Kernel Recursive Least-Squares Tracker for Time-Varying Regression

Steven Van Vaerenbergh, *Member, IEEE*, Miguel Lázaro-Gredilla, *Member, IEEE*, and Ignacio Santamaría, *Senior Member, IEEE*

***Abstract*— In this paper, we introduce a kernel recursive least-squares (KRLS) algorithm that is able to track nonlinear, time-varying relationships in data. To this purpose, we first derive the standard KRLS equations from a Bayesian perspective (including a sensible approach to pruning) and then take advantage of this framework to incorporate forgetting in a consistent way, thus enabling the algorithm to perform tracking in nonstationary scenarios. The resulting method is the first kernel adaptive filtering algorithm that includes a forgetting factor in a principled and numerically stable manner. In addition to its tracking ability, it has a number of appealing properties. It is online, requires a fixed amount of memory and computation per time step, incorporates regularization in a natural manner and provides confidence intervals along with each prediction. We include experimental results that support the theory as well as illustrate the efficiency of the proposed algorithm.**

***Index Terms*— Adaptive filtering, Bayesian inference, Gaussian processes, kernel methods, kernel recursive least-squares (KRLS).**

I. Introduction

KERNEL methods offer an attractive framework to deal with many nonlinear problems in pattern analysis and nonlinear signal processing [1], [2]. These techniques are based on a nonlinear transformation of the data into a high-dimensional reproducing kernel Hilbert space, which allows to solve nonlinear learning problems in the input space as convex optimization problems in the transformed space. By relying on the "kernel trick," efficient algorithms with algebraically simple expressions are obtained, such as support vector machines or kernel principal component analysis [1]. However, in their batch form these algorithms have memory and computational complexity requirements that usually scale cubically with the number of data, rendering the processing of large data sets prohibitive if no additional measures are taken [3].

Manuscript received May 1, 2011; revised May 4, 2012; accepted May 10, 2012. Date of publication June 28, 2012; date of current version July 16, 2012. This work was supported in part by the MICINN Spanish Ministry for Science and Innovation under Grant TEC2010-19545-C04-03 COSIMA and the CONSOLIDER-INGENIO 2010 under Grant CSD2008-00010 COMONSENS.

S. Van Vaerenbergh and I. Santamaría are with the Department of Communications Engineering, University of Cantabria, Santander 39005, Spain (e-mail: steven@gtas.dicom.unican.es; nacho@gtas.dicom.unican.es).

M. Lázaro-Gredilla is with the Department of Communications Engineering, University of Cantabria, Santander 39005, Spain, and also with the Department of Signal Processing and Communications, Universidad Carlos III de Madrid, Leganés 28911, Spain (e-mail: miguel@tsc.uc3m.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2012.2200500

During the last decade, a significant number of kernel-based adaptive filtering techniques have been proposed (see [4]–[6]). These algorithms are online kernel methods that aim to recover a signal of interest by adapting their parameters as new data become available, typically by minimizing a least-squares cost function. The major challenge in designing these algorithms is to deal with their memory and computational complexities. In general, online algorithms require to update their solution as new data become available, and the complexity of each update should be limited [4]. Unfortunately, the functional representation of classical kernel-based algorithms grows linearly with the number of processed data. If an online strategy is adopted naïvely, this representation leads to growing complexities for each consecutive update. To make online kernel algorithms feasible, growth is typically slowed down by approximately representing the solution using only a subset of bases that are considered relevant according to a chosen criterion [5], [7]. In [5], a successful example of this strategy was applied to the kernel recursive least-squares (KRLS) algorithm, which is a kernelized version of the celebrated recursive least-squares (RLS) algorithm [8]. In this particular case, the support is reduced to a sparse "dictionary" of bases and a new basis is only added if it cannot be represented by a combination of other bases that are already present in the dictionary.

Here, we focus on an aspect of kernel adaptive filters that has not been satisfactorily addressed yet in the literature. In particular, kernel adaptive filters are typically designed for stationary environments and they are not capable of dealing with data whose input–output relationship changes over time. The ability to adapt to such changes is required to build a tracking algorithm, and it is fundamental in many filtering applications. Although most linear adaptive filters allow for tracking directly or through some straightforward extension [8], [9], this is not possible in kernel-based algorithms. As a result, most kernel adaptive filters are designed specifically for stationary environments only, on which they converge approximately to the batch filtering solution [5], [6].

To perform tracking, the algorithm is required to possess some mechanism that computes the solution giving more weight to more recent data. A common approach in adaptive filtering is to exponentially weight older data by scaling them with a forgetting factor. In [6] an exponentially weighted KRLS (EW-KRLS) algorithm was proposed that includes a forgetting factor. However, it faces numerical problems that do not allow it to operate on long data sequences. We will comment more on this below. From another

perspective, a few attempts have been made to include a state-space model or process equation within the KRLS framework [6], [10]. Nevertheless, these extended KRLS algorithms consider only very specific models of the state transition matrix or the state noise. A more radical approach to giving more weight to recent data is found in sliding-window algorithms, whose solution depends only on a fixed number of the latest observed data [4], [11], while any older data is discarded. Though this procedure leads to satisfactory tracking results in case all relevant data are present in the sliding window, the obtained solution degrades quickly when important data is discarded.

In this paper, we explore a more sensible approach to tracking. We specifically handle the uncertainty about the inferred input–output relationship, which we consider a latent function, and we study the problem of how older data should be forgotten. Our focus is on the KRLS algorithm, which represents a reasonable compromise between fast convergence and complexity. First, we define a probabilistic framework based on Gaussian processes (GPs) that offers an intuitive view of KRLS and allows to deal with uncertainty and network regularization in a natural manner.[1] Then, we propose a general strategy to forget past information based on blending the informative posterior with a noninformative distribution. The proposed framework allows us to design several different forgetting techniques, out of which we discuss two relevant formulations. The first technique forgets by shifting the probability distribution over the input–output relationship toward the prior belief, converging to it once all data are forgotten, which is consistent with the Bayesian framework. The second technique forgets by gradually injecting uncertainty into older data, which, interestingly, reduces exactly to the exponentially weighted RLS algorithm when a linear kernel is used.

Based on these ideas we propose a concrete algorithm, the KRLS Tracker (KRLS-T), that is capable of tracking nonstationary data that exhibit nonlinear relationships. To guarantee online operation, we limit its complexity in each step to $\mathcal{O}(M^2)$, where M is the number of bases allowed in memory. The presented method is closely related to the paper of Csató and Opper [12], in which a GP perspective is adopted. However, we believe its connection with KRLS has not received enough attention from the signal processing community, which is one of the issues we aim to address. We also extend [12] by adding the concept of forgetting, which yields tracking capabilities. Further novelties of this paper include that it rigorously introduces the concept of regularization into KRLS, relating it directly to the amount of noise in the data, and that the proposed algorithm provides uncertainty estimates of its predictions, which allows to establish confidence intervals. Some preliminary results were presented in [13].

The rest of this paper is structured as follows. In Section II we introduce a Bayesian framework that allows to derive previous results for KRLS in a principled manner and offers additional insight through the introduction of uncertainty. This property is exploited in Section III to design a KRLS algorithm capable of forgetting past information. In Section IV, we provide additional details about the predictive variances, which are obtained by using the presented Bayesian framework. We illustrate the performance of the proposed algorithm with a set of numerical examples in Section V. Finally, Section VI summarizes the main conclusions of this paper.

[1]Note that the equivalence between GPs and KRLS has been mentioned before in the literature [5], [7].

II. Bayesian Perspective of KRLS

First, we provide a Bayesian derivation of previous results for KRLS, obtained within the probabilistic framework of GPs. This unifying interpretation offers a simple and intuitive view of KRLS, adds specific handling of uncertainty and will prove specially useful to handle nonstationary scenarios.

A. Standard KRLS (With Evergrowing Dictionary)

Assume a set of ordered input–output pairs $\mathcal{D}_t \equiv \{\mathbf{x}_i, y_i\}_{i=1}^t$, where $\mathbf{x}_i \in \mathbb{R}^D$ are D-dimensional input vectors and $y_i \in \mathbb{R}$ are scalar outputs. Data pairs are made available on a one-at-a-time basis, that is, $(\mathbf{x}_t, y_t)$ is made available at time t. Our objective is to infer the predictive distribution of a new, unseen output y_{t+1} given the corresponding input $\mathbf{x}_{t+1}$ and data available up to time t, $\mathcal{D}_t$.

1) Bayesian Model: In a Bayesian setting, we need a model that describes the observations, and priors on the parameters of such model. Following the standard setup of GP regression, we can describe observations as the sum of an unobservable *latent function* of the inputs plus some unknown, zero-mean Gaussian noise

$$y_i = f(\mathbf{x}_i) + \varepsilon_i. \tag{1}$$

Equation (1) implies that the likelihood of the latent function is $p(y_i|f_i) = \mathcal{N}(y_i|f_i, \sigma_n^2)$, where we have used the shorthand notation $f_i = f(\mathbf{x}_i)$ and assumed noise power to be constant and equal to σ_n^2. To perform Bayesian inference, we also need a prior over the latent function, which is taken to be a zero-mean GP with covariance function $k(\mathbf{x}, \mathbf{x}')$, also known as *kernel*. A GP prior implies that the prior joint distribution of vector $\mathbf{f}_t = [f_1, \ldots, f_t]^\top$ is a zero-mean multivariate Gaussian with covariance matrix $\mathbf{K}_t$, with elements $[\mathbf{K}_t]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. In line with the previous literature on KRLS, we will refer to $k(\mathbf{x}, \mathbf{x}')$ as the *kernel function*, and to $\mathbf{K}_t$ as the *kernel matrix* (which in this example would correspond to inputs $\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$). For the sake of clarity, in the following we will omit conditioning on the inputs $\{\mathbf{x}_i\}_{i=1}^t$ or the parameters $\boldsymbol{\theta}$ that parameterize the kernel function.

2) Bayesian Recursive Update: The adopted setup corresponds to standard GP regression, which is thoroughly described in [14]. However, instead of the batch case in which all observations are available beforehand, we are interested in the online case, in which a new observation is incorporated at each time instant. Therefore, a new posterior $p(\mathbf{f}_t|\mathcal{D}_t)$ including the most recent observation t must be computed at each time instant. Observe that the posterior is only computed at the locations at which data have been observed, and not for the complete latent function. This is because $p(\mathbf{f}_t|\mathcal{D}_t)$ implicitly defines a posterior for $p(f(\mathbf{x})|\mathcal{D}_t)$ as

$$p(f(\mathbf{x})|\mathcal{D}_t) = \int p(f(\mathbf{x})|\mathbf{f}_t)\, p(\mathbf{f}_t|\mathcal{D}_t) \mathrm{d}\mathbf{f}_t \tag{2}$$

and therefore, both posterior distributions hold the same information. Note that $p(f(\mathbf{x})|\mathbf{f}_t)$ is a GP whose expression can be derived from the prior and does not depend on data.

