2012 American Control Conference
Fairmont Queen Elizabeth, Montréal, Canada
June 27-June 29, 2012

Adaptive Inverse Control Using Kernel Identification

Andrea Abelli, André Ferrari, Salvatore Monaco and Cic Richard

Abstract— **Kernel methods are exploited to implement an adaptive inverse control scheme of which a first introductory presentation is given. The resulting controller has faster convergence than the solutions proposed in literature utilizing Support Vector Machines (SVMs) [1] and Artificial Neural Networks (ANNs) [2]. Smaller residual errors are obtained for trajectory tracking. Simulations are carried out for different scenarios.**

I. Introduction

Analytically speaking, the control of a known nonlinear dynamics can be a hard problem to solve. Since Control Theory was born the linear system branch has uninterruptedly attracted the far greatest deal of studies and efforts so that, nowadays, the design of linear controllers, even for nonlinear systems [3], is supported by a vast theoretical and practical literature. In fact, most of the controllers used in the industry, due to their simplicity of implementation and ease of use, belong to this class. But as the plant becomes more complex some difficulties arise: modeling a highly nonlinear system is everything but a trivial task. Models are inevitably less precise and the feedback schemes more complex implying higher noise levels.

In the last few decades, some efficient but theoretically challenging methods have been developed to deal with these problems: feedback linearization, Lyapunov techniques, etc. [4], [5]. Among the most widespread "local" methods, *gain scheduling* [3], where the plant dynamics is linearized around a certain number of operating points for each one of which a linear controller is synthesized and employed, has gained a large domain of use. Unfortunately, being model based, a precise model needs to be known and the dynamics needs not to move away from preselected operating zones. Another popular method is *feedback linearization* [6], which is based on the dynamic inversion of the plant model, but as above an accurate model of the plant needs to be known and the inverse of the plant dynamics needs to exist, which is generally not the case of more complex cases typical of the nonlinear context.

One particular method, derived from signal processing techniques, immune from these last constraints is Adaptive Inverse Control (AIC) [7]. Firstly proposed by Widrow in 1986, in AIC the "local inverse" of the plant model is used as serial controller such that the open loop approximates some predetermined function in some optimal (Least-Mean Squares (LMSs), Least Squares (LSs), etc..) sense. The principles of AIC can be applied to the control of nonlinear plants using nonlinear adaptive filtering techniques. Since in general, *nonlinear systems do not have inverses, the resulting controller is an acceptable inverse only for the particular control input currently fed to the plant.*

The literature is populated by some attempts to free the designer from the complexity of nonlinear filtering techniques, *e.g.*, [1], [2]. In [1] a pure signal processing approach is used: the method is developed as an adaptive-filtering problem. First, the dynamical system is identified using adaptive system-identification techniques training an ANN with a Real-Time Recurrent Learning (RTRL) [8] algorithm. Then, the dynamic response of the system is controlled using an adaptive feedforward controller, a neural network adapted with a Back-Propagation Through Time (BPTT) [9] algorithm, therein called Back-Propagation Through Plant Model (BPTM). No proper feedback is used, except that the system output is monitored and used by an adaptive algorithm to adjust the controller's parameters. In [2] an adaptive inverse control algorithm is proposed by combining the power of Reproducing Kernel Hilbert Spaces (RKHSs), exploited by on-line Support Vector Regression (SVR), with inverse control. Because the training speed of standard on-line SVR algorithms is generally low, a kernel cache is introduced to accelerate the standard algorithm. Then the customized algorithm is applied to approximate the inverse of the plant, and the residual output is used to adjust the support vector machine. Simulation results show that fair control performance for linear systems and gently nonlinear dynamics can be achieved. For acceptable performance, the algorithm needs to be trained off-line in a neighborhood of the plant's operating point, which implies that the state trajectory needs to stay in its proximity. These methods are directly affected by all the limitations typical of the controller implementation they use. Neural networks suffer from weak generalization ability and their structure must be designed prior to training and usage. Algorithms used to train ANNs are usually slow, require a large diversity of training data for realistic applications and can get stuck in local minima. Furthermore, on-line training converges slowly and it's theoretically non-trivial, *i.e.*, Decoupled Extended Kalman Filter (DEKF) [10]. SVRs on the other side have a good generalization ability, they are the optimal solution of the cost functional used to design them, generally of the form $J[f] = \frac{1}{l}\sum_{i=1}^{l} |y_i - f(\mathbf{x}_i)|_\epsilon + \lambda \|f\|^2_{RKHS}$, $|\cdot|_\epsilon$ being

