{d\times n}$ and the corresponding observed data $\mathbf{y} = [y_1 \dots y_d]^\top$, the problem is to find the posterior distribution over the space of functions. From Bayes' rule it is known that

$$p(h|\mathbf{y}, X, \boldsymbol{\theta}) = \frac{p(\mathbf{y}|h, X, \boldsymbol{\theta})}{p(\mathbf{y}|X, \boldsymbol{\theta})} p(h|\boldsymbol{\theta}), \tag{4}$$

where $p(\mathbf{y}|X, \boldsymbol{\theta}) = \int p(\mathbf{y}|h, X, \boldsymbol{\theta}) p(h|\boldsymbol{\theta}) \mathrm{d}h$ with $h \sim \mathcal{GP}$. This can be evaluated analytically since the prior is Gaussian

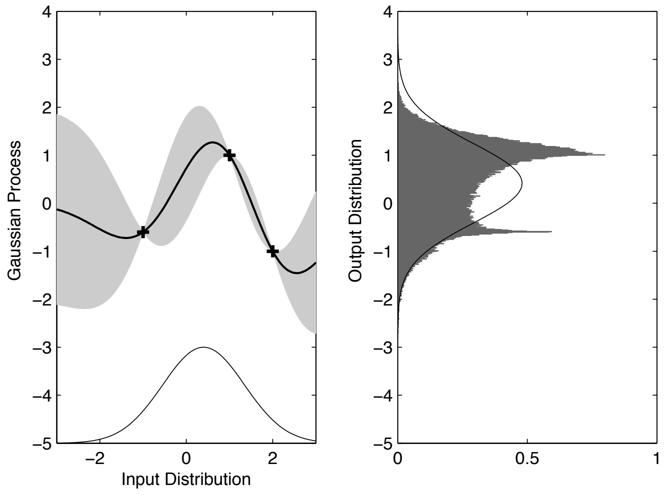

Fig. 2. Illustration of the exact moment matching scheme for propagation of uncertainty through a Gaussian process model. The left hand plot depicts the unnormalised input distribution and the GP model where the black crosses denote training data $\mathcal{D}$ and the shaded region in the left hand plot shows the 95% confidence region. The right hand plot shows the Gaussian approximation for the output distribution and an estimate of the output distribution obtained by simple Monte Carlo sampling.

$h|\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, K)$ and the likelihood is a factorized Gaussian $\mathbf{y}|h, \boldsymbol{\theta}, X \sim \mathcal{N}(h|\boldsymbol{\theta}, \sigma_\epsilon^2 I)$ because of the assumption of independent noise terms. In a fully Bayesian framework, the posterior $p(h|\mathbf{y}, X)$ would be obtained by integrating out the effect of the hyperparameters $\boldsymbol{\theta}$. This would require computationally intensive sampling-based methods. Alternatively, a point estimate $\hat{\boldsymbol{\theta}}$ can be obtained through maximisation of the log-marginal likelihood $p(\mathbf{y}|X, \boldsymbol{\theta})$ which is given by

$$\log p(\mathbf{y}|X, \boldsymbol{\theta}) = -\tfrac{1}{2}\mathbf{y}^\top (K + \sigma_\epsilon^2)^{-1}\mathbf{y} - \tfrac{1}{2}\log|K| - \tfrac{n}{2}\log 2\pi. \tag{5}$$

This is in general a non-convex problem therefore a local maximum can be found using standard gradient ascent methods, in particular the method of conjugate gradients will be used. It is this maximization of the log-marginal likelihood that constitutes "training" of the GP hyperparameters.

2) Prediction for Deterministic Inputs: Once the Gaussian process has been trained it can be used to make predictions as follows. The joint distribution of the observed target values and the function value at a single deterministic test input $\mathbf{x}^*$ under the prior can be expressed as

$$\begin{bmatrix} \mathbf{y} \\ h^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K + \sigma_\epsilon^2 I & \mathbf{k}(\mathbf{x}^*) \\ \mathbf{k}(\mathbf{x}^*)^\top & k(\mathbf{x}^*) \end{bmatrix}\right). \tag{6}$$