Instead of recomputing $p(\mathbf{f}_t|\mathcal{D}_t)$ from scratch at every time instant, we can obtain a low-cost recursive update as follows:

$$p(\mathbf{f}_{t+1}|\mathcal{D}_{t+1}) = p(\mathbf{f}_t, f_{t+1}|\mathcal{D}_t, y_{t+1}) \tag{3a}$$
$$= \frac{p(y_{t+1}|f_{t+1})p(\mathbf{f}_t, f_{t+1}|\mathcal{D}_t)}{p(y_{t+1}|\mathcal{D}_t)} \tag{3b}$$
$$= \frac{p(y_{t+1}|f_{t+1})p(f_{t+1}|\mathbf{f}_t)p(\mathbf{f}_t|\mathcal{D}_t)}{p(y_{t+1}|\mathcal{D}_t)}. \tag{3c}$$

Equation (3b) follows from (3a) by direct application of Bayes rule. Equation (3c) includes an expansion due to the outputs being conditionally independent given the latent function at the corresponding inputs.

Let us assume that the posterior at time t is a known Gaussian $p(\mathbf{f}_t|\mathcal{D}_t) = \mathcal{N}(\mathbf{f}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, and we want to update it to include a new observation $(\mathbf{x}_{t+1}, y_{t+1})$. All the involved quantities can be easily derived from the stated assumptions, using probability rules.

Due to its very definition, the likelihood of $f_{t+1} = f(\mathbf{x}_{t+1})$ given the new observation is

$$p(y_{t+1}|f_{t+1}) = \mathcal{N}(y_{t+1}|f_{t+1}, \sigma_n^2). \tag{4}$$

Introducing the new quantities $\mathbf{Q}_t = \mathbf{K}_t^{-1}$, $\mathbf{q}_{t+1} = \mathbf{Q}_t\mathbf{k}_{t+1}$, and $\gamma_{t+1}^2 = k_{t+1} - \mathbf{k}_{t+1}^\top\mathbf{Q}_t\mathbf{k}_{t+1}$, where $[\mathbf{k}_{t+1}]_i = k(\mathbf{x}_i, \mathbf{x}_{t+1})$ and $k_{t+1} = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1})$, we can express the density of the latent function at the new input given its value at previous inputs as

$$p(f_{t+1}|\mathbf{f}_t) = \mathcal{N}(f_{t+1}|\mathbf{q}_{t+1}^\top\mathbf{f}_t, \gamma_{t+1}^2). \tag{5}$$

This result follows directly from the known prior probability $p(f_{t+1}, \mathbf{f}_t)$ and conditioning on $\mathbf{f}_t$. The inverse of the kernel matrix, $\mathbf{Q}_t$, has been defined because we will be computing and storing it instead of $\mathbf{K}_t$, which will never be directly used.

The denominator of (3), which corresponds to the marginal likelihood, also known as evidence, provides the predictive distribution of a new observation y_{t+1} given past data, and can be computed as

$$p(y_{t+1}|\mathcal{D}_t) = \int p(y_{t+1}|f_{t+1})p(f_{t+1}|\mathbf{f}_t)p(\mathbf{f}_t|\mathcal{D}_t)\mathrm{d}\mathbf{f}_t\mathrm{d}f_{t+1}$$
$$= \mathcal{N}(y_{t+1}|\hat{y}_{t+1}, \hat{\sigma}_{yt+1}^2) \tag{6}$$

with the mean and the variance of this Gaussian being

$$\hat{y}_{t+1} = \mathbf{q}_{t+1}^\top\boldsymbol{\mu}_t \quad \text{and} \quad \hat{\sigma}_{yt+1}^2 = \sigma_n^2 + \hat{\sigma}_{ft+1}^2$$

respectively, where we have also introduced the predictive variance of the latent function at the new input

$$\hat{\sigma}_{ft+1}^2 = k_{t+1} + \mathbf{k}_{t+1}^\top(\mathbf{Q}_t\boldsymbol{\Sigma}_t\mathbf{Q}_t - \mathbf{Q}_t)\mathbf{k}_{t+1}$$
$$= \gamma_{t+1}^2 + \mathbf{q}_{t+1}^\top\mathbf{h}_{t+1}.$$

with $\mathbf{h}_{t+1} = \boldsymbol{\Sigma}_t\mathbf{q}_{t+1}$.

Thus, all involved distributions (4)–(6) appearing in (3) are univariate normal with simple expressions. Using those, (3) can be evaluated and the posterior distribution can be expressed as

$$p(\mathbf{f}_{t+1}|\mathcal{D}_{t+1}) = \mathcal{N}(\mathbf{f}_{t+1}|\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \tag{7a}$$
$$\boldsymbol{\mu}_{t+1} = \begin{bmatrix}\boldsymbol{\mu}_t \\ \hat{y}_{t+1}\end{bmatrix} + \frac{y_{t+1} - \hat{y}_{t+1}}{\hat{\sigma}_{yt+1}^2}\begin{bmatrix}\mathbf{h}_{t+1} \\ \hat{\sigma}_{ft+1}^2\end{bmatrix} \tag{7b}$$
$$\boldsymbol{\Sigma}_{t+1} = \begin{bmatrix}\boldsymbol{\Sigma}_t & \mathbf{h}_{t+1} \\ \mathbf{h}_{t+1}^\top & \hat{\sigma}_{ft+1}^2\end{bmatrix} - \frac{1}{\hat{\sigma}_{yt+1}^2}\begin{bmatrix}\mathbf{h}_{t+1} \\ \hat{\sigma}_{ft+1}^2\end{bmatrix}\begin{bmatrix}\mathbf{h}_{t+1} \\ \hat{\sigma}_{ft+1}^2\end{bmatrix}^\top. \tag{7c}$$

The inverse of the kernel matrix including the new input can also be easily updated using

$$\mathbf{K}_{t+1}^{-1} = \mathbf{Q}_{t+1} = \begin{bmatrix}\mathbf{Q}_t & \mathbf{0} \\ \mathbf{0}^\top & 0\end{bmatrix} + \frac{1}{\gamma_{t+1}^2}\begin{bmatrix}\mathbf{q}_{t+1} \\ -1\end{bmatrix}\begin{bmatrix}\mathbf{q}_{t+1} \\ -1\end{bmatrix}^\top. \tag{8}$$

This illustrates both how probabilistic predictions for new observations can be made (using (6), which does not require knowledge of y_{t+1}), and how these new observations can be included in the posterior once they are available. All computations for a given update can be made in $\mathcal{O}(t^2)$ time, as is obvious from the update formulas. Only $\boldsymbol{\mu}_{t+1}$, $\boldsymbol{\Sigma}_{t+1}$, and $\mathbf{Q}_{t+1}$ will be reused in the next iteration, and the remaining quantities will be computed on demand.

The recursion updates can be initialized by setting

$$\boldsymbol{\mu}_1 = \frac{y_1 k(\mathbf{x}_1, \mathbf{x}_1)}{\sigma_n^2 + k(\mathbf{x}_1, \mathbf{x}_1)} \tag{9a}$$
$$\boldsymbol{\Sigma}_1 = k(\mathbf{x}_1, \mathbf{x}_1) - \frac{k(\mathbf{x}_1, \mathbf{x}_1)^2}{\sigma_n^2 + k(\mathbf{x}_1, \mathbf{x}_1)} \tag{9b}$$
$$\mathbf{Q}_1 = \frac{1}{k(\mathbf{x}_1, \mathbf{x}_1)} \tag{9c}$$

which corresponds to inference according to the proposed model for a single data point.

Since the model is exactly that of GP regression and all provided formulas are exact, probabilistic predictions made at time t for observation $t+1$ are exactly the same as those obtained using a standard GP in the batch setting. Using the batch formulation from [14], we can equivalently express the predictive mean and variance from (6) as

$$\hat{y}_{t+1} = \mathbf{k}_{t+1}^\top(\mathbf{K}_t + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}_t \tag{10a}$$
$$\hat{\sigma}_{yt+1}^2 = \sigma_n^2 + k_{t+1} - \mathbf{k}_{t+1}^\top(\mathbf{K}_t + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_{t+1}. \tag{10b}$$

Direct application of (10a) and (10b) involves a higher, $\mathcal{O}(t^3)$ cost, so the recursive procedure of the previous section is preferred. However, these equations are useful to illustrate the form of the predictive distribution after several iterations, which is somewhat obscured in the recursive formulation.

In the standard KRLS setting, the predictive mean is often expressed as $\hat{y}_{t+1} = \mathbf{k}_{t+1}^\top\boldsymbol{\alpha}_t$, where $\boldsymbol{\alpha}_t$ are the kernel weights. When using the batch formulation, these weights can be obtained with $\boldsymbol{\alpha}_t = (\mathbf{K}_t + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}_t$, whereas in the recursive formulation the same result can be obtained at each step t by computing $\boldsymbol{\alpha}_t = \mathbf{K}_t^{-1}\boldsymbol{\mu}_t = \mathbf{Q}_t\boldsymbol{\mu}_t$. Observe the resemblance between the batch and recursive formulations. In the batch formulation, we are using *noisy* observations $\mathbf{y}_t$, so the kernel matrix includes a regularization term $\sigma_n^2\mathbf{I}$. In the recursive formulation we use the values of the *noiseless* function evaluated

at the inputs $\boldsymbol{\mu}_t$, so no noise term is added. Obviously, the same $\boldsymbol{\alpha}_t$ is obtained in both cases.

B. Fixed-Budget KRLS

As pointed out, the posterior over the latent function given a set of observations can be summarized by the posterior of the latent function at the observed inputs. The set of inputs at which the joint posterior is available is usually referred to as *set of bases* or *dictionary*. The recursive procedure described in the previous section grows the set of bases at each time step, thus increasing computation and memory requirements unboundedly. In this section, we describe how to limit resource usage to a predefined value.

The overall strategy is very simple. Once the amount of bases in the dictionary grows larger than a predefined budget M, remove one basis from the dictionary. To accomplish this, we need a procedure to remove a basis from the dictionary losing as little information as possible, and a criterion to help us decide which basis should be removed. Both are described in the following.

1) How to Optimally Remove a Basis: After the inclusion of several observations, we are left with a posterior of the form $p(\mathbf{f}_t, f_{t+1}|\mathcal{D}_{t+1}) = \mathcal{N}(\mathbf{f}_{t+1}|\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1})$. Without lack of generality we will assume that we want to remove the basis corresponding to f_{t+1}.

There are several equivalent ways of deriving basis removal equations. One option is to try to approximate the full posterior $p(\mathbf{f}_t, f_{t+1}|\mathcal{D}_{t+1})$ with the product of some distribution $q(\mathbf{f}_t)$ (in which the basis f_{t+1} has been removed) times $p(f_{t+1}|\mathbf{f}_t)$ (which does not depend on any data). The optimal form of $q(\mathbf{f}_t)$ is then derived by minimizing the Kullback–Leibler (KL) divergence between the original and the approximate posterior

$$\begin{aligned} q^*(\mathbf{f}_t) &= \underset{q(\mathbf{f}_t)}{\operatorname{argmin}}\ \mathrm{KL}(p(\mathbf{f}_t, f_{t+1}|\mathcal{D}_{t+1})||q(\mathbf{f}_t)p(f_{t+1}|\mathbf{f}_t)) \\ &= p(\mathbf{f}_t|\mathcal{D}_{t+1}). \end{aligned}$$

This gives the obvious result. The optimal way to remove a basis from the posterior within this Bayesian framework is simply to marginalize it out. The above KL divergence is zero when the removed basis is independent of data given the remaining bases $\mathbf{f}_t$, that is, $p(f_{t+1}|\mathbf{f}_t, \mathcal{D}_{t+1}) = p(f_{t+1}|\mathbf{f}_t)$, in which case no loss of information would occur. In the remaining cases, as much information about $\mathcal{D}_{t+1}$ as possible is retained in the posterior distribution.