A. Abelli is with Laboratoire J.L. Lagrange, Université de Nice-Sophia Antipolis, CNRS, Observatoire de la Côte d'Azur, France. andrea.abelli@uniroma1.it

F. Ferrari is with Laboratoire J.L. Lagrange, Université de Nice-Sophia Antipolis, CNRS, Observatoire de la Côte d'Azur, France. andre.ferrari@unice.fr

C. Richard is with Laboratoire J.L. Lagrange, Université de Nice-Sophia Antipolis, CNRS, Observatoire de la Côte d'Azur, France. cedric.richard@unice.fr

S. Monaco is with Dipartimento di Informatica e Sistemistica, Università di Roma, Italy. salvatore.monaco@uniroma1.it

the Vapnik ϵ-insensitive norm [11]. This kind of predictors have a parametric form, *e.g.*, $\hat{f}(\mathbf{x}) = \langle \mathbf{w}, \varphi_{\mathbf{x}} \rangle$, where $\mathbf{w}$ is the parameters vector, φ a finite-dimensional mapping. SVRs belong to a wider class of algorithms called kernel methods at whose foundation is the Representer Theorem (RT) [12]. Kernels can be directly applied to the solution of nonlinear regression problems, the RT guaranteeing that if the regularization functional is strictly increasing all the solutions will have same form as (1)

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{k} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i) \ , \tag{1}$$

where $\{\mathbf{x}_l\}_{l=1}^{d}$ are the training sample points, $\kappa(\cdot,\cdot)$ is a Mercer kernel and solving the problem boils down to solving for the α_i's. In both these approaches the solution has the same size of the training sets, which increases at each time step, k. To overcome this limitation, different approaches, *i.e.*, sparsification methods, must be taken.

For SVRs, soft-cost functionals combining some error-insensitive component to a smoothing term are used, usually, leading to sparse solutions. Still, due to their complexity only few real-time SVR algorithms have been developed and their performance don't fit practical usage: they scale superlinearly in the training set size, *e.g.*, compared to the method presented in this work, an on-line SVR algorithm as in [2] works with "dictionaries" whose size is one or two (dependently on the plant complexity) orders of magnitude bigger, implying a much heavier calculatory burden.

In the case of kernels, attention is given to construct a sparse dictionary of "informative" points, for which the kernel matrix will be calculated. Sparsity reduces the complexity in terms of computation and memory, and it usually gives better generalization ability to unseen data [13]. To achieve this, some methods were developed. Among those, the *novelty criterion* [13] in which when a new data point $\mathbf{x}_k$ is obtained by the network, the distance of this point to the actual dictionary is calculated. If this distance is smaller than some preset threshold, $\mathbf{x}_k$ will not be added to the dictionary. Otherwise, the prediction error is computed and if it's larger than some other preset threshold, $\mathbf{x}_k$ will be accepted as a new dictionary sample; an alternative approach is based on the *approximate linear dependency criterion* as described in Sec. II-C.

This work is a first attempt to bring together the concept of AIC with the strengths of numerical recursive kernel methods-based algorithms. The latter allow to effectively isolate nonlinearities and then handle their inversion in a local, and thus feasible, manner. The Kernel Recursive Least-Square (KRLS) algorithm [14] is used to implement an adaptive inverse controller capable of forcing a suitable nonlinear dynamics, *i.e.*, satisfying the conditions given in Sec. III, to follow a desired output and at the same time check if that output is admissible for the nonlinear dynamics.

The rest of this paper is organized as follows: in Sec. II the basic theory behind the RKHS transformation of the least-squares regression algorithm is introduced. Adaptive inverse control basics and a first approach to the numerical solution of the general output tracking problem [15] are given in Sec. III. In Sec. IV the method is applied to some elementary nonlinear control problems.

II. Kernel Recursive Least-Squares Algorithm

A. The Least-Squares Criterion

Given a set of training input/output couples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\} \subset \mathcal{X} \times \mathbb{R}$, where $\mathcal{X} \subset \mathbb{R}^n$ is the space of the input patterns. The (LS) least-squares problem [16] consists in finding the optimal vector $\mathbf{a}^* \in \mathbb{R}^n$ given by

$$\min_{\mathbf{a}^*} \|\mathbf{y} - X\mathbf{a}\|^2 \ . \tag{2}$$

where $X \in \mathbb{R}^{m \times n}$ is the input data matrix. The solution $\mathbf{a}^*$ is unique if, and only if, the data matrix X has full column rank (*i.e.*, all its columns are linearly independent, which necessarily requires $m \geq n$: over-determined case). In this case, $\mathbf{a}^*$ is given by

