Incremental Learning of Robot Dynamics using Random Features

Arjan Gijsberts and Giorgio Metta

Abstract— **Analytical models for robot dynamics often perform suboptimally in practice due to various non-linearities and the difficulty of accurately estimating the dynamic parameters. Machine learning techniques are less sensitive to these problems and therefore they are an interesting alternative for modeling robot dynamics. We propose a learning method that combines a least squares algorithm with a non-linear feature mapping and an efficient update rule. Using data from five different robots, we show that the method can accurately model manipulator dynamics, either when trained in batch or incrementally. Furthermore, the update time and memory usage of the method are bounded, allowing the method to be used in real-time control loops.**

I. INTRODUCTION

Dynamic modeling plays a significant role in various robotic applications, including computed torque control, zero-gravity control, and contact detection. Commonly, the (inverse) dynamics are modeled analytically using

$$\tau = \mathbf{D}\left(\mathbf{q}\right)\ddot{\mathbf{q}} + \mathbf{C}\left(\mathbf{q}, \dot{\mathbf{q}}\right)\ddot{\mathbf{q}} + \mathbf{g}\left(\mathbf{q}\right) \quad,$$

where $\mathbf{D}$, $\mathbf{C}$ and $\mathbf{g}$ are the inertial, Coriolis, and gravity terms, respectively [1]. Despite being analytically correct, this formulation has limited applicability on real-life robots, due to the difficulty of accurately determining the various kinematic and dynamic parameters. More importantly, the model is idealized and does not capture non-linearities due to friction, elasticity, flexibility, and vibrations.

Machine learning forms a viable alternative for modeling robot dynamics. In recent years, multiple studies have shown that learning techniques often outperform analytical models of the rigid body dynamics [2], [3]. Typically, these studies employ a batch learning setting, in which the model is trained offline prior to deployment on the robot. Batch learning methods assume i.i.d. sampling of the training samples, which conflicts with the nature of the dynamics problem. This is because samples are obtained sequentially from the robot at high sampling frequencies, causing consecutive samples to be strongly dependent. A large body of training samples is therefore required to ensure that the probabilistic distributions of both training and subsequent test data coincide sufficiently. Furthermore, batch methods are static after the initial training phase and may perform suboptimally in

case of concept drift (e.g., sensor drift, mechanical wear, or temperature fluctuations).

Incremental learning methods, on the other hand, adapt their model continuously during online operation, therefore having a clear advantage over batch methods for sequential learning. Locally Weighted Projection Regression (LWPR) has been used extensively in robotics for incremental learning [4]. Unfortunately, LWPR usually does not perform as well as state of the art batch methods in terms of prediction accuracy. Moreover, its large number of hyperparameters are notoriously hard to tune, making it difficult to obtain optimal performance. Nguyen-Tuong et al. propose Local Gaussian Processes (LGP) to alleviate these problems by combining the local learning approach of LWPR with Gaussian Process Regression (GPR) [5], [6]. Though being efficient in a general sense, neither of these two incremental methods gives strict, theoretical guarantees on timing requirements, rendering them unsuitable for (hard) real-time control loops.

In this paper, we propose an incremental algorithm based on the well-known Regularized Least Squares (RLS) algorithm [7], extended with a finite-dimensional kernel approximation to allow use on non-linear problems. The particular advantages over existing methods are (1) a constant time and space complexity for both predictions and model updates, (2) a tunable parameter that trades computational requirements for prediction accuracy, and (3) ease of implementation and use. We verify experimentally that the generalization performance of the method is at least comparable to existing methods, and that timing requirements are significantly lower for large scale problems.

The Regularized Least Squares method is explained in Section II, including the feature mapping for non-linearity and an incremental update routine for use in sequential learning. In Section III, we describe the setup used for our experiments, after which the corresponding results are covered in Section IV. Finally, we conclude the paper in Section V.

II. INCREMENTAL REGULARIZED LEAST SQUARES WITH RANDOM FEATURES

Real-time modeling of robotic dynamics requires a learning method that is non-linear and capable of performing fast predictions. Additionally, we restrict ourselves to methods that can learn incrementally during online operation of the robot. Our proposed algorithm satisfies these requirements by combining traditional RLS with two additional extensions. First, a random feature mapping is used to approximate shift-invariant kernels in a finite number of dimensions. As a

result, we obtain the non-linearity of kernels, while avoiding linear dependence on the number of training samples. Inspired by earlier work in signal processing, the second extension is an efficient update rule that allows the model to be computed incrementally.