The conditional distribution given by $p(h^*|X, \mathbf{y}, \mathbf{x}^*) = \mathcal{N}(m(\mathbf{x}^*), v(\mathbf{x}^*))$ can then be found using the standard identity for a Gaussian conditional distribution which gives the following solution for prediction at $\mathbf{x}^*$

$$m(\mathbf{x}^*) = \mathbf{k}(\mathbf{x}^*)^\top (K + \sigma_\epsilon^2)^{-1}\mathbf{y}, \tag{7}$$

$$v(\mathbf{x}^*) = k(\mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^\top (K + \sigma_\epsilon^2)^{-1}\mathbf{k}(\mathbf{x}^*). \tag{8}$$

The mean can be viewed as a linear combination of d kernel functions, each one centred on a training point. If a multivariable mapping is now considered where $y \in \mathbb{R}^E$, then E independently trained GP models are used to describe the system. This is done rather than training a single multivariable GP because it is much less computationally intensive to train independent models. The hyperparameter vector is then extended to $\boldsymbol{\theta} = \left[\boldsymbol{\theta}^{(1)\top} \dots \boldsymbol{\theta}^{(E)\top}\right]^\top$ and the $\mathbf{y}$ vector becomes a matrix in $\mathbb{R}^{d\times E}$. So the multivariate distribution is given by $\mathcal{N}(\mathbf{m}(\mathbf{x}^*), \mathbf{v}(\mathbf{x}^*))$ where $\mathbf{v}(\mathbf{x}^*) \in \mathbb{R}^{E\times E}$ is a diagonal matrix because of the independence assumption.

3) Prediction for Uncertain Inputs: If the test input is distributed according to $\mathbf{x}^* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ then, in general, the output distribution of the GP will not be Gaussian. An illustration of this concept is shown in Figure 2. However, the mean and variance can be evaluated analytically therefore the output distribution can be approximated by a Gaussian using exact moment matching. The method for achieving this can be found in [5]. It is also possible to evaluate the term given by $\mathrm{cov}[\mathbf{x}^*]^{-1}\mathrm{cov}[\mathbf{x}^* y^\top]$ analytically, the importance of which is outlined in Section II-B.

4) Modelling a Dynamical System: The tools developed so far allow for a discrete time dynamic system of the form given in (1) to be modelled. This can be done using training data of the form

$$X = \left[\begin{bmatrix} \mathbf{x}_{k_1} \\ \mathbf{u}_{k_1} \end{bmatrix} \dots \begin{bmatrix} \mathbf{x}_{k_d} \\ \mathbf{u}_{k_d} \end{bmatrix} \right]^\top \in \mathbb{R}^{d\times(n+m)}, \tag{9}$$

$$\mathbf{y} = \left[\mathbf{x}_{k_1+1} \dots \mathbf{x}_{k_d+1}\right]^\top \in \mathbb{R}^{d\times n}. \tag{10}$$

This concludes the description of how Gaussian processes can be used to model nonlinear dynamical systems and propagate uncertainty in the predictions.

B. Conditional Independence

Within the framework of the proposed algorithm it was often necessary to calculate the cross-covariance between two conditionally independent variables. Consider two variables $\mathbf{a}$ and $\mathbf{c}$ that are conditionally independent given variable $\mathbf{b}$. This is denoted as $\mathbf{a} \perp\!\!\!\perp \mathbf{c}|\mathbf{b}$. It can be shown that the following identity is true for Gaussian distributed variables

$$\mathbf{a} \perp\!\!\!\perp \mathbf{c}|\mathbf{b} \Rightarrow \mathrm{cov}[\mathbf{a},\mathbf{c}] = \mathrm{cov}[\mathbf{a},\mathbf{b}]\mathrm{cov}[\mathbf{b}]^{-1}\mathrm{cov}[\mathbf{b},\mathbf{c}]. \tag{11}$$

Often the term $\mathrm{cov}[\mathbf{b}]^{-1}\mathrm{cov}[\mathbf{b},\mathbf{c}]$ can be evaluated analytically, therefore the inversion of the potentially singular matrix $\mathrm{cov}[\mathbf{b}]$ can be avoided. This leads to numerically stable computations.

C. Linear Quadratic Tracking Control

In this section, the LQ approach to tracking control outlined in Chapter 3 of [2] is reviewed. Consider the reference tracking control of a stochastic, discrete-time, Linear Time-Invariant (LTI) system given by

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k + \mathbf{w}_k, \tag{12}$$

with state, input and noise terms as defined in (1).