$$\mathbf{a}^* = \left(X^T X\right)^{-1} X^T \mathbf{y} \ . \tag{3}$$

When $X^T X$ is singular many solutions $\mathbf{a}^*$ exist, the one that has the smallest euclidean norm is the one that solves

$$\begin{aligned} \min_{\mathbf{a}^*} \quad & \|\mathbf{a}^*\|^2 \\ \text{s.t.} \quad & X^T X \mathbf{a}^* = X^T \mathbf{y} \end{aligned} \tag{4}$$

and $\mathbf{a}^* = X^+ \mathbf{y}$. To improve *generalization* and increase the smoothness of the solution a regularization term is often used

$$J = \min_{\mathbf{a}} \left[\|\mathbf{y} - X\mathbf{a}\| + \lambda \mathbf{a}^T \mathbf{a}\right] \ , \tag{5}$$

where $\lambda \in \mathbb{R}$ is a regularization constant. This problem is known as *regularized least-squares* and has the following optimal solution

$$\mathbf{a}^* = \left(X^T X + \lambda I\right)^{-1} X^T \mathbf{y} \ . \tag{6}$$

Recursive Least-Square (RLS) deal with the problem of taking in consideration new observations as they become available. The previously exposed LS method would imply a very high computational cost if applied on-line since at each time instant, an ever growing, in size, matrix inversion $\left(X^T X\right)^{-1}$ would have to be performed. In the case of linear systems, the RLS algorithm solves recursively the regularized LS problem using the Woodbury matrix identity [17].

B. Kernel Methods

In recent years many efforts were put in extending the scopes of many adaptive algorithms to nonlinear problems, combining the adaptive characteristics of traditional linear adaptive filters with the capability of kernel methods to 'convexify' nonlinear problems.

Kernel methods are based on the idea of mapping through a nonlinear transformation $\varphi : \mathcal{X} \to \mathbb{R}^h$ (where h might be infinity) the input sample $\mathbf{x}_i \in \mathcal{X}$ into a higher-dimensional Hilbert space (*feature space*) in which the transformed data is more likely to be linearly separable. As explained in [18]: "the key property of kernel methods is that scalar products in the feature space can be seen as nonlinear (kernel) functions of the data in the input space". Every continuous function $\kappa(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel. According to Mercer's theorem [19], any positive definite kernel function satisfying Mercer's condition (a *Mercer* kernel), $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \varphi_{\mathbf{x}_i}, \varphi_{\mathbf{x}_j} \rangle$, has an implicit mapping to some higher-dimensional feature space. This allows to perform any conventional scalar product based algorithm in the feature space by solely replacing the scalar products with the Mercer kernel function in the input space.

This simple and elegant idea is known as the *kernel trick*, and it is commonly applied by using a nonlinear kernel such as the Gaussian

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) , \tag{7}$$

which implies an infinite-dimensional feature space.

Using such a technique, a nonlinear mapping can be represented as a linear combination of kernels evaluated on a dictionary of *support vectors* (SVs) $\mathbf{x}_i$, as (1). The RT assures us that f can be approximated arbitrarily well by choosing the training vectors as SVs.

C. Nonlinear Least Squares: a kernel-based approach

Consider the transformed data matrix $\varphi_X \in \mathbb{R}^{m\times h}$ and the corresponding coefficient vector $\varphi_{\mathbf{a}} \in \mathbb{R}^h$, the LS problem can be written in the feature space as

$$J = \min_{\varphi_{\mathbf{a}}} \|\mathbf{y} - \varphi_X \varphi_{\mathbf{a}}\|^2 . \tag{8}$$

Since φ_X is highly dimensional, the solution $\varphi_{\mathbf{a}}$ can now also be represented in the basis defined by its rows, $\varphi_{\mathbf{a}} = \varphi_X^T \boldsymbol{\alpha}$. Introducing the kernel matrix $\mathbf{K} = \varphi_X \varphi_X^T$, the LS problem in feature space can be written as

$$J = \min_{\boldsymbol{\alpha}} \|\mathbf{y} - K\boldsymbol{\alpha}\|^2 . \tag{9}$$

