Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Applying Gaussian Processes to Reinforcement Learning for Fixed-Structure Controller Synthesis

Hildo Bijl, Jan-Willem van Wingerden & Michel Verhaegen

Delft Center for Systems and Control, Delft University of Technology, The Netherlands, {h.j.bijl,j.w.vanwingerden,m.verhaegen}@tudelft.nl

Gaussian processes; Continuous systems; Learning algorithms; Learning control; Self-tuning control.

Abstract: In industrial applications, fixed-structure controllers are often desired. But for systems with large uncertainties, or for systems with mostly unknown system dynamics, it is often unclear as to how to choose the controller parameters. In this paper we propose an algorithm that chooses the parameters of such a controller using only a limited amount of system interaction data. The novel algorithm applies Gaussian process tools to a reinforcement learning problem set-up to derive an approximation of the value function. This approximation is expressed in the system state and in the controller parameters. Then, by assuming a distribution of the initial state of the system, the value function approximation is expressed only as a function of the controller parameters. By subsequently optimizing this value function approximation, the optimal controller parameters with respect to the value function approximation can be found. The effectiveness of the proposed methodology has been shown in a simulation study.

1. INTRODUCTION

In industrial applications one often encounters (possibly nonlinear) continuous-time systems with continuous input and action spaces and a fixed controller architecture. It is assumed that the state can be measured. In this case the system f can be written as

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad \text{with} \quad \boldsymbol{u}(t) = C(\boldsymbol{x}(t), \boldsymbol{\theta}), \qquad (1)$$

where $\boldsymbol{x}$ is the state vector, $\boldsymbol{u}$ is the input vector, C is the (fixed) control law and $\boldsymbol{\theta}$ is a constant vector of to-be-determined controller parameters.

While it may be possible to exactly measure the state $\boldsymbol{x}$, the exact system f is not always known. It can be approximated by a model $\dot{\boldsymbol{x}} \approx \bar{f}(\boldsymbol{x}, \boldsymbol{u})$, but this model $\bar{f}$ generally differs from the unknown system f. This can be due to inaccuracies in modelling, manufacturing inaccuracies, external influences like temperature, ageing effects like wear, or some other reason. In this paper we propose an algorithm capable of dealing with such uncertainties.

1.1 Methods to deal with uncertainties

A conventional way of dealing with uncertainties is through robust control, as described by Skogestad and Postlethwaite [2005]. In this case hard bounds on the uncertainties are assumed to be known. Subsequently a controller is designed that is guaranteed to have a certain performance level for any possible system f within these bounds. This works well if the uncertainties are relatively small. However, if the possible differences between the model $\bar{f}$ and the system f become big, creating a controller that has sufficient performance for all possible systems $\bar{f}$ is not always possible.

A different way to tackle this problem is by using system measurements. Subsequently, it is possible to apply System Identification (SI) methods, as described by Verhaegen and Verdult [2007], to develop a system model. This model is then used to develop a controller. For linear systems, the theory on system identification is very well developed. For nonlinear systems, this is less so. In this case it generally requires a lot of system data before a model with sufficient accuracy is obtained. For industrial applications, obtaining a lot of system data is not always possible mainly due to financial reasons. Hence the amount of available system data is usually strongly limited. As a result, the system model will have relatively large uncertainties as well. Next to this, applying SI with a small data set is likely to result in model bias, as argued by Deisenroth and Rasmussen [2011]. So in this case it may be beneficial to skip the SI step altogether.

Not using a system model at all is something that is often done in Reinforcement Learning (RL) applications. (See the work by Sutton and Barto [1998] for background theory on RL.) For instance, in Q-learning by Watkins [1989] the RL algorithm directly learns the Q-value of applying a certain input $\boldsymbol{u}$ (or action a) in a certain state $\boldsymbol{x}$ (or s). From this, the optimal input $\boldsymbol{u}$ can be obtained. The problem with the Q-learning algorithm and most other reinforcement learning algorithms, is that they are traditionally designed for discrete-time systems with discrete state and action spaces.

There are many RL algorithms that expand on this. For instance, Bertsekas and Tsitsiklis [1996] describe algorithms that can be applied to systems with continuous state and action spaces. They use various types of function approximators to deal with the continuous states and actions in

the (discrete-time) systems. When the value function is approximated by a linear combination of basis functions, then the algorithm generally works well. Most such algorithms have proven convergence and result in a reasonable approximation of the value function, despite the limitations of the linear function approximator. Vrabie et al. [2013] describe an algorithm that expands these methods to the continuous time domain, but this algorithm requires the system dynamics to be affine in the input.