The LQ tracking problem is an extension to the LQR problem by penalising state deviations from some reference

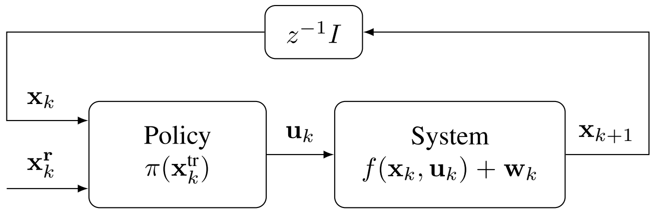
$z^{-1}I$
$\mathbf{x}_k$
$\mathbf{x}^{\mathbf{r}}_k$
Policy
$\pi(\mathbf{x}^{\mathrm{tr}}_k)$
$\mathbf{u}_k$
System
$f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k$
$\mathbf{x}_{k+1}$

Fig. 3. Block diagram displaying a system controlled according to a policy $\pi(\cdot)$ which has access to future reference trajectory information $\mathbf{x}^{\mathbf{r}}$. The term z^{-1} indicates a one-step delay.

trajectory $\mathbf{r}_k$. The policy also has access to the values of the reference trajectory for $H^{\mathbf{r}}$ timesteps in the future $\mathbf{x}^{\mathbf{r}}_k$, where $\mathbf{x}^{\mathbf{r}}_k \in \mathbb{R}^{nH^{\mathbf{r}}}$ contains the stacked future trajectory information $[\mathbf{r}_k^\top \dots \mathbf{r}_{k+H^{\mathbf{r}}-1}^\top]^\top$. The controller setup is depicted in Figure 3. The finite horizon cost function is defined as follows

$$V(H, \mathbf{x}_0) = \sum_{k=0}^{H-1} \mathbb{E}_{\mathbf{x}_k}\left[(\mathbf{x}_k - \mathbf{r}_k)^\top Q(\mathbf{x}_k - \mathbf{r}_k) + \mathbf{u}_k^\top R\mathbf{u}_k\right] + \mathbb{E}_{\mathbf{x}_H}\left[(\mathbf{x}_H - \mathbf{r}_H)^\top P_H(\mathbf{x}_H - \mathbf{r}_H)\right]. \tag{13}$$

Recognising that $\mathbf{r}_k$ is externally prescribed and that the policy will have access to the values of the reference trajectory $H^{\mathbf{r}}$ steps into the future, an artificial state-space model may be defined as

$$\mathbf{x}^{\mathbf{r}}_{k+1} = A^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}_k + B^{\mathbf{r}}\mathbf{n}_k, \quad \mathbf{r}_k = C^{\mathbf{r}}\mathbf{x}^{\mathbf{r}}_k \tag{14}$$

where $\mathbf{n}_k \sim \mathcal{N}(0, \Sigma_{\mathbf{n}})$. In the following $\Sigma_{\mathbf{n}}$ shall be set to 1. If it is assumed that the reference trajectory is generated by the following white noise driven, linear filtering system

$$\mathbf{r}_{k+1} = A^{\mathrm{f}}\mathbf{r}_k + B^{\mathrm{f}}\mathbf{n}_k \tag{15}$$

with specified initial condition $\mathbf{r}_0$, then the state-space matrices are given by the following

$$A^{\mathbf{r}} = \begin{bmatrix} \mathbf{0}_{nH^{\mathbf{r}}\times H^{\mathbf{r}}} & \begin{matrix} I_{(n-1)H^{\mathbf{r}}} \\ \begin{bmatrix}\mathbf{0}_{H^{\mathbf{r}}\times(n-2)H^{\mathbf{r}}} & A^{\mathrm{f}}\end{bmatrix} \end{matrix} \end{bmatrix},$$

$$B^{\mathbf{r}} = \begin{bmatrix} \mathbf{0}_{(n-1)H^{\mathbf{r}}\times H^{\mathbf{r}}} \\ B^{\mathrm{f}} \end{bmatrix}, \; C^{\mathbf{r}} = \begin{bmatrix} I_{H^{\mathbf{r}}} & \mathbf{0}_{H^{\mathbf{r}}\times(n-1)H^{\mathbf{r}}} \end{bmatrix}$$

where $\mathbf{0}_{p\times q}$ is the p by q matrix of zeros. This formulation can characterise a wide range of reference signals. Two examples are shown in Figure 4 of second-order filtered white noise and exponentially decaying sinusoids of a given frequency. In the sinusoid example $B^{\mathrm{f}} = 0$ but randomness arises by allowing a distribution over starting state $\mathbf{r}_0$.