in which the solution $\boldsymbol{\alpha}$ is in $\mathbf{R}^m$. The advantage of writing the nonlinear LS problem in the dual notation is that thanks to the "kernel trick", we only need to compute $K_{[i,j]} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. In [14], a sparsification process based on a dictionary of "meaningful" past samples was proposed. For every new data couple $\{\mathbf{x}_n, y_n\}$, the algorithm tests whether the transformed input point $\varphi_{\mathbf{x}_k}$ is approximately linearly dependent (ALD) on the dictionary samples, by calculating the residual error $\delta_k = \min_{\boldsymbol{\alpha}} \left\| \sum_{i=1}^{d_{k-1}} \alpha_i \varphi_{\mathbf{x}_i} - \varphi_{\mathbf{x}_k} \right\|^2$ where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_{d_k}]^T$ is a vector containing the expansion coefficients of the linear combination of dictionary samples, as in (1). If δ_k does not exceed a certain threshold ν , the new data point $\mathbf{x}_i$ can be approximated sufficiently well in feature space by a linear combination of the samples stored in the current dictionary of size d_k. On the other hand, if $\delta_k > \nu$, the current dictionary does not represent the new data point sufficiently well and it must be expanded. In this case, $\mathbf{x}_i$ is added to the dictionary, yielding $D_k = D_{k-1} \cup \{\mathbf{x}_i\}$ and $d_k = d_{k-1} + 1$.

III. Adaptive Inverse Control

We now derive a control scheme that can be fruitfully used to obtain a numerical solution of the general tracking problem [15]. Thanks to its inherent local nature, this method is able to cope with a high number of nonlinearities: at each instant, the system dynamics is decomposed locally around the actual operating point, making it easier to be handled. For simplicity, we limit our discussion only to single input - single output (SISO) systems. However, the method here discussed can be extended to a MIMO application. We consider the class of nonlinear discrete-time nonlinear systems described by

$$\Sigma : \qquad \mathbf{x}_{k+1} = f(\mathbf{x}_k, u_k), \qquad y_k = h(\mathbf{x}_k) , \tag{10}$$

where k is the discrete time, $\mathbf{x}_k \in \mathcal{X} \subset \mathbb{R}^n$ the state, $y_k \in \mathcal{Y} \subset \mathbb{R}$ the output, $u_k \in \mathcal{U} \subset \mathbb{R}$ the control input, n is the state dimension. Given a neighborhood $B_Y \subset \mathcal{Y}$ and an arbitrary sequence $\{y_k^*\}$ such that $y_k^* \in B_Y$, we shall find the input sequence $\{u_k^*\}$ such that

$$\lim_{k\to\infty} |y_k - y_k^*| = 0 \tag{11}$$

and $y_k \in B_Y$.

The structure of a typical adaptive inverse control system is shown in Fig. 1. Control of plant dynamics is obtained by preceding the plant by an adaptive controller (14) whose dynamics are a type of inverse of those of the plant. In general, the plant need to be stable otherwise it has to be stabilized by standard feedback and then controlled with the proposed method. We consider the controller not having previous knowledge of the plant's model, except it's relative degree.

We might like the controlled system to have a Input-Ouput (I/O) function conforming to some predetermined model. Then, by preceding the plant by a filter whose transfer function is the product of the I/O function of the model and the inverse of the plant, a controller is implemented which gives the desired I/O relationship. When adaptation has converged, the cascade of C and the plant will have a I/O relation equal to M's, see Fig. 1.

As pointed out above, if the plant is strictly causal, then its inverse will be non-causal. In such a case, a delayed inverse can still be obtained. This is done by incorporating a pure delay term Δ into the filter M. This delay is not a deficiency of the method: it's inevitable with any controller for such a plant.

A. Sufficient Conditions Of Applicability

Sufficient conditions for the applicability of the presented method (future works will release some of the constraints) to (10) for the solution of the exact tracking problem [15] are: *a)* $f(\mathbf{0}, 0) = \mathbf{0}$ and $h(\mathbf{0}) = 0$; *b)* the linearization around the origin, Σ_L, of Σ is observable; *c)* $(\mathbf{x}, u) = (\mathbf{0}, 0)$ is an asymptotically stable equilibrium of Σ.

For sufficiently small inputs and outputs, this class of systems admits an exact I/O representation Σ_{IO}

$$y_{k+1} = f(y_k, \ldots, y_{k-n+1}, u_k, \ldots, u_{k-n+1}) , \tag{12}$$

If Σ_{IO} has a well defined *relative degree* [20], r, it can be re-written as

$$y_{k+1} = f\left(y_k, \ldots, y_{k-n+1}, u_{k-r+1}, \ldots, u_{k-n+1}\right) \qquad (13)$$

If the *zero dynamics* [21], [20] is asymptotically stable, then there exists a control law,

$$u_k^* = g\left(y_{k+r}^d, y_k, \ldots, y_{k-n+1}, u_{k-1}, \ldots, u_{k-n+1}\right) \ , \qquad (14)$$

where y_{k+r}^d is the desired plant output at time $k+r$, such that the closed-loop system, $\Sigma_{\mathrm{IO}}(k, y_k, u_k^*)$, satisfies $y_k = y_k^d$ for $k \geq r$.