Another challenge is that RL often has the drawback of being data-inefficient. That is, RL algorithms require a large number of trials to learn a particular task. For many industrial problems in which system data is expensive to obtain, RL algorithms are therefore not suitable. An exception to this rule occurs when the tools used to model Gaussian Processes (GP) are implemented. (Rasmussen and Williams [2006] give an introduction into such tools.) These GP tools can be applied as function approximator, and because they take into account the uncertainties of their approximations and cleverly incorporate prior knowledge, they are generally known for their efficient use of input data.

1.2 Combinations of RL and GP in literature

Reinforcement learning and Gaussian processes have not been combined often in literature. Richard et al. [1998] did approximate a Q-function by using probability distributions for the Q-values. However, their algorithm worked for discrete state and action spaces, so no Gaussian process tools had to be used.

The first significant instance of applying GP to RL problems was by Rasmussen and Kuss [2004]. Though promising, the proposed algorithm applies system identification through GP, in which every state parameter is independently modeled by a separate GP. This results in a highly computationally intensive algorithm. Next to that, the algorithm requires a significant number of maximizations of a Gaussian process during each policy iteration, which is also computationally problematic. This strongly limits the practical applicability of the algorithm.

Another combination of GP and RL was by Engel et al. [2005]. In their article the authors come up with a method of determining the value function that is very similar to the way that we will propose here. However, they have not looked at an efficient method of selecting an action. Their suggestion is maximizing the Gaussian process value function with respect to the input $\boldsymbol{u}$ at each time step. This comes down to solving a nonlinear optimization problem at each time step, which is likely to be infeasible in practice.

Probably the most promising combination of GP and RL is by Deisenroth and Rasmussen [2011]. Their PILCO algorithm (Probabilistic Inference for Learning Control) uses Gaussian processes to learn a system model, instead of a value function. Because of this, PILCO is very suitable for higher-dimensional problems, more so than the algorithm presented in this paper. However, PILCO only works for specific types of fixed-structure controllers, which is a limitation. The algorithm presented in this paper is able to tune controller parameters for any fixed-structure state controller $C(\boldsymbol{x}, \boldsymbol{\theta})$.

1.3 Set-up of this paper

This paper is structured as follows. Section 2 discusses the theory behind the proposed algorithm. Once the theory is in place, section 3 gives an overview of the algorithm. In section 4 an application of the algorithm is described. Section 5 closes off the paper with conclusions and a discussion of the strengths and weaknesses of the novel algorithm.

2. UNDERLYING THEORY

In this section we examine how to apply Gaussian processes to reinforcement learning problems. In subsection 2.1 we present the problem from a reinforcement learning perspective. We briefly introduce Gaussian process tools in subsection 2.2 and apply them to our problem set-up in subsection 2.3. Finally, in subsection 2.4, we examine how we can use our data to select controller parameters.

2.1 Reinforcement learning set-up

To be able to judge the effectiveness of certain controller parameters $\boldsymbol{\theta}$, we need an optimality criterion. In reinforcement learning applications, a common choice is the value function V, describing the value of a certain state $\boldsymbol{x}(t)$ with respect to a given control law. It does this according to

$$\begin{aligned} V(\boldsymbol{x}(t), \boldsymbol{\theta}) &= \frac{\int_t^\infty \gamma^{\tau - t} r(\boldsymbol{x}(\tau), C(\boldsymbol{x}(\tau), \boldsymbol{\theta}))\, d\tau}{\int_t^\infty \gamma^{\tau - t}\, d\tau} \\ &= (-\log(\gamma)) \int_t^\infty \gamma^{\tau - t} r(\boldsymbol{x}(\tau), C(\boldsymbol{x}(\tau), \boldsymbol{\theta}))\, d\tau, \end{aligned} \tag{2}$$

where $\gamma < 1$ is a constant discount factor and r is a reward function chosen to encourage certain types of behavior. In the above relation, we have decided to divide by the (constant) sum of the discount factors. This has (among others) the benefit that it will shorten some of the subsequent equations.

Suppose now that a trial time step is performed. This experiment starts in state $\boldsymbol{x}(t)$, lasts T seconds (with T usually small) and puts the system in state $\boldsymbol{x}(t+T)$. In this case the expression for the value function can be rewritten to

$$\begin{aligned} V(\boldsymbol{x}(t), \theta) = (-\log(\gamma)) \int_t^{t+T} \gamma^{\tau - t} r(\boldsymbol{x}(\tau), C(\boldsymbol{x}(\tau), \boldsymbol{\theta}))\, d\tau \\ + \gamma^T V(\boldsymbol{x}(t+T), \theta). \end{aligned} \tag{3}$$

Note that T in the term γ^T is not a transpose but an exponent. After all, γ is a scalar.