The tracking problem may now be recast as a standard LQR problem by defining an augmented state vector $\mathbf{x}^{\mathrm{tr}} := [\mathbf{x}^\top, \mathbf{x}^{\mathbf{r}\top}]^\top \in \mathbb{R}^{n(H^{\mathbf{r}}+1)}$ and associated state-space equation

$$\mathbf{x}^{\mathrm{tr}}_{k+1} = \begin{bmatrix} A & 0 \\ 0 & A^{\mathbf{r}} \end{bmatrix}\mathbf{x}^{\mathrm{tr}}_k + \begin{bmatrix} B \\ 0 \end{bmatrix}\mathbf{u}_k + \begin{bmatrix} I & 0 \\ 0 & B^{\mathbf{r}} \end{bmatrix}\begin{bmatrix} \mathbf{w}_k \\ \mathbf{n}_k \end{bmatrix}. \tag{16}$$

This leads to $\mathbf{x}_k - \mathbf{r}_k = \begin{bmatrix} I & -C^{\mathbf{r}} \end{bmatrix}\mathbf{x}^{\mathrm{tr}}_k$ and therefore a finite horizon LQR problem with $Q^{\mathrm{tr}} := \begin{bmatrix} I & -C^{\mathbf{r}} \end{bmatrix}^\top Q \begin{bmatrix} I & -C^{\mathbf{r}} \end{bmatrix}$ and $P^{\mathrm{tr}}_H := \begin{bmatrix} I & -C^{\mathbf{r}} \end{bmatrix}^\top P_H \begin{bmatrix} I & -C^{\mathbf{r}} \end{bmatrix}$. In general, this

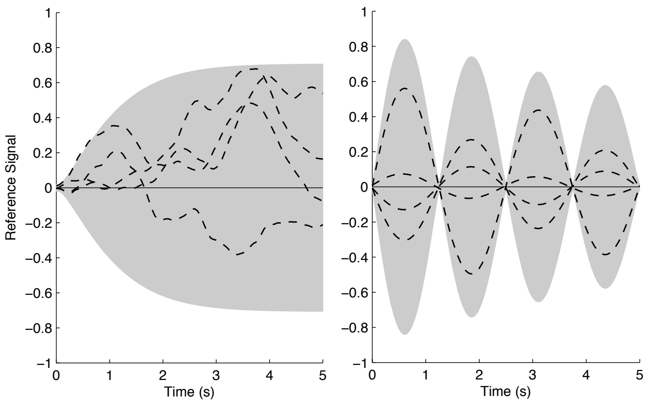

Fig. 4. Illustration of the type of trajectories that can be modelled by the state space equation (15). The left hand plot depicts second-order filtered white noise. The right hand plot depicts a deterministic decaying sinuosoid model of a specified frequency with a Gaussian distribution over the starting state $\mathbf{r}_0$. The solid black lines denote the mean, the shaded area is the 95% confidence region and the dashed lines are sample trajectories.

leads to a linear time-varying control law K^{tr}_{H-k-1} which is undesirable in practice. However, applying $K = K^{\text{tr}}_{H-1}$ as a time-invariant control law yields the associated *receding horizon* policy.

III. THE ALGORITHM

The goal of the proposed algorithm is to learn a time-invariant parameterised control policy $\mathbf{u} = \pi(\mathbf{x}^{\text{tr}})$ by solving a finite horizon optimisation problem equivalent to the one in Section II-C. This policy can be viewed as an approximation to the optimal time-varying solution. It may have any structure provided that the first and second moments of the output distribution can be calculated analytically as well as the input-output covariance, given a Gaussian distribution over the augmented state $\mathbf{x}^{\text{tr}}$. The policy parameters shall be gathered under the variable $\boldsymbol{\psi}$.

There are three layers to the algorithm:

- Learn the transition dynamics of the system.
- Evaluate the (approximate) expected long-term cost of following some policy π.
- Improve the policy by finding the optimal policy π^* in the parameterized set defined by $\boldsymbol{\psi}$.

The learned transition dynamics model is used as the basis of a standard optimisation problem over policy parameters. Once the optimiser has converged the new policy is applied to the system and the dynamics model is updated based on this new information. This process is repeated until the policy parameters have converged. A more detailed breakdown is as follows.