From (5) we know that if $\gamma^2_{t+1} = 0$, there is a deterministic relationship between f_{t+1} and $\mathbf{f}_t$. In this case, the assumption $p(f_{t+1}|\mathbf{f}_t, \mathcal{D}_{t+1}) = p(f_{t+1}|\mathbf{f}_t)$ holds exactly, and the basis can be removed without any information loss. Consequently, whenever we are adding a new basis and γ^2_{t+1} is found to be (numerically) zero, we can remove it immediately thereafter. Since the basis set is left unchanged, we have that $\mathbf{Q}_{t+1} = \mathbf{Q}_t$. This is particularly useful, because the update for $\mathbf{Q}_t$ is not well-defined for $\gamma^2_{t+1} = 0$ (it would represent the inverse of a singular matrix). Note that we remove bases, not observations. Although observations and bases are added simultaneously, when we remove a basis we aim to keep the information about the observations. Thus, if we start with posterior $p(\mathbf{f}_t|\mathcal{D}_t)$, add an observation, and then remove the basis corresponding to that very observation, we are left with a different posterior, $p(\mathbf{f}_t|\mathcal{D}_{t+1})$.

Since marginalizing out a variable in a joint Gaussian distribution is achieved by simply removing the corresponding row and column from its mean vector and covariance matrix, the removal equations become

$$\boldsymbol{\mu}_{t+1} \leftarrow [\boldsymbol{\mu}_{t+1}]_{-i} \tag{11a}$$

$$\boldsymbol{\Sigma}_{t+1} \leftarrow [\boldsymbol{\Sigma}_{t+1}]_{-i,-i} \tag{11b}$$

where the notation $[\cdot]_{-i}$ refers to a vector in which the ith row has been removed, and $[\cdot]_{-i,-i}$ to matrix in which the ith row and column have been removed. Following this notation, we will use $[\cdot]_{-i,i}$ to refer to the ith column of a matrix, excluding the element in the ith row.

The ith basis can be removed from the inverse of the kernel matrix using

$$\mathbf{Q}_{t+1} \leftarrow [\mathbf{Q}_{t+1}]_{-i,-i} - \frac{[\mathbf{Q}_{t+1}]_{-i,i}[\mathbf{Q}_{t+1}]^\top_{-i,i}}{[\mathbf{Q}_{t+1}]_{i,i}}. \tag{12}$$

So far we have been using $\mathbf{f}_t = [f_1, \ldots, f_t]^\top$. However, as we start adding and pruning bases, $\mathbf{f}_t$ no longer maintains the mentioned structure. Instead, it becomes a vector of latent function values, corresponding to the bases that have been added but have not been pruned up till this point. Thus, $\mathbf{f}_t$ grows as described in Section II-A up to size M, and at this point, insertions and deletions are alternated, so that its size is fixed as M. The kernel matrix $\mathbf{K}_t$ and its inverse $\mathbf{Q}_t$ will now correspond to the bases in the set, which will not necessarily be all bases seen up to time t as they were before. Also, when adding a new basis, vector $\mathbf{k}_{t+1}$ will refer to the kernel function between the new data point at $t+1$ and the current basis set, that is, not all past bases but a subset of them.

2) Basis Removal Criterion: To keep our posterior within the required budget, we must remove the least relevant basis after the inclusion of a new basis in the set, when we exceed our budget M. Basis removal is straightforward, as we just saw, but selecting which of the $M+1$ bases should be removed is not. A desirable criterion would be to select the basis i that minimizes the KL-divergence between the exact and approximate posterior

$$\mathrm{KL}(p(\mathbf{f}_{t+1}|\mathcal{D}_{t+1})||p([\mathbf{f}_{t+1}]_i|[\mathbf{f}_{t+1}]_{-i})p([\mathbf{f}_{t+1}]_{-i}|\mathcal{D}_{t+1})).$$

That is, a measure of the information loss because of basis removal. This cost function, which was used in the previous section to determine *how* a basis should be removed, can now be evaluated for each basis i in the dictionary to determine *which* basis should be removed. It is possible to compute this KL-divergence analytically for every basis in the dictionary, but it is computationally too expensive. Instead, here we will consider minimizing the square of the error that the approximation induces in the mean of the posterior. In our experiments, this simpler cost function resulted equally effective in pruning the less relevant bases.

Without lack of generality, we first consider the case in which we remove f_{t+1}, the most recently added basis, from the posterior. As we discussed in the previous section, this implies

approximating the posterior distribution $p(\mathbf{f}_t, f_{t+1}|\mathcal{D}_{t+1})$ with $p(f_{t+1}|\mathbf{f}_t)p(\mathbf{f}_t|\mathcal{D}_{t+1})$. For the exact distribution, the mean of $\mathbf{f}_{t+1}$ is known to be $\boldsymbol{\mu}_{t+1}$, which can also be expressed as

$$\boldsymbol{\mu}_{t+1} = \mathbf{K}_{t+1}\mathbf{Q}_{t+1}\boldsymbol{\mu}_{t+1}.$$

For the approximate distribution, the mean of $\mathbf{f}_{t+1}$ is

$$\tilde{\boldsymbol{\mu}}_{t+1} = \mathbf{K}_{t+1}\begin{bmatrix}\mathbf{Q}_t & \mathbf{0}\\ \mathbf{0}^\top & 0\end{bmatrix}\boldsymbol{\mu}_{t+1}.$$

Subtracting both means and simplifying, we can compute the error that the approximation introduced in each element of the posterior mean

$$\boldsymbol{\mu}_{t+1} - \tilde{\boldsymbol{\mu}}_{t+1} = \begin{bmatrix}\mathbf{0}\\ [\mathbf{Q}_{t+1}\boldsymbol{\mu}_{t+1}]_{t+1}/[\mathbf{Q}]_{t+1,t+1}\end{bmatrix}.$$

Observe that only the mean of f_{t+1}, the removed basis, is distorted. The mean at the remaining bases is left unaffected. With this result, we can easily compute the squared error that would be introduced if we removed each of the bases, and thus flag for removal the basis that incurs in the least error, found as

$$\underset{i}{\operatorname{argmin}} \left(\frac{[\mathbf{Q}_{t+1}\boldsymbol{\mu}_{t+1}]_i}{[\mathbf{Q}]_{i,i}}\right)^2.$$

This is a well-known pruning criterion, used for instance in [15] and [16].

The described fixed-budget KRLS algorithm alternates the inclusion of bases and observations with the deletion of bases, using a simple basis selection and removal procedure. A complete iteration requires $\mathcal{O}(M^2)$ computation and $\mathcal{O}(2M^2 + M(D+1))$ storage, since we need to keep track of $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t, \mathbf{Q}_t$, and a list of the inputs in the basis set.

C. Remarks and Relation With Previous Works

Some interesting observations can be made on the above-introduced formulation. First, even if the kernel function does not include regularization, the Gaussian noise assumption in (1) automatically introduces a regularization term $\sigma_n^2\mathbf{I}$ in the batch formulation (10). Although not necessary, introducing (even a negligible amount of) noise is highly recommended since, as shown in the batch formulation, it enhances the conditioning of the kernel matrix and allows it to be safely inverted. This is less obvious in the recursive formulation, but equally true.

Though the connection of the recursive formulation (7) with standard KRLS might not be immediately apparent, it is clear from (10a) that if all observations are included, predictions are exactly the same as those made by standard approaches in literature, including [5], in which $\sigma_n^2 = 0$, and [6], which introduces a regularization term to penalize the least-squares solution directly. In addition to the predictive mean provided by standard KRLS, this probabilistic formulation also provides an uncertainty estimate, at no additional cost.

Some commonplace expressions in the standard KRLS definition also appear in our recursive formulation, with a clear probabilistic interpretation. For instance, $\mathbf{q}_{t+1} = \mathbf{K}_t^{-1}\mathbf{k}_{t+1}$ projects the values of $f(\mathbf{x})$ at inputs $1, \ldots, t$ onto $f(\mathbf{x}_{t+1})$ —it is the optimal linear predictor in the mean-squared error (MSE) sense— whereas γ_{t+1}^2 expresses the uncertainty of the resulting projection. To be completely clear, if $\gamma_{t+1}^2 = 0$, then f_{t+1} can be exactly determined from the knowledge of $\mathbf{f}_t$. As we will see in the next section, being aware of exact or approximate redundancies such as this one can lead to reductions in the complexity of the algorithm.

The probabilistic interpretation we provided is also much clearer than standard KRLS regarding information flow. We start with a GP prior and no observations. After each time step, observations are blended with the prior producing a posterior that will serve as prior for the next iteration. All the information available up to the current time step is reflected in a single distribution over a finite set of variables: $p(\mathbf{f}_t|\mathcal{D}_t)$. This distribution represents our current belief about the latent function and can be expanded to a distribution over functions using (2), that is, it can be seen as a sparse representation of the posterior GP. This representation makes operations such as pruning or forgetting data remarkably simple, as we shall see. A closely related algorithm was presented in the context of GPs in [12], using a different parameterization of the posterior GP. Specifically, $\boldsymbol{\alpha}_t = \mathbf{Q}_t\boldsymbol{\mu}_t$ and $\mathbf{C}_t = \mathbf{Q}_t\boldsymbol{\Sigma}_t\mathbf{Q}_t - \mathbf{Q}_t$ are used, which renders basis removal slightly less obvious.

Finally, network pruning has been discussed in the literature on many occasions. It was first used in the context of neural networks [17] to trim large networks, and it was only recently adopted in online methods, notably in the perception [18] and radial basis function (RBF) networks [19]. Pruning was mentioned in least-squares techniques in the context of least-squares support vector machines [16], [20]. And recently, a fixed-budget KRLS was introduced in [21], although that approach does not preserve information of discarded bases.

III. KRLS Tracker

In the previous section, we have provided a Bayesian perspective of fixed-budget KRLS. The algorithm learns by including new observations in the posterior distribution. Some information is lost whenever a basis is pruned, but we take every measure to keep this information loss to a minimum. In a time-varying scenario, however, only recent samples have relevant information, whereas the information contained in older samples is actually misleading. In such a case, we would be interested in having a KRLS *tracker* that is able to forget past information and track changes in the target latent function.

The marginal distribution of a GP over a finite set of points is a joint multivariate Gaussian distribution. Since up to this point we only cared about distributions over a finite set of points (those in the dictionary), we only needed to work with Gaussian distributions. In this section, we are interested in developing forgetting strategies and assessing their effect throughout the whole input space, so we will work with complete GPs. We briefly remind the reader of the GP notation: GPs are stochastic processes that are defined through a mean function $m(\mathbf{x})$ and a covariance function $c(\mathbf{x}, \mathbf{x}')$. To denote that $f(\mathbf{x})$ is a stochastic function drawn from a GP, we will use the notation $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), c(\mathbf{x}, \mathbf{x}'))$. Loosely speaking, one can think of a GP as a Gaussian distribution over an infinite set of points (evaluating $f(\mathbf{x})$ at every possible $\mathbf{x}$ and

thus building an infinitely long vector), with a corresponding infinitely long mean (given by the evaluation of $m(\mathbf{x})$ at every possible point) and an infinitely big covariance matrix (given by evaluating $c(\mathbf{x}, \mathbf{x}')$ at each possible pair of points). A complete background on GPs can be found in [14].