The above integral is usually not analytically solvable. In such a case it is often easier to approximate the (varying) reward by a constant (mean) reward $\bar{r}$ throughout the trial time step T. Since T is often small, such an approximation can be justified. The above equation then reduces to

$$\begin{aligned} V(\boldsymbol{x}(t), \boldsymbol{\theta}) = (1 - \gamma^T) \bar{r}(\boldsymbol{x}(t), C(\boldsymbol{x}(t), \boldsymbol{\theta})) \\ + \gamma^T V(\boldsymbol{x}(t+T), \boldsymbol{\theta}). \end{aligned} \tag{4}$$

This recursive relation will prove to be useful when Gaussian process tools will be implemented.

2.2 A brief introduction into Gaussian process tools

In literature on reinforcement learning, it is customary to use a function approximator to approximate the value function. We will do something similar here. We will assume that the value function V is a Gaussian process, and consequentially approximate it using the tools developed to describe Gaussian processes. An important advantage of using Gaussian processes is that we do not only get an estimate of the value function V, but also of the uncertainty (i.e. the variance) of our approximation.

Let us briefly look at the underlying theory of Gaussian processes. When we work with Gaussian processes, we usually do measurements y of a function $h(\boldsymbol{z})$, with $\boldsymbol{z}$ the (vector) input and y the scalar output. We assume that these measurements are corrupted by white noise with intensity σ_n^2. We can assemble all n sets of measurement data $(\boldsymbol{z}, y)$ into a matrix Z and a vector $\boldsymbol{y}$ like

$$Z = [\boldsymbol{z}_1 \ \cdots \ \boldsymbol{z}_n] \quad \text{and} \quad \boldsymbol{y} = [y_1 \ \cdots \ y_n]^T. \tag{5}$$

In the above definition, the indices indicate the measurement number. Subsequently, we define a prior mean function $m(\boldsymbol{z})$ and a prior covariance function $k(\boldsymbol{z}, \boldsymbol{z'})$ for the function h, where $\boldsymbol{z}$ and $\boldsymbol{z'}$ can be any function inputs to h. Customary functions are the zero mean function $m(\boldsymbol{z}) = 0$ and the squared exponential covariance function

$$k(\boldsymbol{z}, \boldsymbol{z'}) = \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{z} - \boldsymbol{z'})^T \Sigma_d^{-1} (\boldsymbol{z} - \boldsymbol{z'})\right), \tag{6}$$

with Σ_d a diagonal matrix of (squared) length scales.

Now examine a point $\boldsymbol{z_*}$ for which we want to estimate $h(\boldsymbol{z_*})$, or alternatively the corresponding (noise-corrupted) measurement y_*. From our assumptions it follows that

$$\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(Z) \\ m(\boldsymbol{z_*}) \end{bmatrix}, \begin{bmatrix} k(Z,Z) & k(Z,\boldsymbol{z_*}) \\ k(\boldsymbol{z_*},Z) & k(\boldsymbol{z_*},\boldsymbol{z_*}). \end{bmatrix} + \sigma_n^2 I\right), \tag{7}$$

with $\mathcal{N}(.,.)$ denoting a multivariate Gaussian distribution. If we use the fact that the measurements $\boldsymbol{y}$ are known, then we can derive a distribution for y_*. Denoting $k(Z,Z) = K$, $k(Z, \boldsymbol{z_*}) = K_*$ and $k(\boldsymbol{z_*}, \boldsymbol{z_*}) = K_{**}$, we get

$$\begin{aligned} y_* | \boldsymbol{y} \sim \mathcal{N}(&m(\boldsymbol{z_*}) + K_*^T (K + \sigma_n^2 I)^{-1} (\boldsymbol{y} - m(Z)), \\ &K_{**} - K_*^T (K + \sigma_n^2 I)^{-1} K_* + \sigma_n^2 I). \end{aligned} \tag{8}$$

2.3 Applying Gaussian processes to reinforcement learning

Equation (8) is a familiar equation in Gaussian process theory. However, it cannot be directly applied to reinforcement learning problems. This is because in RL we do not get (noisy) measurements of the value function $V(\boldsymbol{x}, \boldsymbol{\theta})$. Instead, at every trial time step we find a relation between points in the value function. We can rewrite such a relation, equation (4), to

$$\begin{bmatrix} 1 & -\gamma^T \end{bmatrix} \begin{bmatrix} V(\boldsymbol{x^i}, \theta) \\ V(\boldsymbol{x^f}, \theta) \end{bmatrix} = \begin{bmatrix} (1-\gamma^T)\bar{r} \end{bmatrix}, \tag{9}$$

where, for the sake of compactness of notation, we have shortened the initial state $\boldsymbol{x}(t)$ to $\boldsymbol{x^i}$, the final state $\boldsymbol{x}(t+T)$ to $\boldsymbol{x^f}$, and we have dropped the brackets after $\bar{r}$. If we do an additional trial time step, then the above turns into