A. Learn Dynamics

The first step of the process is to generate a data set $\mathcal{D}$ by applying an arbitrary policy to the system. This data set is then used to train a Gaussian process model of the transition dynamics. After each optimisation of the policy a trajectory is sampled from the reference class and the system is run with the new policy and this sampled reference. The data generated from this test run is then used to update the learned GP model and the hyperparameters are re-trained.

B. Policy Evaluation

Policy evaluation was carried out in an analogous manner to the LQ tracking method outlined in Section II-C. The aim is to learn policy parameters of a given feedback policy structure that perform well over a given class of reference trajectories given by the linear state space system (15).

1) Policy Structure: The policy structure that was investigated in this paper was a quadratic controller. This is defined by augmenting the state with the pairwise product of each state as follows

$$\mathbf{u} = \tilde{\pi}(\mathbf{x}^{\text{tr}}) = [K, L] \begin{bmatrix} \mathbf{x}^{\text{tr}} \\ \mathbf{z}^{\text{tr}} \end{bmatrix} \tag{17}$$

where $\mathbf{z}^{\text{tr}} \in \mathbb{R}^{0.5q(q+1)}$, $q = (H^{\mathbf{r}} + 1)n$ contains all pairwise products $x^{\text{tr}}_i x^{\text{tr}}_j, i, j \in \{1 \dots q\}$ and $L \in \mathbb{R}^{m \times 0.5q(q+1)}$. This is a richer class of control structure than the standard linear mapping but includes it as a special case by setting $L = 0$. The parameter set is then defined by $\boldsymbol{\psi} = \{K, L\}$.

In order to ensure the algorithm respected the system input constraints the output of the policy was squashed through a sine to give a bounded policy output $\pi(\mathbf{x}^{\text{tr}}) = u_{\text{max}} \sin\left(\tilde{\pi}(\mathbf{x}^{\text{tr}})\right)$. This operation allows analytic evaluation of the mean and variance of the control input given a Gaussian state distribution $\mathbf{x}^{\text{tr}} \sim \mathcal{N}$.

2) Cost Function: Since the policy is bounded through the sine it was not necessary to penalise the input action, therefore the following cost function was used to evaluate performance

$$V^{\boldsymbol{\psi}}(H, \mathbf{x}_0) = \sum_{k=0}^{H} \mathbb{E}_{\mathbf{x}^{\text{tr}}_k} \left[c\left(\mathbf{x}^{\text{tr}}_k, Q^{\text{tr}}\right) \right]. \tag{18}$$

The stage cost $c(\cdot, \cdot)$ was given by the saturating quadratic $c(\mathbf{a}, T) = 1 - \exp\left(-\mathbf{a}^\top T \mathbf{a}\right)$ instead of the standard quadratic cost. This stage cost was chosen rather than the quadratic since it was shown in [3] that it is more conducive to the learning process because it encodes the notion of being "far away" from the target state while not caring exactly how far. Furthermore, it shares with the quadratic cost the useful property that $\mathbb{E}_{\mathbf{a}}[c(\mathbf{a}, \cdot)]$ can be evaluated analytically given $\mathbf{a} \sim \mathcal{N}$.

Evaluating this cost function for a given setting of the policy parameters $\boldsymbol{\psi}$ involves propagating the approximated Gaussian distribution over the augmented state at the current timestep, $p(\mathbf{x}^{\text{tr}}_k)$ to the distribution at the next timestep $p(\mathbf{x}^{\text{tr}}_{k+1})$. This was achieved through the following three steps:

- Determine $p(\mathbf{u}_k)$ using the parameterised policy with input $p(\mathbf{x}^{\text{tr}}_k)$ where uncertainty propagation depends on the specific policy structure.
- Determine $p(\mathbf{x}_{k+1})$ using the Gaussian process transition dynamics model with input $p(\mathbf{x}_k, \mathbf{u}_k)$ and propagating the uncertainty using results from Section II-A.3.

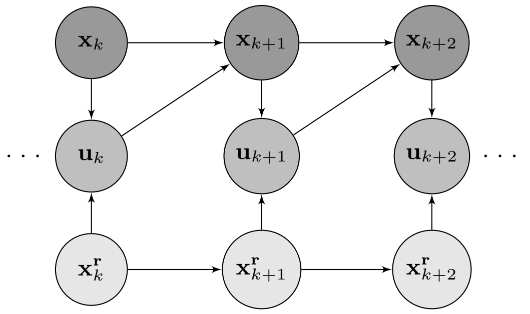
Fig. 5. Graphical model representation of the propagation of uncertainty over current state, reference state and input $p(\mathbf{x}_k^{\text{tr}}, \mathbf{u}_k)$ to the distribution at the following timestep $p(\mathbf{x}_{k+1}^{\text{tr}}, \mathbf{u}_{k+1})$ etc given the two stochastic processes: dynamics and reference.