A. Regularized Least Squares

Let us define a set of m samples $\mathcal{S} = \{\mathbf{x}_i, y_i\}_{i=1}^m$, where $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^n$ is the i-th n-dimensional input vector and $y_i \in \mathcal{Y} \subseteq \mathbb{R}$ the associated output label. We consider an algorithm that constructs a linear hyperplane

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \mathbf{x} \ , \tag{1}$$

where $\mathbf{w}$ is the weight vector. In linear Regularized Least Squares (RLS, also known as ridge regression [7]), weight vector $\mathbf{w}$ is chosen such that both the model complexity and the errors are minimized, i.e.,

$$\min_{\mathbf{w}} J(\mathbf{w}, \lambda) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \ , \tag{2}$$

where $\lambda > 0$ is a regularization parameter that balances the tradeoff between minimizing both terms. Additionally, $\mathbf{X}$ is an $m \times n$ matrix defined as $[\mathbf{x}_1, ..., \mathbf{x}_m]^{\mathrm{T}}$ and analogously $\mathbf{y} = [y_1, ..., y_m]^{\mathrm{T}}$. Setting the gradient of (2) to zero, we find that the optimal $\mathbf{w}$ is given by

$$\mathbf{w} = \left(\lambda \mathbf{I} + \mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \mathbf{X}^{\mathrm{T}} \mathbf{y} \ . \tag{3}$$

Computing the optimal solution thus requires solving a system of n linear equations, and the time and space complexity of linear RLS are $\mathcal{O}\left(n^3 + mn^2\right)$ and $\mathcal{O}\left(n^2 + mn\right)$, respectively. Sample predictions are only in $\mathcal{O}(n)$ and thus independent of the number of training samples.

B. Kernel Regularized Least Squares

Practical use of RLS is limited due to its linearity. This limitation is circumvented in Kernel RLS (KRLS) by applying the kernel-trick to map samples implicitly into a (typically non-linear) feature space [8], [9]. A consequence of the kernel machinery, however, is that the prediction function is written as a kernel expansion

$$f(\mathbf{x}) = \sum_{i=1}^m c_i k(\mathbf{x}, \mathbf{x}_i) \ , \tag{4}$$

where $k(\cdot, \cdot)$ is an admissible kernel function [10]. Defining kernel matrix $\mathbf{K} = [k(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^m$, we can compute the optimal m-dimensional vector of coefficients $\mathbf{c}$ as

$$\mathbf{c} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y} \ .$$

Contrary to linear RLS, training KRLS equates to solving a system of m linear equations, yielding an overall time complexity of $\mathcal{O}\left(m^3 + nm^2\right)$. Due to the kernel expansion in (4), the prediction function of KRLS has become linearly dependent on the number of training samples. This scaling behavior is computationally prohibitive when $n \ll m$, as is the case when learning robot dynamics.

C. Approximating the Kernel with Random Features

The dependence on the number of training samples can be avoided by limiting the growth of the kernel expansion. Possible ways of achieving this include discarding old samples once a given budget has been reached [11], projecting linear dependent samples on the current support set [12], or dividing the global model into multiple local models [6]. Another approach is to approximate the kernel using a finite number of basis functions. Bochner's theorem states that a continuous shift-invariant kernel $k(\mathbf{x} - \mathbf{y})$ on $\mathbb{R}^d$ is positive definite if and only if k is the Fourier transform of a non-negative measure $\mu(\omega)$ [13]. Rahimi and Recht use this theorem to find a random projection that gives an unbiased estimate of a shift-invariant kernel, that is

$$k(\mathbf{x}, \mathbf{y}) = \int_{\mathbb{R}^d} e^{-i\omega^{\mathrm{T}}(\mathbf{x}-\mathbf{y})} d\mu(\omega) = \mathbb{E}\left[z_\omega(\mathbf{x}) z_\omega(\mathbf{y})\right] \ , \tag{5}$$

where $z_\omega(\mathbf{x}) = \sqrt{2}\cos\left(\omega^{\mathrm{T}}\mathbf{x} + b\right)$[1], ω is drawn from μ, and b is drawn from a uniform distribution from 0 to 2π [14]. In the remainder of this text, we will focus on the Radial Basis Function (RBF) kernel

$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2} \quad \text{for } \gamma > 0 \ ,$$

for which the corresponding measure μ is a normal distribution with covariance $2\gamma\mathbf{I}$.

The variance of the approximation can be lowered arbitrarily using a vector of D randomly chosen projections normalized by $1/\sqrt{D}$. In order to non-linearize RLS with these random features, it suffices to replace each input vector $\mathbf{x}$ with its random projection $\mathbf{z} = \frac{1}{\sqrt{D}}[z_1(\mathbf{x}), \dots, z_D(\mathbf{x})]^{\mathrm{T}}$ in (1) and (3). Rahimi and Recht show that RLS with random features (RFRLS) converges to KRLS as $\mathcal{O}\left(\frac{1}{\sqrt{D}}\right)$ [15]. Increasing D will therefore give a more accurate approximation, at the cost of computational efficiency.