$$\begin{bmatrix} 1 & -\gamma^{T_1} & 0 & 0 \\ 0 & 0 & 1 & -\gamma^{T_2} \end{bmatrix} \begin{bmatrix} V(\boldsymbol{x_1^i}, \boldsymbol{\theta_1}) \\ V(\boldsymbol{x_1^f}, \boldsymbol{\theta_1}) \\ V(\boldsymbol{x_2^i}, \boldsymbol{\theta_2}) \\ V(\boldsymbol{x_2^f}, \boldsymbol{\theta_2}) \end{bmatrix} = \begin{bmatrix} (1-\gamma^{T_1})\bar{r}_1 \\ (1-\gamma^{T_2})\bar{r}_2 \end{bmatrix}, \tag{10}$$

where the index denotes the trial number. In this way, we can expand the above equation for any number of trial time steps, even when these trials have different durations T.

What we can do next is write the above relation as $M\boldsymbol{V} = \bar{\boldsymbol{r}}$, with M, $\boldsymbol{V}$ and $\bar{\boldsymbol{r}}$ defined accordingly. Subsequently, given that $M\boldsymbol{V} = \bar{\boldsymbol{r}}$, it can be derived (an outline of this derivation is given in appendix A) that $\boldsymbol{V}$ is distributed according to

$$\begin{aligned} \boldsymbol{V} | (M\boldsymbol{V} = \bar{\boldsymbol{r}}) \sim \mathcal{N}(&KM^T (MKM^T + \Sigma_n)^{-1} \bar{\boldsymbol{r}}, \\ &K - KM^T (MKM^T + \Sigma_n)^{-1} MK). \end{aligned} \tag{11}$$

Note that, for compactness of notation, we have assumed that the mean $m(\boldsymbol{x}, \boldsymbol{\theta}) = 0$. If this is not the case, $m(\boldsymbol{x}, \boldsymbol{\theta})$ can still be incorporated in the above relation. The matrix Σ_n is the covariance matrix of $\bar{\boldsymbol{r}}$. In this article we assume that $\Sigma_n = (1-\gamma^T)^2 \sigma_n^2 I$, where σ_n^2 is the variance of the noise present when measuring the mean reward $\bar{r}$.

The matrix K in the above equation is found through a covariance function $k(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{x'}, \boldsymbol{\theta'})$. In the case where we only have one trial time step, K equals

$$K = \begin{bmatrix} k(\boldsymbol{x_1^i}, \boldsymbol{\theta_1}, \boldsymbol{x_1^i}, \boldsymbol{\theta_1}) & k(\boldsymbol{x_1^i}, \boldsymbol{\theta_1}, \boldsymbol{x_1^f}, \boldsymbol{\theta_1}) \\ k(\boldsymbol{x_1^f}, \boldsymbol{\theta_1}, \boldsymbol{x_1^i}, \boldsymbol{\theta_1}) & k(\boldsymbol{x_1^f}, \boldsymbol{\theta_1}, \boldsymbol{x_1^f}, \boldsymbol{\theta_1}) \end{bmatrix}. \tag{12}$$

If we have performed n trial time steps, then K expands accordingly to a $2n \times 2n$ covariance matrix.

It is customary to decouple the covariance function $k(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{x'}, \boldsymbol{\theta'})$ into $k_x(\boldsymbol{x}, \boldsymbol{x'}) k_\theta(\boldsymbol{\theta}, \boldsymbol{\theta'})$. Subsequently, we can use a squared exponential function for both k_x and k_θ. This would result in

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{x'}, \boldsymbol{\theta'}) = \sigma_f^2 \exp\Bigg(&-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x'})^T \Sigma_d^{-1} (\boldsymbol{x} - \boldsymbol{x'}) \\ &-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta'}) \Sigma_t^{-1} (\boldsymbol{\theta} - \boldsymbol{\theta'})\Bigg). \end{aligned} \tag{13}$$

Now the framework is in place to give us an estimate of the value $V(\boldsymbol{x_*}, \boldsymbol{\theta_*})$ for some state $\boldsymbol{x_*}$ and set of controller parameters $\boldsymbol{\theta_*}$. To get this estimate, we can add $V(\boldsymbol{x_*}, \boldsymbol{\theta_*})$ to the value vector $\boldsymbol{V}$, add a corresponding column of zeros to the constraint matrix M and add a column and row to K. Next, we can apply relation (11) to derive the posterior distribution of $\boldsymbol{V}$ and hence of $V(\boldsymbol{x_*}, \boldsymbol{\theta_*})$. If we do this for multiple points, we can plot $E[V(\boldsymbol{x}, \boldsymbol{\theta})]$ with respect to $\boldsymbol{x}$ and/or $\boldsymbol{\theta}$.