- Determine $p(\mathbf{x}_{k+1}^{\mathbf{r}})$ using the linear state space model (14) with input $p(\mathbf{x}_k^{\mathbf{r}})$ and propagating the uncertainty using standard Kalman filtering equations.

The cross covariance terms that are not evaluated explicitly in these steps are found using the conditional independence method outlined in Section II-B since $\mathbf{x}_{k+1} \perp\!\!\!\perp \mathbf{x}_k^{\mathbf{r}} | (\mathbf{x}_k, \mathbf{u}_k)$ and $\mathbf{x}_{k+1}^{\mathbf{r}} \perp\!\!\!\perp (\mathbf{x}_k, \mathbf{u}_k) | \mathbf{x}_k^{\mathbf{r}}$. These conditional independence relationships can be confirmed by the graphical model (see Chapter 8 of [1] for an introduction) in Figure 5.

C. Policy Improvement

Using this method of evaluating the policy parameters $\boldsymbol{\psi}$, policy improvement can be achieved by optimisation. The aim is to find the parameters that satisfy

$$\boldsymbol{\psi}^* = \underset{\boldsymbol{\psi}}{\operatorname{argmin}} V^{\boldsymbol{\psi}}(H, \mathbf{x}_0) \tag{19}$$

for a given dynamics model. Since all distributions are (approximate) Gaussians the derivatives of $V^{\boldsymbol{\psi}}(H, \mathbf{x}_0)$ with respect to $\boldsymbol{\psi}$ can be evaluated analytically. Taking $p(\mathbf{x}_k^{\text{tr}}) \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, the derivatives are given by

$$\begin{aligned}\frac{\mathrm{d}V^{\boldsymbol{\psi}}(H, \mathbf{x}_0)}{\mathrm{d}\boldsymbol{\psi}} = \sum_{k=1}^{H} \Bigg[\left(\frac{\partial}{\partial \boldsymbol{\mu}_k} \mathbb{E}_{\mathbf{x}_k^{\text{tr}}} [c(\cdot, \cdot)] \right) \frac{\mathrm{d}\boldsymbol{\mu}_k}{\mathrm{d}\boldsymbol{\psi}} \\ + \left(\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \mathbb{E}_{\mathbf{x}_k^{\text{tr}}} [c(\cdot, \cdot)] \right) \frac{\mathrm{d}\boldsymbol{\Sigma}_k}{\mathrm{d}\boldsymbol{\psi}} \Bigg].\end{aligned} \tag{20}$$

The mean and covariance derivatives at each timestep are calculated using the recursive rule

$$\frac{\mathrm{d}\boldsymbol{\alpha}_k}{\mathrm{d}\boldsymbol{\psi}} = \frac{\partial \boldsymbol{\alpha}_k}{\partial \boldsymbol{\mu}_{k-1}} \frac{\mathrm{d}\boldsymbol{\mu}_{k-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial \boldsymbol{\alpha}_k}{\partial \boldsymbol{\Sigma}_{k-1}} \frac{\mathrm{d}\boldsymbol{\Sigma}_{k-1}}{\mathrm{d}\boldsymbol{\psi}} + \frac{\partial \boldsymbol{\alpha}_k}{\partial \boldsymbol{\psi}}, \tag{21}$$

where $\boldsymbol{\alpha} \in \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. The availability of these derivatives means that optimisation can be achieved using standard gradient descent methods.