After several inclusion–deletion steps, all information available up to time t has (approximately) been stored in the posterior density over the dictionary bases $\mathbf{f}_t|\mathcal{D}_t \sim \mathcal{N}(\mathbf{f}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. Inserting this $p(\mathbf{f}_t|\mathcal{D}_t)$ in (2) and solving the integral, we obtain the implied posterior GP over the whole input space

$$f(\mathbf{x})|\mathcal{D}_t \sim \mathcal{GP}(\mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \boldsymbol{\mu}_t,\ k(\mathbf{x}, \mathbf{x}') + k_t(x)^\top \mathbf{Q}_t(\boldsymbol{\Sigma}_t - \mathbf{K}_t)\mathbf{Q}_t \mathbf{k}_t(\mathbf{x}'))$$

where $\mathbf{k}_t(\mathbf{x})$ is the vector of covariances between $\mathbf{x}$ and all the bases in the dictionary at time t. Observe that this equation has the same form as the prediction (6), but it extends to the whole input space.

To make KRLS able to adapt to nonstationary environments, we should make it able to "forget" past samples, that is, to intentionally force the posterior $p(f(\mathbf{x})|\mathcal{D}_t)$ to lose some information. A very general approach to this is to linearly combine $f(\mathbf{x})|\mathcal{D}_t$ with another independent GP, $n(\mathbf{x})$, that holds no information about data. Since this new posterior after forgetting will be a linear combination of two GPs, it will also be a GP, and we will denote it as $\breve{f}(\mathbf{x})|\mathcal{D}_t$. We denote the linear combination as

$$\breve{f}(\mathbf{x})|\mathcal{D}_t = \alpha f(\mathbf{x})|\mathcal{D}_t + \beta n(\mathbf{x}) \tag{13}$$

where $\alpha, \beta > 0$ are used to control the trade-off between the informative GP $f(\mathbf{x})|\mathcal{D}_t$ and the uninformative, "forgetting noise" $n(\mathbf{x})$.

The posterior GP after forgetting, $p(\breve{f}(\mathbf{x})|\mathcal{D}_t)$, must also be expressible in terms of a distribution over the latent points in the dictionary. We will refer to this distribution as $\mathcal{N}(\breve{\boldsymbol{\mu}}_t, \breve{\boldsymbol{\Sigma}}_t)$. Using (2) again, we can express the posterior after forgetting in terms of $\breve{\boldsymbol{\mu}}_t$ and $\breve{\boldsymbol{\Sigma}}_t$

$$\breve{f}(\mathbf{x})|\mathcal{D}_t \sim \mathcal{GP}(\mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \breve{\boldsymbol{\mu}}_t,\ k(\mathbf{x}, \mathbf{x}') + k_t(x)^\top \mathbf{Q}_t(\breve{\boldsymbol{\Sigma}}_t - \mathbf{K}_t)\mathbf{Q}_t \mathbf{k}_t(\mathbf{x}')). \tag{14}$$

Different definitions for α, β, and $n(\mathbf{x})$ will result in different types of forgetting. In the following, we introduce two relevant forgetting strategies.

A. Back-to-the-Prior (B2P) Forgetting

First, we select the form of the GP $n(\mathbf{x})$ that acts as noise. This GP holds no information about the data and it is independent of $f(\mathbf{x})|\mathcal{D}_t$. Assume for a moment that we want to forget all past data. In this case, we must set $\alpha = 0$ to completely remove the informative GP. In that case, our posterior GP would be $\beta n(\mathbf{x})$. The distribution of the posterior when no data has been observed, should, by definition, be equal to the prior. Therefore $n(\mathbf{x})$ must be a scaled version of the GP prior. Without lack of generality, we can choose this scale to be 1, so that the noise GP becomes

$$n(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}')).$$

Obviously, with this choice, setting $\alpha = 0$ should imply $\beta = 1$. Observe that $n(\mathbf{x})$ corresponds to colored noise, using the same coloring as the prior.

Once $n(\mathbf{x})$ has been defined, the distribution of $\breve{f}(\mathbf{x})|\mathcal{D}_t$ can be obtained from its definition (13). Since both GPs are independent, their linear combination is distributed as

$$\breve{f}(\mathbf{x})|\mathcal{D}_t \sim \mathcal{GP}(\alpha \mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \boldsymbol{\mu}_t,\ (\alpha^2 + \beta^2)k(\mathbf{x}, \mathbf{x}') + k_t(x)^\top \mathbf{Q}_t(\alpha^2 \boldsymbol{\Sigma}_t - \alpha^2 \mathbf{K}_t)\mathbf{Q}_t \mathbf{k}_t(\mathbf{x}')). \tag{15}$$

Comparing (14) and (15) and identifying terms, we obtain

$$\breve{\boldsymbol{\mu}}_t = \alpha \boldsymbol{\mu}_t; \qquad \breve{\boldsymbol{\Sigma}}_t = \alpha^2 \boldsymbol{\Sigma}_t + (1 - \alpha^2)\mathbf{K}_t; \qquad \alpha^2 + \beta^2 = 1$$

which provides the relationship between the posterior distribution before and after the forgetting occurs. Forgetting depends on a single positive parameter, α, and one can find the corresponding $\beta = \sqrt{1 - \alpha^2}$. This latter equation implies that α cannot be bigger than 1. Its values are therefore in the range from 0 (all past data is forgotten and we arrive back at the prior) to 1 (no forgetting occurs and we are left with the original, unmodified posterior). Re-parameterizing $\alpha^2 = \lambda$ for convenience, the forgetting updates are finally

$$\boldsymbol{\Sigma}_t \leftarrow \lambda \boldsymbol{\Sigma}_t + (1 - \lambda)\mathbf{K}_t \tag{16a}$$

$$\boldsymbol{\mu}_t \leftarrow \sqrt{\lambda}\boldsymbol{\mu}_t \tag{16b}$$

where we denote $\lambda \in (0, 1]$ as the *forgetting factor*. The smaller the value of λ, the faster the algorithm can track changes (and the less it is able to learn, because information is quickly discarded). Usually, only values in the $[0.95, 1]$ range are sensible. We call this technique B2P forgetting. Interestingly, this type of forgetting reduces to extended RLS when using a linear kernel.

B. Uncertainty-Injection (UI) Forgetting

An alternative way to implement forgetting is to keep the posterior GP unscaled (i.e., set $\alpha = 1$), and increase its uncertainty by adding zero-mean noise. To increase the uncertainty without changing its structure, the noise process must have the same structure as the posterior covariance. Nevertheless, at the same time, we want our posterior GP to be expressible in terms of the posterior over the dictionary bases. We can approximately fulfill both desiderata by using

$$n(\mathbf{x}) \sim \mathcal{GP}(0, \mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \boldsymbol{\Sigma}_t \mathbf{Q}_t \mathbf{k}_t(\mathbf{x}')).$$

This noise covariance function matches the posterior covariance function and thus results in a re-scaling whenever $k(\mathbf{x}, \mathbf{x}') - \mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \mathbf{k}_t(\mathbf{x}') = 0$. When this condition is met, an exact re-scaling of the covariance occurs, with the amount of increase being determined by β. This occurs when the dictionary is a complete basis that spans the whole function space, for instance when using a kernel that admits a finite-dimensional expansion and the number of bases in the dictionary M is bigger than the number of dimensions. Note that even if the modification does not correspond to an exact scaling, the uncertainty still grows, as we are simply adding colored noise.

Using (13), we can compute the distribution of the posterior GP after forgetting for this type of noise

$$\breve{f}(\mathbf{x})|\mathcal{D}_t \sim \mathcal{GP}(\mathbf{k}_t(\mathbf{x})^\top \mathbf{Q}_t \boldsymbol{\mu}_t,\ k(\mathbf{x},\mathbf{x}') + k_t(x)^\top \mathbf{Q}_t((1+\beta^2)\boldsymbol{\Sigma}_t - \mathbf{K}_t)\mathbf{Q}_t \mathbf{k}_t(\mathbf{x}')). \quad (17)$$

Comparing (14) and (17) and identifying terms, we obtain

$$\breve{\boldsymbol{\mu}}_t = \boldsymbol{\mu}_t; \qquad \breve{\boldsymbol{\Sigma}}_t = (1+\beta^2)\boldsymbol{\Sigma}_t$$

which, again, provides the relationship between the posterior distribution before and after the forgetting occurs. Forgetting now depends on the single positive parameter β, whose values are in the range from 0 (no forgetting occurs and we are left with the original, unmodified posterior) to ∞ (uncertainty grows to infinity and all information is lost). Re-parameterizing $\beta^2 = (1-\lambda)/\lambda$ for notational convenience and consistency with the previous rule, the forgetting updates finally become

$$\boldsymbol{\Sigma}_t \leftarrow \frac{1}{\lambda}\boldsymbol{\Sigma}_t \quad (18a)$$

$$\boldsymbol{\mu}_t \leftarrow \boldsymbol{\mu}_t \quad (18b)$$

where $\lambda \in (0, 1]$ (to span the mentioned range in β). The forgetting factor λ has a similar meaning to the one in the previous section, with $\lambda = 1$ also corresponding to no forgetting and $\lambda = 0$ corresponding to complete forgetting. However, it is not correct to assume that the same value of λ corresponds to the same level of adaptivity in both forgetting strategies.

We denote this type of forgetting as UI forgetting, and it has the nice property of making KRLS-T reduce exactly to the exponentially weighted RLS when a linear kernel is used, as we will discuss in Section III-D. Unfortunately, it also de-emphasizes regularization over time (just as standard exponentially weighted RLS does, see [9]), and successive forgetting iterations may result in a posterior with higher uncertainty than the prior, which is not sensible from a Bayesian perspective. It may be argued that these are not desirable properties in a forgetting strategy, but we consider it nonetheless due to its theoretical relevance.

C. KRLS-T Algorithm

The KRLS-T algorithm is summarized in Algorithm 1. Note that it has $\mathcal{O}(M^2)$ memory and computational complexity. Furthermore, when adding a new basis to the dictionary, some caution is to be exercised. Including a redundant basis in the dictionary (besides wasting one slot of the dictionary) would make the dictionary matrix singular and its inverse $\mathbf{Q}_t$ undefined. To avoid this, we check for redundancy before adding each basis by using a threshold ϵ close to the machine precision. Observe that this threshold is *not* a criterion to reduce dictionary growth, but a criterion for floating point identity under finite precision. Much in the same spirit, the covariance function $k(\mathbf{x}_i, \mathbf{x}_j)$ can be redefined as $k(\mathbf{x}_i, \mathbf{x}_j) + \epsilon\delta_{ij}$ to enhance the conditioning of matrix $\mathbf{Q}_t$. This "jitter" noise is added to increase numerical stability and is fundamentally different from observation noise. Thus ϵ is not a parameter of the algorithm, but a machine-dependent parameter. A MATLAB implementation of KRLS-T including some experiments from this paper can be obtained at http://www.tsc.uc3m.es/~miguel.