2.4 Determining the expected value of controller parameters

With the method described previously, it is possible to approximate the value $V(\boldsymbol{x}, \boldsymbol{\theta})$, as a function of the state $\boldsymbol{x}$ and a set of controller parameters $\boldsymbol{\theta}$. By maximizing $V(\boldsymbol{x}, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, for a constant $\boldsymbol{x}$, it is possible to find the optimal set of controller parameters for each state. That is, as much as is allowed by the accuracy of the approximation of $V(\boldsymbol{x}, \boldsymbol{\theta})$.

However, for our problem we cannot let the controller parameters/gains vary with each state. Instead, a controller with constant gains is required. Choosing the optimal $\boldsymbol{\theta}$ now takes place differently.

Suppose that, from experience, it is known that the prior state $\boldsymbol{x}$ is distributed according to $p(\boldsymbol{x})$. Then the set of

optimal controller parameters $\boldsymbol{\theta_{opt}}$, that (per definition) maximizes the expected value, is found through

$$\boldsymbol{\theta_{opt}} \equiv \arg\max_{\boldsymbol{\theta}} \int_X V(\boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{x})\, d\boldsymbol{x}. \tag{14}$$

Finding an analytic solution for this relation is generally not possible. However, if we use a squared exponential covariance function like in equation (13), and if $p(\boldsymbol{x})$ also has a Gaussian distribution – that is, $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu_x}, S)$ – then the integral in equation (14) can be solved analytically. The mathematics behind this are explained by Girard et al. [2003] and Candela et al. [2003]. Application results in

$$\begin{aligned}\bar{V}(\boldsymbol{\theta}) \equiv\ & E\left[V(\boldsymbol{x}, \boldsymbol{\theta}) | \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu_x}, S)\right] = \\ & \boldsymbol{l}^T M^T (MKM^T + \sigma_n^2 I)^{-1} \bar{\boldsymbol{r}},\end{aligned} \tag{15}$$

where the elements l_i of the column vector $\boldsymbol{l}$ depend both on the current controller parameters $\boldsymbol{\theta}$ and the respective experiment input data $(\boldsymbol{x_i}, \boldsymbol{\theta_i})$. In fact, they are given by

$$\begin{aligned}l_i = & k_S(\boldsymbol{x_i}, \boldsymbol{\theta_i}, \boldsymbol{\mu_x}, \boldsymbol{\theta}) \equiv \\ & \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x_i} - \boldsymbol{\mu_x})^T (\Sigma_d + S)^{-1} (\boldsymbol{x_i} - \boldsymbol{\mu_x}) \right. \\ & \left. -\frac{1}{2}(\boldsymbol{\theta_i} - \boldsymbol{\theta}) \Sigma_t^{-1} (\boldsymbol{\theta_i} - \boldsymbol{\theta})\right) \frac{\sqrt{|\Sigma_d|}}{\sqrt{|\Sigma_d + S|}}.\end{aligned} \tag{16}$$

Note that the subscript i now is an index, indicating a relation to the i^{th} element of $\boldsymbol{V}$ and its corresponding state $\boldsymbol{x_i}$ and controller parameters $\boldsymbol{\theta_i}$. Also note that, for reasons of notation, the conditioning on $M\boldsymbol{V} = \bar{\boldsymbol{r}}$ has been omitted.

The function $\bar{V}(\boldsymbol{\theta})$ is called the approximated controller quality function. This quality function can subsequently be plotted with respect to the controller parameters $\boldsymbol{\theta}$. Such a plot can then be used to gain more understanding of how the value of the system varies with respect to the controller parameters. Additionally, it is possible to use an optimization method to maximize this function, automatically tuning the controller parameters.

3. THE GP CONTROLLER TUNING ALGORITHM

The ideas and theories of the previous section can be assembled into a Gaussian process controller tuning algorithm. To automatically tune the controller parameters, the following steps need to be taken.

- Generate n measurements. Record the initial state $\boldsymbol{x^i}$, the final state $\boldsymbol{x^f}$, the weighted average reward $\bar{r}$, the applied controller parameters $\boldsymbol{\theta}$ and the experiment duration T.
- Assemble M and $\bar{\boldsymbol{r}}$ as shown in equation (10) and the matrix K as shown in equation (12).
- Optionally, tuning of hyperparameters like σ_f, Σ_d and Σ_t can be done through evidence maximization (also known as relevance determination) as described by Rasmussen and Williams [2006] (chapter 5).
- Derive the approximated controller quality function $\bar{V}(\boldsymbol{\theta})$ through (15).
- Optimize $\bar{V}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ to find the controller parameters.