B. Control And Adaptation Schemes

Since in the case of nonlinear systems the concept of *inverse* is deeply local or bound to a specific class of inputs, this method exploits the recursive adaptation capabilities of the KRLS algorithm, as described in Sec. II-C, to overcome this limitation: this is why this innovative application of regression with kernels wouldn't have been possible with a batch kernel regression algorithm.

The control signal synthesis is carried out concurrently to the controller's adaptation procedure. To identify the plant's inverse $q_k = g(\mathbf{p}_k)$, at each time step, the following data vectors (15) are to be fed to the KRLS algorithm,

$$\begin{aligned} \mathbf{p}_k &= \{y_{k+r}, y_k, \ldots, y_{k-n+1}, u_{k-1}, \ldots, u_{k-n+1}\} \\ q_k &= u_k \ , \end{aligned} \qquad (15)$$

with the set of training input vectors $\mathcal{P} \doteq \{\mathbf{p}_k, k > 0\}$ being a compact subset of a Banach space [22]. Accordingly, it returns the weight vector $\boldsymbol{\alpha}$ and the support vector's dictionary D_k. To synthesize the control signal the input vector

$$\mathbf{x}_k \quad = \left\{y_{k+r}^d, y_k, \ldots, y_{k-n+1}, u_{k-1}, \ldots, u_{k-n+1}\right\} \ , \qquad (16)$$

is fed to the predictor's equation to obtain the wished control input value:

$$u_k = \sum_{i=1}^{d_{k-1}} \alpha_i \kappa\left(\mathbf{x}_i^{dict}, \mathbf{x}_k\right) \ . \qquad (17)$$

We consider the system to be time invariant, this constraint is present since the identification algorithm used - *i.e.*, KRLS - by construction cannot handle abrupt changes in the dynamics [23]. In future works this constraint will be released as algorithms capable of handling time varying dynamics will be used - *i.e.* custom on-line MIMO identification algorithms [24].

C. Design Parameters

As stated above, the tuning parameters are: 1) the kernel parameters. 2) the Approximate Linear Dependency (ALD)'s threshold, δ.

Our approach considers the use of a Gaussian kernel (7), depending only on one parameter, the *kernel width* σ also known as the *characteristic length-scale*. It can be thought of as the distance one has to cover in the input space for the function value to change significantly. Normally, σ can be hand-picked by rule-of-thumb, the stronger the nonlinearities the smaller σ one should chose. Or, in a more formal way through Bayesian inference or cross-validation [24].

For what concerns the ALD's threshold, it must be chosen dependently on the expected performance, normally $\approx 10^{-3}/10^{-4}$.

IV. Numerical Simulations

Here we introduce some simple applications of the method herein presented: the advantages of the method over the cited works are a higher convergence speed, the vast amount of nonlinearities that it can handle and the small tracking error obtained with a simple and straightforward parametrization - the user selectable design parameters being the ALD's threshold, δ, and the kernel parameters (*i.e.*, for a Gaussian kernel, the width σ). Regarding the convergence rate, in Wang [2] the SVR algorithm is pre-trained with data belonging to the surroundings of the operating point so a fair comparison cannot be made; in Plett [1], convergence speed is dependent on the training algorithm used: RTRL converges in 10^7 steps, Dynamic Decoupled Extended Kalman Filter (DDEKF), the fastest method presented [25], needs 10^5 steps. We may also compare to the identification part of Narendra's work [26], the control method used being different and we see that the convergence is achieved in approximately $10^4/10^5$ steps, depending on the system. For more details, the interested reader can refer to the above mentioned literature.

A. First Order (as seen in [27])

As a first example consider the first-order nonlinear non-affine dynamics

$$y_{k+1} = \sin(y_k) + (5 + \cos(y_k u_k))u_k \ . \qquad (18)$$

We fix the objective trajectory $y_k^r = 2\sin(2\pi k/50) + 2\sin(2\pi k/100)$. The controller geometry used is shown in the upper part of Fig. 1. The controller was implemented training a KRLS algorithm to identify the function $u_k = \eta(y_{k+1}^d, y_k)$. In Fig. 2 we can see the result for $\sigma = 1$ and $\nu = 0.0001$. Eventually, at instant $k = 300$ the dictionary size is 58, see Fig. 6. The squared tracking error is shown in Fig. 5, it starts decreasing monotonically after about 100 steps, that is, when the algorithm has converged: this newborn approach to AIC is well promising.