Algorithm 1 Kernel Recursive Least-Squares Tracker

Parameters: Forgetting factor λ, regularization σ_n^2, kernel function $k(\mathbf{x}, \mathbf{x}')$, including its parameters $\boldsymbol{\theta}$, budget M.
Observe $(\mathbf{x}_1, y_1)$.
Initialize $\boldsymbol{\mu}_1$, $\boldsymbol{\Sigma}_1$, $\mathbf{Q}_1$ as per Eq. (9).
Add $\mathbf{x}_1$ to the dictionary.
for time instant $t = 1, 2, \ldots$ **do**
  Forget using B2P-forgetting (16) or UI-forgetting (18).
  Observe new input $\mathbf{x}_{t+1}$.
  Compute $\mathbf{k}_{t+1}$, the kernel between $\mathbf{x}_{t+1}$ and every basis in the dictionary.
  Compute $k_{t+1} = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1})$.
  Compute $\mathbf{q}_{t+1} = \mathbf{Q}_t \mathbf{k}_{t+1}$.
  Compute projection uncertainty $\gamma_{t+1}^2 = k_{t+1} - \mathbf{k}_{t+1}^\top \mathbf{q}_{t+1}$.
  Compute $\mathbf{h}_{t+1} = \boldsymbol{\Sigma}_t \mathbf{q}_{t+1}$.
  Compute noiseless pred. var. $\hat{\sigma}_{ft+1}^2 = \gamma_{t+1}^2 + \mathbf{q}_{t+1}^\top \mathbf{h}_{t+1}$.
  Output predictive mean $\hat{y}_{t+1} = \mathbf{q}_{t+1}^\top \boldsymbol{\mu}_t$.
  Output predictive variance $\hat{\sigma}_{yt+1}^2 = \sigma_n^2 + \hat{\sigma}_{ft+1}^2$.
  Observe actual output y_{t+1}.
  Compute $\boldsymbol{\mu}_{t+1}$, $\boldsymbol{\Sigma}_{t+1}$ as per Eq. (7).
  if $\gamma_{t+1}^2 < \epsilon$ (for some $\epsilon > 0$ close to machine precision) **then**
    Remove basis $t+1$ (introduces no error):
    $\boldsymbol{\mu}_{t+1} \leftarrow [\boldsymbol{\mu}_{t+1}]_{-(t+1)}$, $\boldsymbol{\Sigma}_{t+1} \leftarrow [\boldsymbol{\Sigma}_{t+1}]_{-(t+1),-(t+1)}$.
  else
    Compute $\mathbf{Q}_{t+1}$ as per Eq. (8).
    Add basis $\mathbf{x}_{t+1}$ to the dictionary.
    if Number of bases in the dictionary $> M$ **then**
      Compute squared errors for each candidate basis removal $([\mathbf{Q}_{t+1}\boldsymbol{\mu}_{t+1}]_i / [\mathbf{Q}_{t+1}]_{i,i})^2$.
      Remove basis i that introduces minimum error:
      $\boldsymbol{\mu}_{t+1} \leftarrow [\boldsymbol{\mu}_{t+1}]_{-i}$, $\boldsymbol{\Sigma}_{t+1} \leftarrow [\boldsymbol{\Sigma}_{t+1}]_{-i,-i}$.
      Remove basis i from $\mathbf{Q}_{t+1}$ as per Eq. (12).
      Remove basis $\mathbf{x}_i$ from the dictionary.
    end if
  end if
end for

D. Connection With Exponentially Weighted RLS

When a linear kernel $k(\mathbf{x}, \mathbf{x}') = \sigma_0^2 \mathbf{x}^\top \mathbf{x}'$ and uncertainty-injecting forgetting are used, the proposed KRLS-T algorithm is identical to the standard regularized exponentially weighted RLS [9]. It also incurs in the same computational cost, because after D bases have been inserted into the dictionary (where D is the dimensionality of $\mathbf{x}$), γ_t^2 will always be zero and no new bases will be added.

In the standard RLS definition, matrix $\mathbf{R}$ is the weighted correlation matrix and $\mathbf{r}$ is the cross-correlation between $\mathbf{x}$ and y. However, in the standard recursive implementation, these quantities are not directly used, and instead it is $\mathbf{P} = \mathbf{R}^{-1}$ and $\mathbf{w} = \mathbf{R}^{-1}\mathbf{r}$ that are used and iteratively updated. When the canonical basis is used by KRLS-T, that is, $\mathbf{K}_t = \mathbf{Q}_t = \mathbf{I}$, the

following correspondence exists:

$$\mathbf{w}_t = \boldsymbol{\mu}_t; \qquad \mathbf{P}_t = \lambda/\sigma_n^2 \boldsymbol{\Sigma}_t$$

where $\boldsymbol{\Sigma}_t$ is taken immediately after the forgetting step. Observe that the prior over the canonical basis is $\mathcal{N}(\mathbf{0}, \sigma_0^2\mathbf{I})$, by which the above correspondence yields the following starting point for the RLS algorithm: $\mathbf{w}_0 = \mathbf{0}$, $\mathbf{P}_0 = \lambda\sigma_0^2/\sigma_n^2\mathbf{I}$. All successive updates, as described in the KRLS-T algorithm, are then identical to those of standard RLS. If a noncanonical basis is used, a simple linear transformation is sufficient to express the equivalence of both algorithms.

Standard RLS de-emphasizes regularization over time (see [9]), so the predictive means of the algorithm become independent of σ_n^2 after enough iterations have passed (this effect is faster for smaller λ). Interestingly, a regularized version of RLS (known as extended RLS) can be simply obtained by using a linear kernel with the back-to-the-prior forgetting procedure, without incurring in additional computational cost. However, the analysis of such an algorithm falls outside the scope of this paper.

IV. About the Predictive Variances

The use of a Bayesian formulation provides us with predictive variances $\hat{\sigma}_{yt+1}^2$ at no additional cost, as has been exposed in the previous sections. Predictive variances are not provided in the standard KRLS formulation, but could be useful to determine confidence intervals when making a prediction, or even to discard highly uncertain predictions. In this section, we will discuss the practical details that need to be considered when using KRLS-T to produce predictive variances in addition to the standard predictive means.

When considering only predictive means, re-scalings of both the kernel $k(\mathbf{x}, \mathbf{x}')$ and the noise power σ_n^2 do not have any effect; exactly the same predictive means are obtained. However, this global scale factor does affect the predictive variances, so it is important that both $k(\mathbf{x}, \mathbf{x}')$ and the noise power σ_n^2 have the correct scale when these are used. Either this global scale is known beforehand, and the appropriately scaled kernel $k(\mathbf{x}, \mathbf{x}')$ and σ_n^2 are provided in Algorithm 1, or we must somehow estimate them. In the following, we will be concerned with global scale estimation.

Assume that from some unscaled kernel $k^{(u)}(\mathbf{x}, \mathbf{x}')$ and noise power $\sigma_n^{2(u)}$ we define the usual kernel and noise power as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_0^2 k^{(u)}(\mathbf{x}, \mathbf{x}'); \qquad \sigma_n^2 = \sigma_0^2 \sigma_n^{2(u)}$$

in which an arbitrary scaling σ_0^2 has been introduced. If we selected an appropriate value for this new hyperparameter, we would ensure that the predictive variances are correctly scaled. As we will see, it turns out that a precise estimate of σ_0^2 can be obtained in closed form, thus removing the need to have any previous knowledge about the scale of the process.

We first define at each time instant the quantity

$$\upsilon_t = \frac{\hat{\sigma}_{yt}^2}{\sigma_0^2} = \sigma_n^{2(u)} + \frac{\hat{\sigma}_{ft}^2}{\sigma_0^2}$$

which, just as the predictive mean $\hat{y}_{t+1}$, is independent of σ_0^2 (though σ_0^2 appears in this formula, it also appears implicitly in $\hat{\sigma}_{yt}^2$ and both cancel out). Then, the predictive distribution at each time instant becomes

$$p(y_{t+1}|\mathcal{D}_t) = \mathcal{N}(y_{t+1}|\hat{y}_{t+1}, \sigma_0^2 \upsilon_{t+1}) \tag{19}$$

and we can express the marginal likelihood (evidence) of σ_0^2 given all past data as

$$\begin{aligned} p(\mathcal{D}_t|\sigma_0^2) &= \prod_{i=1}^{t} p(y_i|y_1, \ldots, y_{i-1}, \sigma_0^2) \\ &= \prod_{i=1}^{t} \mathcal{N}(y_i|\hat{y}_i, \sigma_0^2 \upsilon_i). \end{aligned}$$

The latter expression can be maximized with respect to σ_0^2 to obtain its type-II maximum likelihood (ML-II) estimate, which is the standard way to perform model selection in GPs. This maximization can be done analytically and the result is

$$\hat{\sigma}_{0\mathrm{ML}}^2 = \frac{1}{t}\sum_{i=1}^{t} \frac{(y_i - \hat{y}_i)^2}{\upsilon_i}.$$

Keeping an updated estimate $\hat{\sigma}_{0\mathrm{ML}}^2$ at each time instant does not add any overhead to the algorithm and allows to compute correctly scaled predictive variances using (19) with $\sigma_0^2 = \hat{\sigma}_{0\mathrm{ML}}^2$. This value will be an increasingly good estimate of the (noiseless) signal power. Furthermore, since the algorithm becomes independent of σ_0^2 except in the computation of the final predictive variance, we can set $\sigma_0^2 = 1$, which makes $k(\mathbf{x}, \mathbf{x}') = k^{(u)}(\mathbf{x}, \mathbf{x}')$, $\sigma_n^2 = \sigma_n^{2(u)}$ and $\upsilon_t = \hat{\sigma}_{yt}^2$. Thus, Algorithm 1 can be used as is, using $\hat{\sigma}_{0\mathrm{ML}}^2 \hat{\sigma}_{yt+1}^2$ as predictive variances.

A. Scale Adaptation

In a time-varying scenario, signal power σ_0^2 may also undergo changes. If this is the case, an adaptive estimation of its value may be in place. Maximizing an exponentially weighted version of the marginal likelihood yields the following adaptive estimation:

$$a_t = a_{t-1} + \lambda \frac{(y_t - \hat{y}_t)^2}{\upsilon_t} \tag{20a}$$

$$b_t = b_{t-1} + \lambda \tag{20b}$$

$$\hat{\sigma}_{0\mathrm{ML}}^2 = \frac{a_t}{b_t} \tag{20c}$$

which, for $\lambda = 1$, corresponds to the standard ML-II presented before. Recursion is initialized using $a_0 = b_0 = 0$.

V. Numerical Experiments

In this section, we perform a number of experiments to demonstrate the regression and tracking capabilities of the KRLS-T algorithm. We start by evaluating its performance on a stationary benchmark, and then move on to illustrate its tracking performance on different experiments with simulated and real-world nonstationary data.

A. Compared Algorithms

We experimentally compare the following algorithms.