The resulting algorithm has a relatively high computational complexity. This is because it requires inversion of the matrix $(MKM^T + \sigma_n^2 I)$, which is of size $n \times n$. The run-time of the algorithm hence is $\mathcal{O}(n^3)$. This makes the algorithm unsuitable for systems in which a lot of data is available. In such cases conventional system identification algorithms can be applied. The algorithm presented here is mainly suitable for nonlinear systems for which a limited data set is available.

4. APPLICATION IN AN EXPERIMENT

To test the performance of the algorithm, we apply it to a well-known problem: stabilizing an inverted pendulum.

4.1 The problem set-up

The equation of motion of an inverted pendulum is given by

$$mr^2\ddot{\alpha} + c\dot{\alpha} - mg\sin(\alpha) = u, \tag{17}$$

where m is the mass (1 kg), r is the radius (1 m), α is the angle (in radians) with respect to the vertical, c is the friction coefficient (0.5 kg m/s), g is the graviational constant (10 m/s^2) and u is the input to the system (in Nm).

The problem to be solved is the maximization of the value function of equation (2). The parameter γ is set to 1/2, and as reward function we will use

$$r(\boldsymbol{x}, \boldsymbol{u}) = -\frac{1}{2}\left(\left(\frac{\alpha}{\bar{\alpha}}\right)^2 + \left(\frac{\dot{\alpha}}{\bar{\dot{\alpha}}}\right)^2 + \frac{1}{10}\left(\frac{u}{\bar{u}}\right)^2\right). \tag{18}$$

In this equation, $\bar{\alpha}$, $\bar{\dot{\alpha}}$ and $\bar{u}$ are normalizing constants. We define $\bar{\alpha}$ as 45 degrees. We then define $\bar{\dot{\alpha}} = 140$ deg/s, which is (approximately) the angular velocity that the pendulum gets at $\alpha = 45°$ when released from a stationary position at $\alpha \approx 0°$. Finally, we define $\bar{u} = 5\sqrt{2}$ Nm, which is the input torque required to keep the pendulum stationary at $\alpha = 45°$.

In the reward function above, the first term is present to make the system move the pendulum to the upright position, making α zero. The second term is to prevent the system from making excessively violent movements, which are likely to result in an overshoot. The third term is present to prevent the system from applying excessively large inputs.

For this problem, we assume that (due to external reasons) we are restricted to a PD controller architecture $u = -K_p\alpha - K_d\dot{\alpha}$. If the proportional gain K_p is high enough to overcome the gravitational force, and if the derivative gain K_d is positive (adding some damping), the inverted pendulum is already stabilized. But the question remains which controller gains maximize the value function. Very aggressive controllers give high velocities $\dot{\alpha}$ and inputs u, while very cautious controllers are slow at reducing α.

4.2 Application of the controller tuning algorithm

To solve the problem, the algorithm of section 3 will be applied.

First of all data is generated. For the given system, a total of $n = 200$ trial time steps have been run. For every trial time step, the system was given a random initialization

$$\begin{bmatrix}\alpha_0\\ \dot{\alpha}_0\end{bmatrix} \sim \mathcal{N}(\boldsymbol{\mu_x}, S) = \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}(\bar{\alpha}/2)^2 & 0\\ 0 & (\bar{\dot{\alpha}}/2)^2\end{bmatrix}\right). \quad (19)$$

Next to this, the design choice was made to let $\log(K_p)$ and $\log(K_d)$ be uniformly distributed, with $\log(K_p) \in [5/2, 9/2]$ and $\log(K_d) \in [3/2, 7/2]$. Both K_p and K_d were kept constant during a trial time step.

The duration of each trial time step was 0.1 seconds. For each time step, the initial state $\boldsymbol{x^i}$ and the final state $\boldsymbol{x^f}$ was recorded, as well as an estimate $\bar{r}$ of the average reward received during the time step. Also the controller parameters $\boldsymbol{\theta} = [K_p \ K_d]^T$ were noted. Based on this, the matrices M and K and the vector $\bar{\boldsymbol{r}}$ were set up.

The covariance function used was given in equation (13). For this covariance function, we have applied evidence maximization to tune the hyperparameters, using a gradient descent method. Of course the exact results varied per algorithm execution, because random inputs have been used. However, the general trends were the same per test run. What we will present here is a single test run that can be seen as an 'average' run. The resulting matrices Σ_d and Σ_t were

$$\Sigma_d = \begin{bmatrix}0.68 & 0\\ 0 & 35.9\end{bmatrix} \quad \text{and} \quad \Sigma_t = \begin{bmatrix}3.8 & 0\\ 0 & 5.0\end{bmatrix}. \quad (20)$$

After tuning, σ_f^2 became 0.22 and σ_n^2 became $2.0 \cdot 10^{-6}$. Furthermore, it has been assumed that the prior mean $m(\boldsymbol{x})$ of the value function was a constant value μ. This constant μ has also been tuned and was set to -0.86.