Freed the constraint of an off-line training stage, it's characterized by its ease of use and straightforward implementation. While assuring similar performances it is at least 2 orders of magnitude faster than the cited methods.

B. 2nd Order (as seen in [27])

The plant is described by

$$\begin{aligned} x_1(k+1) &= 0.1x_1(k) + 2\tfrac{u(k)+x_2(k)}{1+(u(k)+x_2(k))^2} \\ x_2(k+1) &= 0.1x_2(k) + u(k)(2 + \tfrac{u(k)^2}{1+x_1(k)^2+x_2(k)^2}) \\ y(k) &= x_1(k) + x_2(k) \ . \end{aligned} \qquad (19)$$

While the representation given is not in the form of (12), it's easy to verify that the system satisfies the conditions given in Sec. III. We want to track $y^r_k = 4\sin(2\pi k/50) + 4\sin(2\pi k/100)$. The controller implements the function $u_k = \eta(y^d_{k+1}, y_k, y_{k-1}, u_{k-1})$. In Fig. 3 we can see the output for $\sigma = 4.3$ and $\nu = 0.0001$. Eventually, at instant $k = 400$ the dictionary size is 120, see Fig. 6, the algorithm fully converges after 205 steps. The squared tracking error is shown in Fig. 5.

C. Model Reference (as seen in [26])

The plant is described by

$$y_{k+1} = \frac{y_k y_{k-1}(y_k + 2.5)}{1 + y_k^2 + y_{k-1}^2} + u_k \tag{20}$$

Given $y^r_k = \sin(2\pi k/50)$, the signal to be tracked is the output of the system

$$y^m_{k+1} = 0.6 y^m_k + 0.2 y^m_{k-1} + p_k \ , \tag{21}$$

as in Fig. 1. The controller implements $u_k = \eta(y^d_{k+1}, y_k, y_{k-1}, y_{k-2}, u_{k-1})$. In Fig. 4 we can see the result for $\sigma = 4$ and $\nu = 0.001$. Eventually, at instant $k = 250$ the dictionary size is 82, see Fig. 6. The squared tracking error is shown in Fig. 5.

D. Additive Measurement Noise

Even if this particular setup isn't meant for noisy measurements, it's interesting to present, without any modification in the identification/control mechanisms, its performance in case of measurement noise. We consider the case of additive measurement noise with zero mean and standard deviation $\sigma_{noise} = 0.2$.

To solve the tracking problem with the first order system of Sec IV-A, the following parameters where chosen: $\sigma = 15.3$ and $\nu = 0.0001$. Full convergence is obtained after 4×10^2 steps, while the dictionary size grows up to $d = 110$ and the Mean Squared Error calculated on the next 1000 samples is 0.1788. For the second order system of Sec IV-B we fixed $\sigma = 14.65$ and $\nu = 0.0005$: convergence is achieved in 5×10^2 steps, $d = 120$ and the Mean Squared Error is 0.09. For the model reference problem of Sec. IV-C: $\sigma = 19.5$ and $\nu = 0.00001$: the system converges in 4.5×10^2 steps, $d = 470$ and the Mean Squared Error is 0.0818. The time evolution of dictionaries is shown in Fig. 7.

Thanks to noise, the space of possible datasets, $\mathcal{P}$, that the one could observe gets expanded leading to bigger dictionaries, higher risk of overfitting, etc. Considering that the presented method wasn't designed to be able to deal with this kind of problems, its performance result to be nonetheless satisfactory.

V. Conclusions

In this paper a new kind of numerical adaptive inverse control scheme was proposed making use of the KRLS algorithm for trajectory tracking. This is just a first attempt over an approach which needs to be theoretically developed and, at this stage, is mainly motivated by the encouraging simulation results. Moreover, this method can be easily adapted to verify if an arbitrary linear/nonlinear system can track a given path p_k in finite time just by tuning a single parameter (i.d., in the case of a RBF kernel only σ).

Future works will develop a theoretical framework to support the numerical results obtained herein.

VI. Acronyms

AIC	Adaptive Inverse Control
ALD	Approximate Linear Dependency
ANN	Artificial Neural Network
BPTT	Back-Propagation Through Time
BPTM	Back-Propagation Through Plant Model
DDEKF	Dynamic Decoupled Extended Kalman Filter
DEKF	Decoupled Extended Kalman Filter
KRLS	Kernel Recursive Least-Square
I/O	Input-Ouput
LMS	Least-Mean Square
LS	Least Square
RLS	Recursive Least-Square
RT	Representer Theorem
RTRL	Real-Time Recurrent Learning
RKHS	Reproducing Kernel Hilbert Space
SVR	Support Vector Regression
SVM	Support Vector Machine

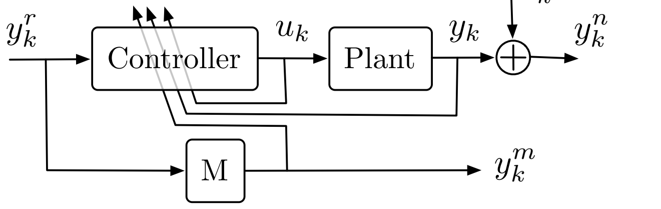

Fig. 1: Typical geometry of a model-reference adaptive inverse control system .