1) Approximate linear dependency KRLS (ALD-KRLS) [5] is the first KRLS algorithm proposed in the literature. It constructs an approximate solution to a batch kernel regression problem in an online manner. To slow down dictionary growth, it uses a sparsity criterion based on linear dependency. Its only parameter, apart from the kernel, is a sensitivity threshold ν which determines whether a basis will be accepted into the dictionary. If a basis is accepted, the stored variables are extended and updated to reflect its information. If not, the variables are not extended but the datum's information is still included in the posterior by means of a *partial* update. A notable characteristic of ALD-KRLS is that it does not intrinsically handle regularization, but rather achieves this by constructing a sparse basis. Remark also that ALD-KRLS is not a tracking algorithm.
2) Surprise criterion KRLS (SC-KRLS) uses a sparsification criterion based on *surprise* [7], which is "a subjective information measure of data with respect to a learning system." This criterion exploits the data labels y_t in addition to the input data $\mathbf{x}_t$, and it uses an additional threshold to distinguish outliers from redundant data. It is worth noting that if a datum causes little surprise, it is discarded as redundant, whereas ALD-KRLS would still include its information in the posterior. Also, if a new input $\mathbf{x}_t$ is already well represented by the dictionary, it can still be considered as learnable due to its corresponding output y_t being surprising enough, thus wasting a memory slot. SC-KRLS includes a regularization term that can be used to enforce smoothness. Note that SC-KRLS is not a tracking algorithm.
3) Sliding-window KRLS (SW-KRLS) achieves tracking by constructing a regression solution based only on the M last observed data and by updating this solution efficiently [11], [22]. Although conceptually simple, its performance is limited by the quality of the bases in its support, over which it has no control. In particular, it has no means to avoid redundancy in its dictionary or to maintain older bases that are relevant to its kernel expansion. Improvements to this procedure can be found in [21].
4) Naive online regularized risk minimization algorithm (NORMA) is a kernel-based version of leaky least-mean squares [4]. It is closely related to the kernel least-mean squares algorithm proposed in [23] but it includes regularization. As a result of this property, the coefficients of this filter shrink over time, allowing it to discard the oldest bases in a sliding-window fashion. Although NORMA has linear time and memory complexity in terms of the number of bases in its window, it requires many more bases to match the steady-state performance of KRLS algorithms.
5) EW-KRLS [6] is a kernelized version of the linear RLS algorithm with exponential weighting [8]. By including a forgetting factor, it is theoretically capable of performing tracking. However, it shows several weaknesses that rule out its practical usage as a tracker on long data sequences. First, it loses regularization over time, see [6, Eq. (4.52)], where β^i ends up being 0 on finite-precision machines, for $\beta < 1$ and $i \gg 1$], similar to the linear case (see [8]). This leads to stability issues. As a byproduct of regularization loss, it also loses its forgetting ability (since β^i becomes 0, thus weighting all samples equally). Finally, to perform tracking it requires evergrowing memory, as it only adapts its solution when growing its dictionary. EW-KRLS includes a regularization term, and its only parameter, apart from the kernel, is the forgetting factor.
6) The proposed KRLS-T algorithm aims to overcome the shortcomings of these algorithms in tracking scenarios, as discussed in Sections II and III of this paper. Its only free parameter, apart from the kernel, is the factor λ used in its forgetting scheme, while its dictionary size M is a budget-dependent parameter.

In the following, we present a series of experiments to compare the behavior of KRLS-T to each of the other algorithms. Unless stated otherwise, we will use a RBF kernel of the form

$$k(\mathbf{x}, \mathbf{x}') = \sigma_0^2 \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\ell^2}\right)$$

in which σ_0^2 is the signal power and ℓ is the length scale.

B. Online Regression in a Stationary Environment

In the first experiment, we train the online algorithms to perform regression of the KIN40K data set.[2] Since this is a stationary regression problem, we only consider the algorithms that are not trackers (i.e., ALD-KRLS and SC-KRLS) and compare them to KRLS-T with $\lambda = 1$. The KIN-40K data set is obtained from the forward kinematics of an eight-link all-revolute robot arm. It contains 40 000 examples, each consisting of an 8-D input vector and a scalar output. KIN40K was generated with maximum nonlinearity and little noise, representing a very difficult regression test. We randomly selected 10 000 data points for training and used the remaining 30 000 points for testing the regression.

For all algorithms we use an anisotropic Gaussian kernel in which the hyperparameters were determined offline by standard GP regression. In particular, the noise-to-signal ratio was $\sigma_n^2/\sigma_0^2 = 0.0021$. In this experiment, we wish to limit the memory of each algorithm to $M = 500$ bases. Since ALD-KRLS and SC-KRLS are controlled by a sensitivity threshold rather than by a memory budget, we determine the value of their thresholds that yields a total memory size of 500 bases at the end of the experiment. For ALD-KRLS we obtain sensitivity $\nu = 0.45$, and for SC-KRLS the threshold becomes $T_1 = 0.472$. The second threshold of SC-KRLS is set to $T_2 = +\infty$, which implies that no points are treated as outliers. We also exploit the property of ALD-KRLS of performing partial updates by running a modified version of ALD-KRLS in which we set $\nu = 0$ and maintain the dictionary unchanged after it reaches 500 bases (i.e., only partial updates

[2] Available at http://www.cs.toronto.edu/~delve/data/datasets.html.

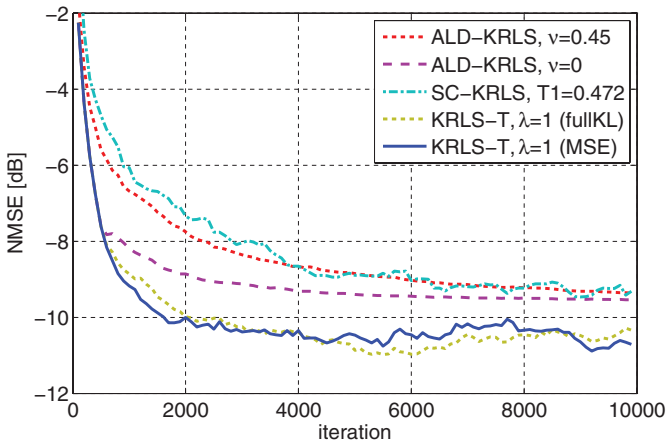

Fig. 1. NMSE comparison of different KRLS algorithms on the KIN40K regression problem. Each algorithm uses a dictionary of $M = 500$ bases.

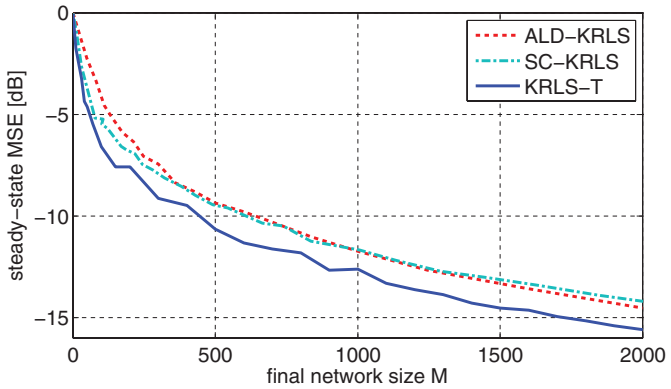

Fig. 2. Final network size M versus steady-state NMSE on the KIN40K regression problem.

are performed after this point). Finally, we apply the proposed KRLS-T algorithm with a dictionary size of $M = 500$ bases and no forgetting ($\lambda = 1$). To prune the dictionary, we use the slower criterion that minimizes KL-divergence ("fullKL") in a first test and the faster MSE-based criterion in a second test.

Each algorithm performs a single run over the data. The performance is measured as the normalized mean-square error (NMSE) on the test data set at different points throughout the training run. The results are displayed in Fig. 1. Although ALD-KRLS with $\nu = 0.45$ converges much slower than its modified version with $\nu = 0$, both versions converge to the same NMSE. SC-KRLS obtains very similar performance. KRLS-T outperforms the other algorithms by a significant margin. Thanks to its principled handling of uncertainty it is able to properly weight all samples and to trade weaker bases in the dictionary for more relevant ones during the entire experiment.

In Fig. 2, we represent the steady-state NMSE of the algorithms for different final network sizes, which allows to compare the performances of ALD-KRLS and SC-KRLS in their original growing-budget context to KRLS-T. The steady-state NMSE is calculated as the average NMSE over the last 500 iterations. As can be observed, KRLS-T requires a significantly smaller memory to obtain results similar to ALD-KRLS and SC-KRLS.

TABLE I
IMPULSE RESPONSES OF THE LINEAR CHANNELS USED IN THE SIMULATIONS

n	1	2	3	4	5
$\mathbf{h}_1[n]$	1.000	−0.3817	−0.1411	0.5789	0.191
$\mathbf{h}_2[n]$	1.000	−0.0870	0.9852	−0.2826	−0.1711

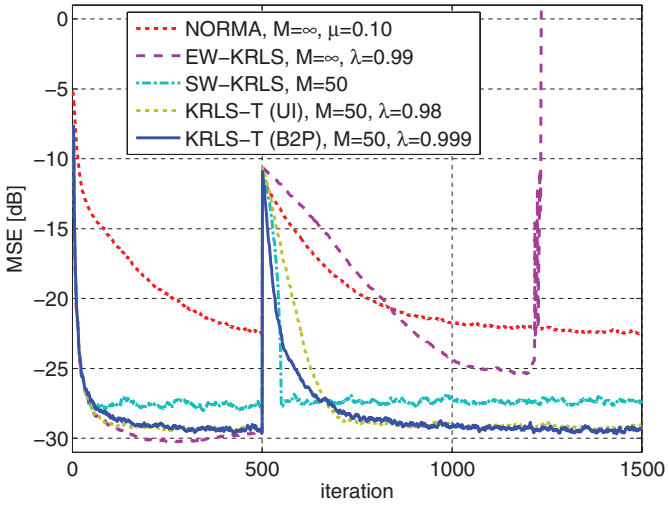

Fig. 3. MSE performance comparison of different tracking algorithms on a communications channel that shows an abrupt change at $t = 500$.

C. Adaptive Identification of a Time-Varying Communication Channel

In the second experiment, we study the capability of several tracking algorithms to re-converge after a model switch. For this experiment we consider the problem of nonlinear identification of a communication channel that undergoes an abrupt change [24]. In this setup, a signal $x_t \in \mathcal{N}(0, 1)$ is fed into a nonlinear channel that consists of a linear finite impulse response channel followed by the nonlinearity $y = \tanh(z)$, where z is the output of the linear channel. During the first 500 iterations the impulse response of the linear channel is chosen as $\mathbf{h}_1$ in Table I, and at iteration 501 it is abruptly switched to $\mathbf{h}_2$. These impulse responses were chosen randomly and re-scaled to have 1 as the first coefficient. The channel output contains 20 dB of additive Gaussian white noise.

We perform an online identification experiment, in which the identification algorithm is given one input sample and one output sample at each time instant. A time-embedding of five taps is considered, that is, the input data is taken as vectors $\mathbf{x}_t = [x_t, x_{t-1}, \ldots, x_{t-4}]^T$. The performance of each algorithm is tested on a set of 100 data points that are generated with the current channel model.