It is interesting to note that, should either K_p or K_d have no influence on the value function, then the corresponding length parameter in Σ_t would increase to infinity during evidence maximization. Because of this, it is possible to determine which controller parameters are useful and which ones have less effect. In this case all elements of Σ_t converged to a finite value, so all controller parameters have a significant effect on the value.

Next, the approximated controller quality function $\bar{V}(\theta)$ was derived through relation (15). When doing this, we assumed that our prior distribution of $\boldsymbol{x}$ was $\mathcal{N}(\boldsymbol{\mu_x}, S)$, with $\boldsymbol{\mu_x}$ and S as defined in equation (19). The resulting approximation is shown in figure 1.

Some interesting things can be derived from figure 1. First of all, the optimal set of parameters within the ascribed interval can be obtained. For this test run, $\bar{V}(\boldsymbol{\theta})$ was at a maximum for $\log(K_p) = 3.61$ and $\log(K_d) = 2.57$. For these two controller parameters the approximated value was $V(\theta) = -0.0488$. Other experiment runs gave log-gain values that were on average 0.2 higher or lower, but the estimated value stayed close to -0.0488.

Secondly, figure 1 also shows how well the controller performs for other parameter combinations. If K_p and K_d are both small, or are both big, then the system will have a reasonable performance. However, if K_p is large and K_d is small, then the system gets oscillations, resulting in a low value. Similarly, if K_p is small and K_d is large, then the system behaves sluggishly, again giving a low value. This is confirmed by figure 2, which shows simulation runs for various values of K_p and K_d.

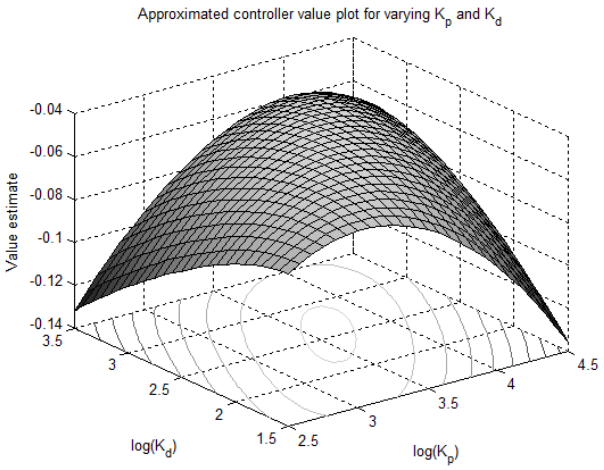

Fig. 1. A plot of the approximated value with respect to the controller gains K_p and K_d. The plot was made based on 200 trial time steps.

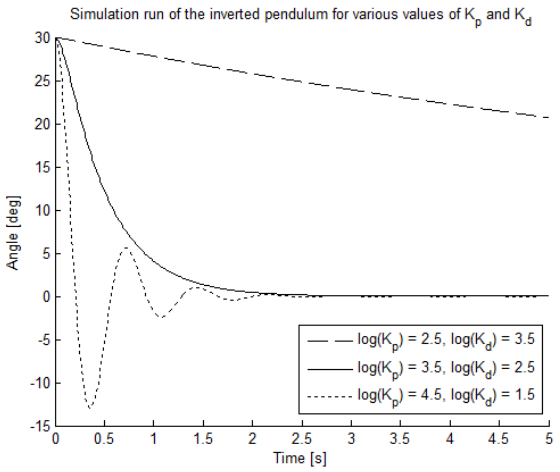

Fig. 2. Simulation runs of the inverted pendulum for various values of K_p and K_d. The system was put in an initial state of $\alpha_0 = 30$ deg and $\dot{\alpha}_0 = 0$ deg/s.

4.3 Comparison with analytic results

To find out more about the accuracy of the algorithm, it is useful to compare it with analytic results. This is complicated, because the system that we examined was nonlinear. However, it can be linearized. (In equation (17) replace $\sin(\alpha)$ by α.) For this linear system, it is possible to derive the value function $V(\boldsymbol{x}, \boldsymbol{\theta})$ analytically. Also the expected value

$$E[V(\boldsymbol{x}, \boldsymbol{\theta}) | \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, S)] \quad (21)$$

can be derived analytically. Numerically optimizing this with respect to $\boldsymbol{\theta}$ gave us the true optimal parameters for the linear system.

The results were very similar to what our approximation predicted. The above quantity was at a maximum when $\log(K_p) = 3.58$ and $\log(K_d) = 2.46$. The corresponding value that was obtained was $V(\theta) = -0.0489$. So, while the exact gains giving the maximum value are slightly different from what we got from our approximation, this did not significantly affect the system value.