IV. SIMULATIONS

A. Inverted Pendulum

1) Setup: The algorithm was applied to the inverted pendulum problem as shown in the Figure 6. The state of this system is defined as the cart position x, cart velocity $\dot{x}$, pole position θ and pole velocity $\dot{\theta}$ giving $\mathbf{x} = [x, \dot{x}, \theta, \dot{\theta}]^\top$. The input $\mathbf{u}$ is simply the lateral force applied to the cart u. The nonlinear equations of motion are given by

$$(M + m)\ddot{x} + \tfrac{1}{2}Ml\ddot{\theta}\cos\theta - \tfrac{1}{2}Ml\dot{\theta}^2 \sin\theta = u - b\dot{x} \tag{22}$$

$$2l\ddot{\theta} + 3\ddot{x}\cos\theta + 3g\sin\theta = 0 \tag{23}$$

where the following physical constants were set (as used by [7]) $M = 0.5\,\text{kg}$ the mass of the cart, $m = 0.5\,\text{kg}$ the mass of the pole, $b = 0.1\,\text{N}\,\text{s}\,\text{m}^{-1}$ the friction between cart and track, $l = 0.6\,\text{m}$ the length of the pole and $J = 0.06\,\text{kg}\,\text{m}^2$ the moment of inertia around the tip of the pole. The control input u was constrained to the range $u \in [-u_{\text{max}}, u_{\text{max}}] = [-10, 10]\,\text{N}$ and was implemented as a zero-order hold with discrete timestep length $\Delta t = 0.15\,\text{s}$. The prediction horizon was set as $H = 3\,\text{s}$ and the state weighting matrix Q was defined so as to penalise the approximate squared Euclidean distance from the tip of the pendulum to the reference point.

The reference trajectory was defined as a change in the x-position of the cart with the pendulum upright. The class of reference was chosen to be first order filtered white noise with parameters $a = 0.98$, $A^{\text{f}} = a$, $B^{\text{f}} = l\sqrt{1 - a^2}$. These parameters were chosen so that the limiting standard deviation of the reference was equal to the length of the pendulum l, as this is an appropriate length scale.

The Gaussian process model of the transition dynamics was initialised using $6\,\text{s}$ of training data generated from interaction with the system by applying random inputs drawn from a uniform distribution. The learning process was stopped after 8 iterations of the algorithm by which time the policy parameters had normally converged. This is a total interaction time of $30\,\text{s}$.

2) Results: Policies were trained for timestep horizons $H^{\mathbf{r}} \in \{1 \ldots 7\}$ with a discrete timestep length of $\Delta t = 0.15\,\text{s}$. The learned controllers were then run on 200 sample trajectories of $3\,\text{s}$ in length drawn from the stochastic system and the average stage cost was calculated for each. These were compared with a controller derived using a discrete linearised form of the system dynamics and the receding horizon LQ controller outlined in Section II-C. When applied online, the output was saturated at u_{max} in order to obtain a fair comparison. Setting the input weighting to $R = 5 \times 10^{-4}$ gave the best results. The results are displayed in Figure 7.

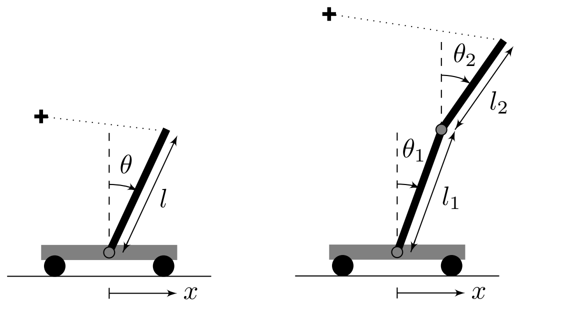

Fig. 6. The pendulum and the double pendulum tracking scenarios. The black cross indicates the reference point and the dotted line shows the Euclidian distance from this point to the tip of the pendulum.

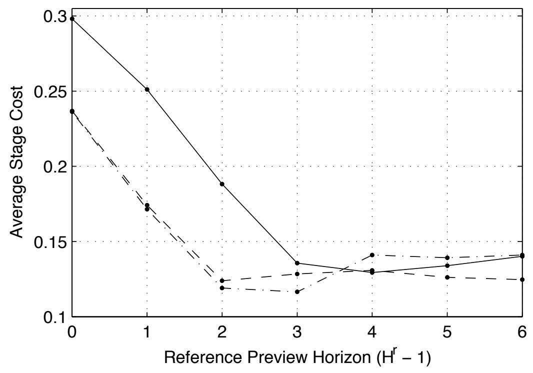

Fig. 7. The average one-step cost for different policies evaluated over 200 sample reference trajectories of 3 s each. The solid line depicts the results for the LQ controller, the dash-dot line depicts the learned linear policy ($L = 0$) and the dashed line depicts the learned quadratic policy.