We compare the performance of different tracking algorithms on this identification problem, in particular NORMA, EW-KRLS, SW-KRLS, and KRLS-T. An RBF kernel with $\ell = 1$ is used in all algorithms, and the regularization is set to 0.01, which matches the true value of the noise-to-signal ratio. We aim to limit each algorithm's budget to $M = 50$. Exceptions are made for NORMA, which requires a larger dictionary to obtain a performance

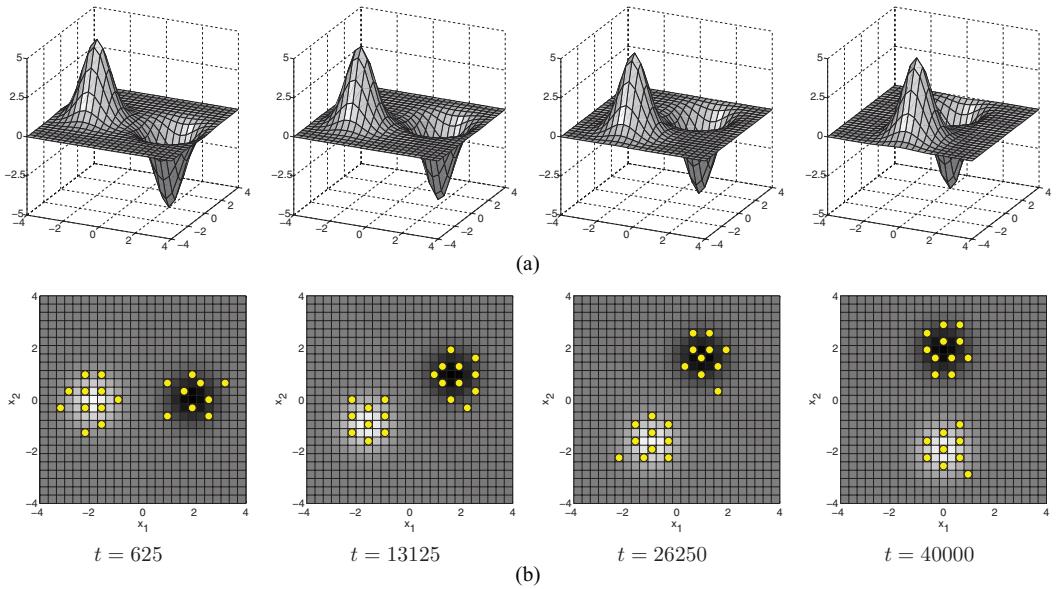

Fig. 4. (a) Observed surface over time. (b) Surface predictions by KRLS-T, and active bases (dots) over time.

comparable to KRLS, and for EW-KRLS, which cannot perform tracking without growing its dictionary. These algorithms are given an evergrowing memory, indicated as $M = \infty$. The adaptation rates are chosen as follows. NORMA uses learning rate $\eta = 0.1$, EW-KRLS has forgetting factor $\lambda = 0.99$ and KRLS-T is first applied with UI forgetting and $\lambda = 0.98$ and then with B2P forgetting and $\lambda = 0.999$. The values of λ were chosen to obtain similar convergence levels. As discussed in Section III, the same value of λ does not correspond to the same level of adaptivity in both forgetting strategies.

The identification results, averaged out over 25 simulations, can be found in Fig. 3. EW-KRLS is capable of performing tracking, but when implemented on a finite-precision machine it encounters numerical problems after a certain number of iterations. As additional tests confirmed, lowering its forgetting factor leads to a similar steady-state MSE but causes the numerical problems to occur earlier. SW-KRLS obtains reasonable results on this example, indicating that its memory of the 50 latest samples is sufficient to identify the channel satisfactorily. The proposed KRLS-T algorithm with UI forgetting and $\lambda = 0.98$ obtains a good steady-state MSE, but its convergence rate after the channel switch is somewhat slower than SW-KRLS. Note that this rate can be increased at the cost of a higher steady-state MSE. Lastly, KRLS-T with B2P forgetting and $\lambda = 0.999$ outperforms the other algorithms, both in terms of convergence rate and MSE convergence. (EW-KRLS obtains better MSE convergence during the first iterations of this experiment but it uses more memory.) This indicates that the B2P-forgetting technique is a more sensible approach to forgetting. Note that KRLS-T with B2P forgetting maintains its regularization at all times, unlike the UI-forgetting technique. Finally, while SW-KRLS reaches convergence earlier after the channel switch, KRLS-T (B2P) initially converges much faster and obtains a better steady-state MSE.

D. Online Regression of a Nonstationary 2-D Mapping

In the following simulation, we illustrate the capability of the KRLS-T algorithm to adapt its dictionary. Specifically, we are interested in the algorithm's efficiency to adjust its regression solution when the regions of interest are changing throughout the input space. To force the algorithm to adapt its dictionary, we only allow it to use very few bases.

The scenario of this simulation consists of a 2-D surface, which could for instance represent the light intensities of the sky containing different moving celestial bodies. Assume our observations are made by a sensor (such as a telescope) that can only capture a small region of the entire space at each time instant. To assemble a global picture of the observed space, we need to scan it taking various snapshots at different positions. Here we will scan the space by taking successive observations along the nodes of a grid, column by column, although it could be plausible to target different regions of the space more frequently if they were considered of interest. The number of grid points is chosen as 625, which corresponds to 25 columns and 25 rows. The following parameterized 2-D surface is simulated:

$$y = 5\exp\left(-(x_1 + 2\cos(\omega t))^2 - (x_2 + 2\sin(\omega t))^2\right) \\ -5\exp\left(-(x_1 - 2\cos(\omega t))^2 - (x_2 - 2\sin(\omega t))^2\right)$$

in which x_1 and x_2 represent coordinates in the input space, ω is a constant fixed at $\omega = \pi/(2 \times 40\,000)$, and $t = 1, 2, \ldots$ represents the temporal index. At each time instant t, one observation y_t is obtained at one node $\mathbf{x}_t$ of the input space grid. This surface contains two Gaussian bumps, one positive and one negative, that perform a rotation of $\pi/2$ radians between $t = 1$ and $t = 40\,000$, as illustrated in Fig. 4(a). Note that these Gaussians are moving considerably slower than the period required to scan the entire grid.

Given the successive input–output patterns $\{\mathbf{x}_t, y_t\}$ we apply KRLS-T to model the entire observed space, using

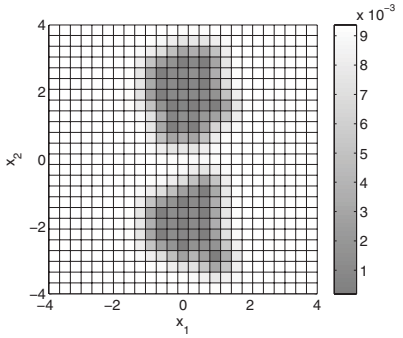

Fig. 5. Predictive variance obtained by KRLS-T, which quantifies its uncertainty about its own predictions.

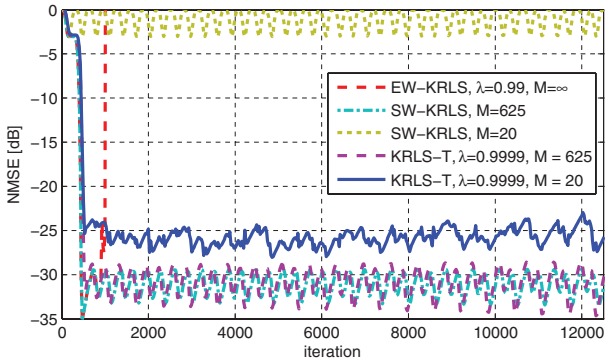

Fig. 6. NMSE results of predicting the nonstationary surface.

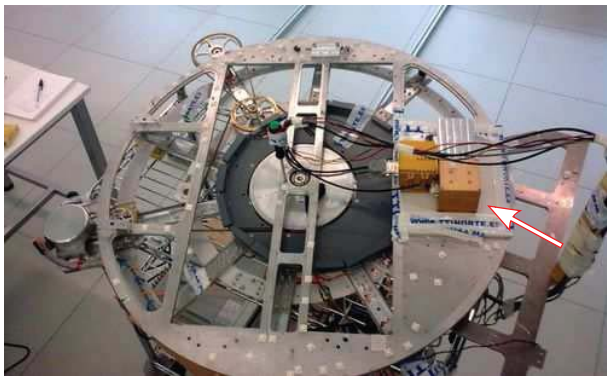

Fig. 7. Picture of the experimental setup. The IMU is contained within the box indicated by the arrow.

back-to-the-prior forgetting, $\lambda = 0.9999$ and 10^{-5} regularization. The number of bases for KRLS-T is chosen as $M = 20$, which is substantially lower than the number of grid nodes. Fig. 4(b) shows snapshots of the active bases chosen at four time instants during the experiment. The first snapshot is taken after the first full scan of the grid. As can be observed, KRLS-T chooses its bases to cover the entire region of interest, and it adapts its dictionary correctly over time. In Fig. 5, we illustrate the predictive variance of KRLS-T about the entire space, at $t = 40\,000$. The lower the predictive variance, the more confident the algorithm is about its prediction. Note that the regions of high confidence correspond nicely to the positions of the active bases at this time step, which are shown in Fig. 4(b).

Next, we compare KRLS-T's performance to EW-KRLS and SW-KRLS. We measure the prediction error of the entire surface, during the entire experiment. EW-KRLS is applied with $\lambda = 0.99$ and evergrowing memory. SW-KRLS and KRLS-T are first given the full budget $M = 625$ to obtain their baseline performance, and then a reduced budget of $M = 20$, as in Fig. 4. The results are shown in Fig. 6. EW-KRLS runs into numerical problems that cause it to diverge after 1000 iterations. With the full budget $M = 625$, SW-KRLS and KRLS-T both obtain very good results that oscillate between -30 and -33 dB. With the reduced budget, the strength of KRLS-T's basis pivoting mechanism becomes clear. It detects which bases should be maintained in the dictionary and which ones should be replaced, during the entire experiment. SW-KRLS does not possess such a mechanism, and it performs poorly. It cannot adequately model examples that have received more than M time steps before.

E. Modeling the Changing Dynamics of an Attitude Control System

In the last experiment, we illustrate the proposed algorithm's capability to model changing nonlinear dynamics using real-world data gathered from an attitude estimation system. Attitude estimation is concerned with determining the orientation of a vehicle, commonly an aircraft, and it is essential in flight control and navigation [25]. The basic components of attitude estimation systems are inertial measurement units (IMUs) that typically contain three-axis accelerometers to measure acceleration and three-axis gyroscopes to measure the vehicle's angular velocities. The performance of these components is affected by sensor noise, especially in the accelerometers, and sensor bias, which can lead to significant drift when integrated out. For these reasons many navigation systems rely on additional components such as GPS, pedometers, or a magnetic compass [26].

The experimental setup, which has been used for a parallel ongoing research project, is shown in Fig. 7. The IMU, which contains low-end accelerometers and gyroscopes, is mounted on a round table that is rotated by a motor and that provides acceleration and angular velocity measurements. The tracking algorithm needs to learn the slowly changing nonlinear relationship between the gyroscope measurements and the rotation angle reported by the motor, on the one hand, and the noisy acceleration signal, on the other hand. By modeling this relationship, a de-noised version of the instantaneous acceleration signal is obtained, which can be used to improve different aspects of the navigation system such as the detection of movement and the calibration of the accelerometers.

The described setup includes a nonstationary component in the form of the gyroscope bias, which changes slowly over time. We add further nonstationarity by triggering abrupt changes in the inclination of the table at different time instants, which require the learned model to re-converge.