D. Solving Regularized Least Squares Incrementally

Besides applications in the fields of machine learning, optimization, and statistics, the RLS algorithm has also been used as an adaptive filter in the field of signal processing. This particular use has led to the development of a wide array of efficient update rules, collectively known as Recursive Least Squares methods[2]. Commonly, updating a solution with a new sample only requires $\mathcal{O}\left(n^2\right)$, as compared to $\mathcal{O}\left(n^3\right)$ needed to completely recompute the solution. The effect on space complexity is even more profound, as the incremental methods do not require previous samples to be stored in memory.

The easiest method to incrementally update an RLS solution is to perform rank-1 updates on the inverse of the covariance matrix $\left(\lambda\mathbf{I} + \mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}$ (cf. (3)) using the Sherman–Morrison formula. However, this update formula is known to be sensitive to rounding errors [16]. A numerically much more stable alternative is to update the Cholesky factor

---

[1]There are other equivalent projections that satisfy (5).

[2]The terminology *recursive* is common-place in the field of signal processing. In this paper, however, we will consistently use *incremental*.

Algorithm 1 Incremental RLS with RBF Random Features

Require: $\lambda > 0, \gamma > 0, n > 0, D > 0$

1: $\mathbf{R} \leftarrow \frac{1}{\sqrt{\lambda}}\mathbf{I}_{D \times D}$
2: $\mathbf{w} \leftarrow \mathbf{0}_{D \times 1}$
3: $\beta \leftarrow \mathbf{0}_{D \times 1}$
4: $\mathbf{\Omega} \sim \mathcal{N}\left(0, 2\gamma\right)_{D \times n}$
5: $\mathbf{b} \sim \mathcal{U}\left(0, 2\pi\right)_{D \times 1}$
6: **for all** $(\mathbf{x}, y)$ **do**
7:     $\mathbf{z} \leftarrow \sqrt{\frac{2}{D}}\cos\left(\mathbf{\Omega x} + \mathbf{b}\right)$ // *random feature mapping*
8:     $\hat{y} \leftarrow \mathbf{w}^{\mathrm{T}}\mathbf{z}$ // *predict label*
9:     $\beta \leftarrow \beta + \mathbf{z}y$
10:    $\mathbf{R} \leftarrow \textsc{CholeskyUpdate}\left(\mathbf{R}, \mathbf{x}\right)$
11:    $\mathbf{w} \leftarrow \textsc{CholeskySolve}\left(\mathbf{R}, \beta\right)$ // *update weights*
12:    **yield** $\hat{y}$
13: **end for**

$\mathbf{R}$ of the covariance matrix instead, such that $\mathbf{RR}^{\mathrm{T}} = \left(\lambda\mathbf{I} + \mathbf{X}^{\mathrm{T}}\mathbf{X}\right)$. This update strategy is known as the QR algorithm in the field of adaptive filtering [17]. Rank-1 updates of the Cholesky factor can be computed efficiently using a sequence of Givens rotations [16], whereas the weight vector $\mathbf{w}$ can be obtained using back substitution. We want to emphasize that the solution obtained using incremental updates is identical to the batch solution on the same samples.

The incremental update procedure can trivially be combined with RFRLS. An interesting perspective on the resulting algorithm is to consider it an approximation of Kernel Recursive Least Squares [18], the main difference being that the latter algorithm projects linear dependent samples in order to limit the growth of the kernel expansion. Algorithm 1 summarizes our proposed algorithm for an approximated RBF kernel. Note that the algorithm can be used with other shift invariant kernels by sampling from the corresponding probability distribution in line 4. A fundamental observation is that the $\mathcal{O}\left(D^2\right)$ time and space complexity of incremental RFRLS is constant with respect to the number of training samples, allowing us to give strict real-time guarantees on computation time and memory storage.

Remark 1: We can observe in (2) that the level of regularization decreases with an increasing number of training samples. Regularization is primarily needed in the initial stage of the algorithm, when $\mathbf{X}$ is not full column-rank. Subsequent rank-1 updates increase the smallest singular value of $\mathbf{R}$ [16], therefore reducing the need for regularization in later stages.

III. Experimental Setup

Several experiments have been conducted to evaluate both batch and incremental variants of RFRLS for modeling inverse dynamics of a range of distinct robot manipulators. Prior to presenting the results, we describe the datasets that were used for the experiments, as well as preprocessing and hyperparameter optimization procedures.

TABLE I
Datasets used for the experiments.