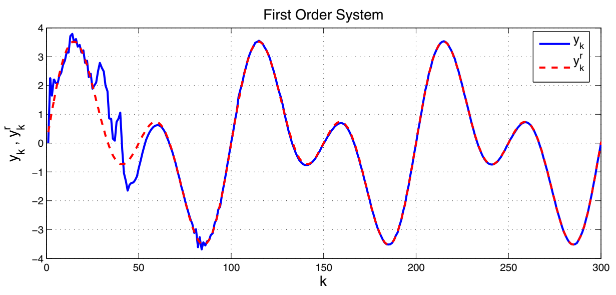

Fig. 2: The first order system (18) is identified and controlled in 100 steps.

References

[1] G. Plett, "Adaptive inverse control of linear and nonlinear systems using dynamic neural networks," *Neural Networks, IEEE Transactions on*, vol. 14, no. 2, pp. 360–376, 2003.

[2] H. Wang, D. Pi, and Y. Sun, "Online svm regression algorithm-based adaptive inverse control," *Neurocomputing*, vol. 70, no. 4-6, pp. 952–959, 2007.

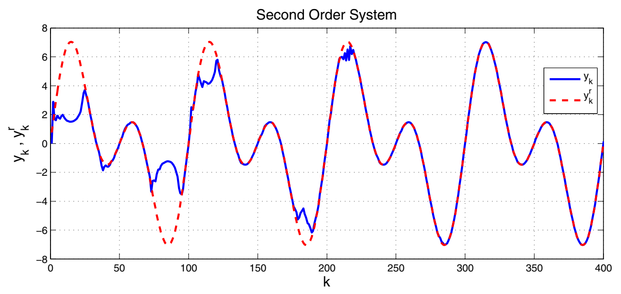

Fig. 3: The second order system (19) tracks the periodic trajectory in less than 120 steps

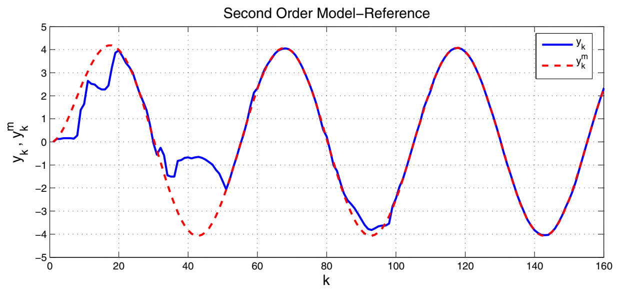

Fig. 4: The system (20) converges towards the output of system (21) in about 250 steps.

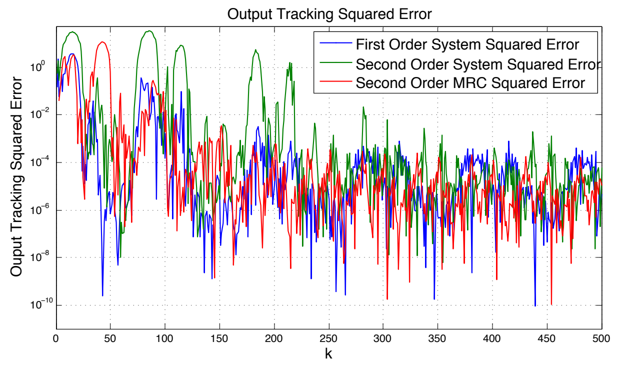

Fig. 5: The squared tracking error e_k^2.

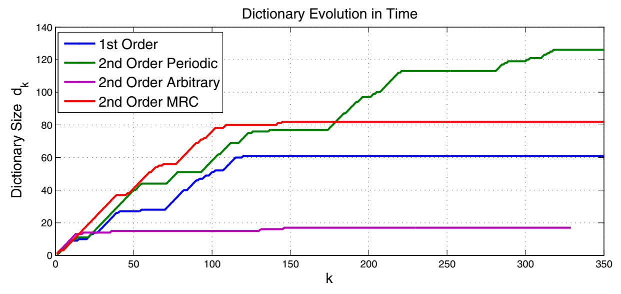

Fig. 6: Time evolution of dictionaries in the noise-free case.