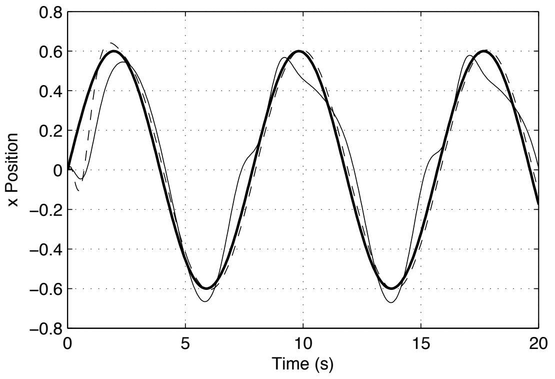

Fig. 8. The x-position of the cart for the double pendulum as it tracks a sinusoidal trajectory (thick solid) using an LQ derived policy (dashed) and a learned quadratic policy (solid).

It can be observed from Figure 7 that both the linear and the quadratic learned policies yield improved or comparable performance to the LQ controller. This is probably due to the fact that the algorithm has taken into account the nonlinear dynamics of the system and has adopted a principled treatment of input constraints. The inclusion of the quadratic terms in this case has given, essentially, no further improvement on the linear policy, since the system has been operating mainly in the linear regime.

B. Double Inverted Pendulum

1) Setup: The algorithm was then run on the double inverted pendulum tracking problem as shown in Figure 6. The equations of motion for this system can be found in Appendix B of the thesis [3]. The parameters of this system were set to be the same as in the single pendulum case except with the addition of a pendulum of the same length and weight attached to the end of the first one $l_1 = l_2 = l$. The sampling time was reduced to $\Delta t = 0.05\,\mathrm{s}$. The prediction horizon was set as $H = 5\,\mathrm{s}$ and the state weighting matrix Q was again defined to penalise approximately the squared Euclidean distance from the tip of the second pendulum to the reference point as shown in Figure 6.

The algorithm was trained for tracking sine waves with frequency of $0.8\,\mathrm{rad\,s^{-1}}$, amplitude distributed according to $a \sim \mathcal{N}(0, l)$ and access to the position and rate of change of the sine wave (since this fully characterises the sinusoid there is no need for further future information).

2) Results: It was found that the the algorithm could not find a stabilising solution using the linear policy but within a few iterations could find one using the quadratic structure. The learned policy was then compared with a policy derived from the standard LQ method as demonstrated in Figure 8. In this case the learned policy is able to achieve the task but is not able to match the performance of the LQ policy.

V. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

The contribution of this paper has been to present a new algorithm for learning reference tracking controllers given no prior knowledge of the dynamical system and limited interaction through the learning process. Concepts from the field of reinforcement learning, Bayesian statistics and classical control have been brought together to create this algorithm. On the task of reference tracking using the inverted pendulum it was shown to yield generally improved performance on the best controller derived from the standard linear quadratic method using only $30\,\mathrm{s}$ of total interaction with the system. Finally, the algorithm was shown to work on the double pendulum proving the ability to solve highly nontrivial tasks.

B. Future Work

As it stands, the algorithm learns the dynamics of a system and assumes a given linear model for the reference dynamics. However, the framework is applicable to the inclusion of a nonlinear reference model and to learning both the dynamics of the system and a nonlinear reference model. In the future we plan to show application of this.

REFERENCES

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] R. R. Bitmead, M. Gevers, and V. Wertz. *Adaptive Optimal Control: The Thinking Man's GPC*. Prentice Hall International Series in Systems and Control Engineering. Prentice Hall, 1990.

[3] M. P. Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes*. PhD thesis, Cambridge University, November 24 2009.

[4] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[5] J. Quiñonerno-Candela, A. Girard, J. Larsen, and C. E. Rasmussen. Propagation of uncertainty in Bayesian kernel models - Application to multiple-step ahead forecasting. In *In Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP03)*, volume 2, pages 701–704, Hong Kong, April 2003.

[6] T. Raiko and M. Tornio. Variation Bayesian learning of nonlinear hidden state-space models for model predictive control. *Neurocomputing*, 72(16–18):3704–3712, October 2009.

[7] C. E. Rasmussen and M. P. Deisenroth. Probabilistic inference for fast learning in control. In *Recent Advances in Reinforcement Learning: 8th European Workshop*, pages 229–242, Berlin, Germany, November 2008. Springer.

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

[9] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.

[10] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, Cambridge, UK, 1989.