But what causes the difference between the approximation given by our algorithm and the analytical results for the linear system? To find that out, the parameter tuning algorithm can also be applied to the linear system. This has been set up, and the results were not significantly different. This shows that the deviation was mainly caused by inaccuracies in the approximation, and was not due to linearizing the system.

5. CONCLUSIONS AND DISCUSSION

In this paper an algorithm has been developed that can tune the parameters of a fixed-structure controller, when applied to a system. It can be used for any system type and no model of the system is required. Only a set of system experiment data is needed with varying states and varying controller parameters. When applied to a test system, the algorithm proved to be functional and to give accurate results, even when little system data was available.

The algorithm is computationally quite intensive. The run-time is $\mathcal{O}(n^3)$, with n the number of trial time steps. It is therefore not suitable for systems with lots of measurement data. Instead, the algorithm was designed for systems for which a limited amount of measurement data is available. The algorithm uses this measurement data quite efficiently. In a test case in which two controller gains needed to be tuned, the gains were put close to the optimum using only 200 random trial time steps.

Other than the run-time, the main downside of the algorithm is that it requires complete knowledge of the state $\boldsymbol{x}$. This data is required by the covariance function k when setting up the covariance matrix K. Possibly, the system may also have a reasonable performance when, instead of the state $\boldsymbol{x}$, some output vector $\boldsymbol{y}$ is passed to the covariance function k. In this case the controller C would of course also reduce from a state controller to an output controller. How well such a set-up would work is a subject for future research.

Another very interesting topic for future research would be to enable the algorithm to apply online learning. Currently, the algorithm takes a set of measurement data and uses it to tune control parameters offline. When this can be done online, during the operation of the system, then an extra set of applications opens up. The algorithm may then for instance be applied to tune controllers of time-varying or parameter-varying systems.

Appendix A. OUTLINE OF A PROOF OF EQUATION (11)

In this appendix, an outline for the proof of equation (11) will be given. In fact, we will show that it follows from equation (8).

The proof is based on a coordinate transformation of $\boldsymbol{V}$. We define

$$\begin{bmatrix} \boldsymbol{V_a'} \\ \boldsymbol{V_b'} \end{bmatrix} = \boldsymbol{V'} = T\boldsymbol{V} = \begin{bmatrix} M \\ N \end{bmatrix} \boldsymbol{V}, \tag{A.1}$$

with the square matrix T defined as shown above and with N defined such that its rows span the null space of M. In other words, N is defined such that T is invertible but also $MN^T = 0$. By applying $\boldsymbol{V} = T^{-1}\boldsymbol{V'}$, it can be shown that the relation $M\boldsymbol{V} = \bar{\boldsymbol{r}}$ reduces to $\boldsymbol{V_a'} = \bar{\boldsymbol{r}}$. That is, $\boldsymbol{V_a'}$ is known (given), just like in equation (8) the term $\boldsymbol{y}$ was given. From this, the subsequent distribution of $\boldsymbol{V_b'}$ can be derived. Next, by transforming the result back, equation (11) is obtained.

It is interesting to note that equation (11) is actually a generalization of equation (8). (That is, apart from the assumption that $m(\boldsymbol{x}, \boldsymbol{\theta}) = 0$.) If $M = [I \quad 0]$, then equation (11) reduces back to equation (8).

ACKNOWLEDGEMENTS

This research is supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO), and which is partly funded by the Ministry of Economic Affairs.

REFERENCES

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

Joaquin Q. Candela, Agathe Girard, Jan Larsen, and Carl E. Rasmussen. Propagation of uncertainty in bayesian kernel models - application to multiple-step ahead forecasting. In *Advances in Neural Information Processing Systems*, pages 701–704. MIT Press, 2003.

March P. Deisenroth and Carl E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning*, pages 465–472. ACM Press, 2011.

Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd International Converfence on Machine Learning*, pages 201–208. ACM Press, 2005.

Agathe Girard, Carl E. Rasmussen, Joaquin Q. Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems*, pages 529–536. MIT Press, 2003.

Carl E. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems 16*, pages 751–759. MIT Press, 2004.

Carl E. Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Dearden Richard, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *In AAAI/IAAI*, pages 761–768. AAAI Press, 1998.

Sigurd Skogestad and Ian Postlethwaite. *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, 2005.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Michel Verhaegen and Vincent Verdult. *Filtering and System Identification: A Least Squares Approach*. Cambridge University Press, 2007.

Draguna Vrabie, Kyriakos G. Vamvoudakis, and Frank L. Lewis. *Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles*. The Institution of Engineering and Technology, 2013.

Christopher J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.