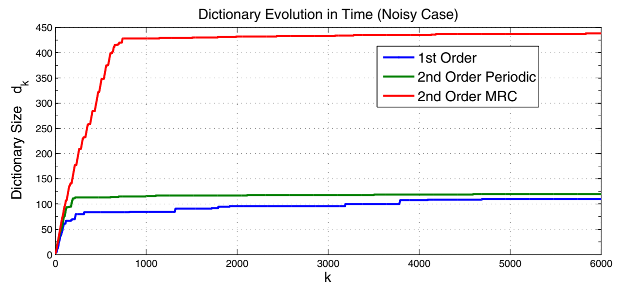

Fig. 7: Time evolution of dictionaries in the noisy case.

[3] W. Rugh, "Research on gain scheduling," DTIC Document, Tech. Rep., 1998.

[4] A. Isidori, *Nonlinear control systems*. Springer Verlag, 1995.

[5] ——, "Nonlinear control systems," 1997.

[6] S. Monaco, D. Normand-Cyrot, and S. Stornelli, "On the linearizing feedback in nonlinear sampled data control schemes," in *Decision and Control, 1986 25th IEEE Conference on*, vol. 25. IEEE, 1986, pp. 2056–2060.

[7] B. Widrow and E. Walach, "Adaptive inverse control: a signal processing approach, reissue edition," 2007.

[8] R. Williams and D. Zipser, "Experimental analysis of the real-time recurrent learning algorithm," *Connection Science*, vol. 1, no. 1, pp. 87–111, 1989.

[9] D. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *Control Systems Magazine, IEEE*, vol. 10, no. 3, pp. 18–23, 1990.

[10] L. Feldkamp and G. Puskorius, "Training controllers for robustness: multi-stream dekf," in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 4. IEEE, 1994, pp. 2377–2382.

[11] V. Vapnik, *The nature of statistical learning theory*. Springer Verlag, 2000.

[12] G. Wahba *et al.*, "Support vector machines, reproducing kernel hilbert spaces and the randomized gacv," 1998.

[13] J. Platt, "A resource-allocating network for function interpolation," *Neural computation*, vol. 3, no. 2, pp. 213–225, 1991.

[14] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *Signal Processing, IEEE Transactions on*, vol. 52, no. 8, pp. 2275–2285, 2004.

[15] J. Cabrera and K. Narendra, "The general tracking problem for discrete-time dynamical systems," in *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, vol. 2, dec 1997, pp. 1451 –1456 vol.2.

[16] A. Sayed, *Fundamentals of adaptive filtering*. Wiley-IEEE Press, 2003.

[17] K. Petersen and M. Pedersen, "The matrix cookbook," *Technical University of Denmark*, pp. 7–15, 2008.

[18] S. Vaerenbergh, "Kernel methods for nonlinear identification, equalization and separation of signals," 2010.

[19] H. Minh, P. Niyogi, and Y. Yao, "Mercers theorem, feature maps, and smoothing," *Learning theory*, pp. 154–168, 2006.

[20] S. Monaco and D. Normand-Cyrot, "Minimum-phase nonlinear discrete-time systems and feedback stabilization," in *Decision and Control, 1987. 26th IEEE Conference on*, vol. 26. IEEE, 1987, pp. 979–986.

[21] ——, "Some remarks on the invertibility of nonlinear discrete-time systems," in *American Control Conference, San Francisco, CA*, 1983, pp. 324–328.

[22] M. Anthony and P. L. Bartlett, *Neural network learning:theoretical foundations*. Cambridge, U.K.: Cambridge University Press, 1999.

[23] W. Liu, I. Park, Y. Wang, and J. Principe, "Extended kernel recursive least squares algorithm," *Signal Processing, IEEE Transactions on*, vol. 57, no. 10, pp. 3801 –3814, oct. 2009.

[24] C. Rasmussen, "Gaussian processes in machine learning," *Advanced Lectures on Machine Learning*, pp. 63–71, 2004.

[25] G. Plett and H. Bottrich, "Ddekf learning for fast nonlinear adaptive inverse control," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 3. IEEE, 2002, pp. 2092–2097.

[26] K. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *Neural Networks, IEEE Transactions on*, vol. 1, no. 1, pp. 4–27, 1990.

[27] K. Narendra and S. Mukhopadhyay, "Adaptive control using neural networks and approximate models," *Neural Networks, IEEE Transactions on*, vol. 8, no. 3, pp. 475–485, 1997.