Fig. 8 (top) illustrates the table movement. The initial position of the IMU is referenced as a table rotation of 0°. The table then performs a series of 360° rotations,

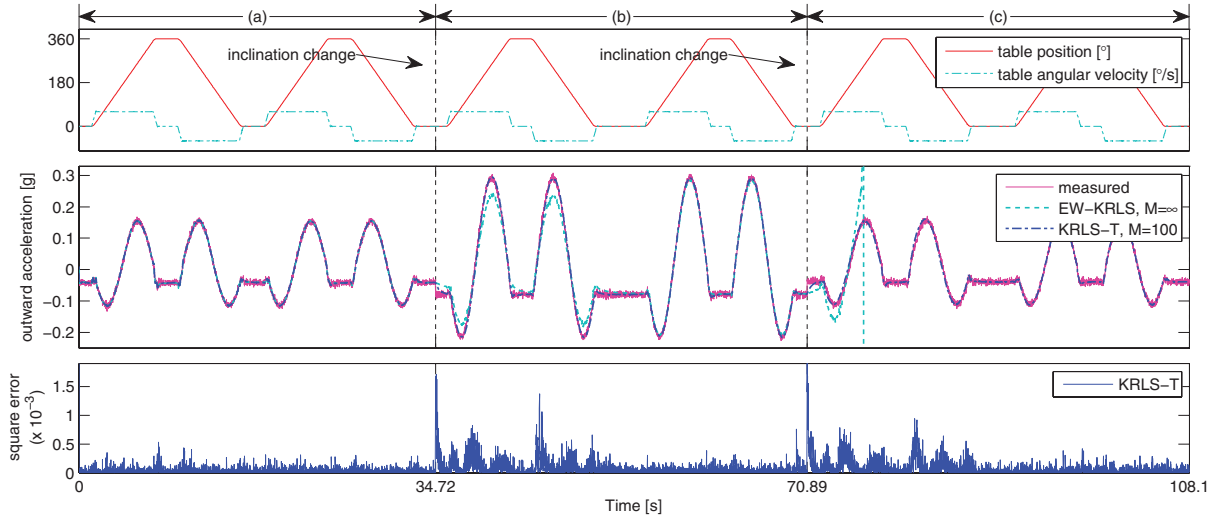

Fig. 8. Results of the fourth experiment. Top plot: angular position measured by the table motor and angular velocity measured around the (upward-pointing) z-axis of the IMU, which are used as the input signals in the experiment. During segment (a) table inclination is 7.9°, (b) at 34.72 s the inclination is changed to 14.5°, and (c) at 70.49 s it is changed back to 7.9°. Middle plot: measured outward acceleration and learned de-noised accelerations. Bottom plot: square error between the measured acceleration and de-noised acceleration obtained by KRLS-T.

alternatingly counterclockwise and clockwise. At the start of the experiment, the table inclination amounts to 7.9°. After four entire rotations, the inclination is changed to 14.5°, and afterwards it is returned to 7.9°. The input signal in this experiment consists of the angular velocity measured around the (upward-pointing) z-axis of the IMU, and the table rotation angle reported by the motor. The desired output signal is the noisy outward acceleration as measured by the corresponding accelerator of the IMU. All signals are sampled at a rate of 100 samples per second. A Gaussian kernel is used on the input data, and its hyperparameters are determined by offline GP regression on a data register containing a single revolution at a table inclination of 7.9°.

We perform tracking by KRLS-T (B2P) with $\lambda = 0.999999$, and as a reference we include results for EW-KRLS with $\lambda = 0.999$. Although the maximum budget of KRLS-T is fixed at 100 bases, EW-KRLS is allowed evergrowing memory. The results of the tracking experiment can be seen in Fig. 8. The middle plot shows the measured noisy acceleration and the learned de-noised accelerations. A misadjustment of the algorithms is clearly visible after every table inclination change, and both algorithms require around two rotations to re-converge. EW-KRLS diverges around the instant $t = 76.4$ s, where it uses a memory of $M = 7640$ slots. Although its divergence can be postponed by raising λ, this would reduce its already poor tracking ability. Furthermore, it is clear that its evergrowing budget causes a too large computational and memory burden when using long data sequences. KRLS-T obtains good results. Its square error, visualized in the bottom plot, shows that it is capable of reconverging and reaching excellent steady-state performance after each model change.

VI. Conclusion

In this paper, we have addressed an issue that we believe has been overlooked to some extent in kernel-based versions of adaptive filtering algorithms, in particular the tracking ability in nonstationary scenarios. Although linear adaptive filtering algorithms can typically be used on both stationary and nonstationary scenarios with little or no modifications, this is not true for kernel adaptive filtering algorithms. These algorithms are commonly designed to construct the solution to a batch problem in an online manner, and up till this point the only approaches that aimed to perform tracking on nonstationary data were formulated either as an evergrowing network or in a sliding-window fashion, both of which show numerous difficulties.

We have introduced a Bayesian framework that unifies existing KRLS theory and provides additional insight by explicitly handling uncertainty, which allows to define the concept of "forgetting" in a natural manner in the context of KRLS. Then, we have described two sensible forgetting techniques, and we have shown how one of them reduces exactly to the exponentially weighted RLS algorithm when a linear kernel is used. The presented framework naturally introduces regularization into KRLS, and it provides uncertainty estimates for its predictions, which can be turned directly into confidence intervals. Finally, we have combined these ideas into a concrete algorithm, KRLS-T, which works with fixed memory and computational requirements per time step, and allows for simple, practical implementation.

We have included different numerical experiments that show how the proposed algorithm outperforms existing online kernel methods not only in the nonstationary scenarios for which it was designed, but also in stationary scenarios (by setting its forgetting factor to $\lambda = 1$) due to its basis trading mechanism and its more rigorous approach to regularization.

The described Bayesian framework opens the door to a number of interesting future research lines. First, although the two proposed types of forgetting are well-motivated, our proposal allows for other forgetting schemes. Moreover, by using a linear kernel, each type of forgetting yields a corresponding linear adaptive filtering algorithm. Finally, the

proposed tracking algorithm and framework could be extended to the fields of complex [27] and quaternionic nonlinear adaptive filtering [28].

ACKNOWLEDGMENT

The authors would like to thank TTI Norte, Santander, Spain, for providing us the data and picture used in the experiment on inertial measurement units.

REFERENCES

[1] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA: MIT Press, Dec. 2001.
[2] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. London, U.K.: Cambridge Univ. Press, Jun. 2004.
[3] C. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. Cambridge, MA: MIT Press, 2000, pp. 682–688.
[4] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2165–2176, Aug. 2004.
[5] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least squares algorithm," *IEEE Trans. Signal Process.*, vol. 52, no. 8, pp. 2275–2285, Aug. 2004.
[6] W. Liu, J. C. Príncipe, and S. Haykin, *Kernel Adaptive Filtering: A Comprehensive Introduction*. New York: Wiley, 2010.
[7] W. Liu, I. Park, and J. C. Príncipe, "An information theoretic approach of designing sparse kernel adaptive filters," *IEEE Trans. Neural Netw.*, vol. 20, no. 12, pp. 1950–1961, Dec. 2009.
[8] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice Hall, Sep. 2001.
[9] A. Sayed, *Fundamentals of Adaptive Filtering*. New York: Wiley, 2003.
[10] W. Liu, I. Park, Y. Wang, and J. C. Príncipe, "Extended kernel recursive least squares algorithm," *IEEE Trans. Signal Process.*, vol. 57, no. 10, pp. 3801–3814, Oct. 2009.
[11] S. Van Vaerenbergh, J. Vía, and I. Santamaría, "A sliding-window kernel RLS algorithm and its application to nonlinear channel identification," in *Proc. IEEE Int. Conf. Acoustics Speech Signal Process.*, vol. 5. Toulouse, France, May 2006, pp. 789–792.
[12] L. Csató and M. Opper, "Sparse online Gaussian processes," *Neural Comput.*, vol. 14, no. 3, pp. 641–668, 2002.
[13] M. Lázaro-Gredilla, S. Van Vaerenbergh, and I. Santamaría, "A Bayesian approach to tracking with kernel recursive least-squares," in *Proc. IEEE Int. Workshop Mach. Learn. Signal Process.*, Sep. 2011, pp. 1–6.
[14] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006.
[15] L. Csató and M. Opper, "Sparse representation for Gaussian process models," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2001, pp. 444–450.
[16] B. De Kruif and T. De Vries, "Pruning error minimization in least squares support vector machines," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 696–702, May 2003.
[17] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," *Adv. Neural Inform. Process. Syst.*, vol. 2, no. 1, pp. 598–605, 1990.
[18] O. Dekel, S. Shalev-Shwartz, and Y. Singer, "The forgetron: A kernel-based perceptron on a fixed budget," in *Advances in Neural Information Processing Systems*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, pp. 259–266.
[19] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF GGAP-RBF neural network for function approximation," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 57–67, Jan. 2005.
[20] A. Kuh and P. De Wilde, "Pruning error minimization in least squares support vector machines," *IEEE Trans. Neural Netw.*, vol. 18, no. 2, pp. 606–609, May 2007.
[21] S. Van Vaerenbergh, I. Santamaría, W. Liu, and J. C. Príncipe, "Fixed-budget kernel recursive least-squares," in *Proc. IEEE Int. Conf. Acoustics Speech Signal Process.*, Apr. 2010, pp. 1882–1885.
[22] S. Van Vaerenbergh, J. Vía, and I. Santamaría, "Nonlinear system identification using a new sliding-window kernel RLS algorithm," *J. Commun.*, vol. 2, no. 3, pp. 1–8, May 2007.
[23] W. Liu, P. P. Pokharel, and J. C. Príncipe, "The kernel least-mean-square algorithm," *IEEE Trans. Signal Process.*, vol. 56, no. 2, pp. 543–554, Feb. 2008.
[24] S. Van Vaerenbergh, J. Vía, and I. Santamaría, "Adaptive kernel canonical correlation analysis algorithms for nonparametric identification of Wiener and Hammerstein systems," *EURASIP J. Adv. Signal Process.*, vol. 1, pp. 1–13, Apr. 2008.
[25] D. Titterton and J. Weston, *Strapdown Inertial Navigation Technology*. Stevenage, U.K.: Peregrinus, 2004.
[26] H. Rehbinder and X. Hu, "Drift-free attitude estimation for accelerated rigid bodies," *Automatica*, vol. 40, no. 4, pp. 653–659, 2004.
[27] P. Bouboulis, K. Slavakis, and S. Theodoridis, "Adaptive learning in complex reproducing kernel hilbert spaces employing wirtinger's subgradients," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 3, pp. 425–438, Mar. 2012.
[28] B. Che Ujang, C. Took, and D. Mandic, "Quaternion-valued nonlinear adaptive filtering," *IEEE Trans. Neural Netw.*, vol. 22, no. 8, pp. 1193–1206, Aug. 2011.

Steven Van Vaerenbergh (S'06–M'11) received the B.S. degree in electrical engineering from Ghent University, Ghent, Belgium, in 2003, and the Ph.D. degree from the University of Cantabria, Santander, Spain, in 2010.

He is a Post-Doctoral Associate with the Department of Telecommunications Engineering, University of Cantabria. He was a Visiting Researcher with the Computational NeuroEngineering Laboratory, University of Florida, Gainesville. His current research interests include machine learning, information theory, and their applications to adaptive learning and tracking.

Miguel Lázaro-Gredilla (M'11) received the Telecommunication Engineering degree (Hons.) from the University of Cantabria, Santander, Spain, and the Ph.D. degree (Hons.) from the Universidad Carlos III de Madrid, Leganés, Spain, in 2004 and 2010, respectively.

He is currently a Visiting Professor with the Universidad Carlos III de Madrid, after stays with the University of Cambridge, Cambridge, U.K., the University of Manchester, Manchester, U.K., and the University of Cantabria. His current research interests include Gaussian processes and Bayesian models.

Ignacio Santamaría (M'96–SM'05) received the Telecommunication Engineering degree and the Ph.D. degree in electrical engineering from the Polytechnic University of Madrid, Madrid, Spain, in 1991 and 1995, respectively.

He joined the Department of Telecommunications Engineering, University of Cantabria, Santander, Spain, in 1992, where he is currently a Full Professor. He was a Visiting Researcher with the Computational NeuroEngineering Laboratory, University of Florida, Gainesville, and the Wireless Networking and Communications Group, University of Texas at Austin, Austin. He has published more than 100 publications in refereed journals and international conference papers. His current research interests include signal processing algorithms for wireless communication systems, multivariate statistical techniques, and machine learning theories.