	#joints	output	#train	#test
Simulated Sarcos	7	$\tau \times 7$	14904	5520
Sarcos	7	$\tau \times 7$	13922	5569
Barrett	7	$\tau \times 7$	13572	5000
James	4	$[F, \tau]_{x,y,z}$	15000	195977
iCub	4	$[F, \tau]_{x,y,z}$	15000	72850

A. Datasets

A total of five different datasets was used in our study, divided in two groups based on manipulator configuration (Table I). The first group is composed of robot arms with 7 degrees of freedom, with a torque sensor placed in each individual joint. The datasets for this category are taken from a Simulated Sarcos arm, a real Sarcos arm, and a Barrett WAM, and have been used previously in the context of learning dynamics [3], [6]. The manipulators were programmed to perform a periodic motion, during which samples were collected at a rate of approximately 500Hz. Training and test sets were obtained by subsampling the resulting dataset at regular intervals.

The second category contains robot arms of 4 degrees of freedom[3] that have a single force/torque sensor mounted in the upper arm, just below the shoulder joints. These datasets were obtained from the upper torso humanoid James [19] and the full body iCub humanoid [20]. Fumagalli et al. have used the James dataset previously for a comparison between learning methods and an analytical model of the rigid body dynamics [2]. For both datasets[4], the end effector was moved to random Cartesian coordinates in the robot's workspace at constant intervals, while samples were collected at a frequency of roughly 50Hz. In order to investigate predictive behavior over a longer period of time, we restrict the training set to the first 15000 samples.

B. Preprocessing and Hyperparameter Selection

Data preprocessing and hyperparameter selection often has a profound effect on the performance of learning methods. For the dynamics datasets, it is crucial to properly scale the input features when using an (isotropic) RBF kernel, as to avoid large acceleration features to dominate lower-valued position features. In our setup, all input features are therefore divided by a characteristic length scale $\ell_i > 0$ for $1 \leq i \leq n$. Optimizing these length scales using grid search is computationally infeasible due to its exponential scaling behavior. Instead, we exploit the similarity between GPR and KRLS, and therefore indirectly also RFRLS. In GPR, hyperparameters can be optimized analytically by maximizing the marginal likelihood [5]. We use this method to optimize the hyperparameters of GPR trained on a random subset of 2000 training samples using an anisotropic RBF

---

[3]We limit ourselves to the shoulder and elbow joints; the actual robot arms have more than 4 degrees of freedom.
[4]These datasets can be obtained by contacting the authors.

kernel. The formulation of this kernel in the context of GPR can be rewritten as

$$k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \sigma_f^2 \, e^{\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j)\right)} + \delta_{ij}\sigma_n^2 \ ,$$

where δ_{ij} is the Kronecker delta, σ_n^2 and σ_f^2 respectively the noise and signal variance, and $\mathbf{M}$ an $n \times n$ scaling matrix with $\left[\ell_1^{-2}, \ldots, \ell_n^{-2}\right]$ on its diagonal. Furthermore, the locally optimal hyperparameters can be converted to KRLS and RFRLS with isotropic RBF kernel by scaling each i-th input feature by ℓ_i^{-1} for $1 \le i \le n$, and subsequently setting $\lambda = \sigma_n^2/\sigma_f^2$ and $\gamma = \frac{1}{2}$.

The hyperparameters have to be optimized for each output separately. Preliminary experiments, however, showed that all outputs share similar dependencies on hyperparameters: configurations that perform well for one output are also likely to perform well on the other outputs. The computational cost of our method can therefore be reduced significantly by sharing a hyperparameter configuration for all outputs. In this case, we compute a single kernel or covariance matrix, which is subsequently used to solve multiple weight vectors concurrently (i.e., one for each output). We can thus solve for multiple outputs at negligible additional cost. For RFRLS, the gain in computation time could be used to improve prediction accuracy by increasing the number of random features. From the optimal hyperparameter configurations for each output, we select the configuration with the lowest mean error as measured using cross validation on the remaining training samples. Identical configurations were used both for KRLS and RFRLS.

IV. RESULTS

The experimental results are divided in two sets. In the first set of experiments, RFRLS is evaluated terms of prediction errors and scalability in a batch setting. Results on these datasets will give insight in how well our method performs with respect to other state of the art learning methods. The second set of experiments aims to verify the benefits of incremental learning over batch learning on two large scale datasets. For these experiments, we compare batch KRLS with incremental RFRLS.

A. Batch Learning

The Simulated Sarcos, Sarcos, and Barrett datasets have been used previously for learning experiments (e.g., [3], [6]), and therefore form an excellent benchmark setting for RFRLS. In particular, we focus on the results for LWPR, GPR, and LGP[5] as presented by Nguyen-Tuong et al. [6]. Identical experiments were performed with batch RFRLS using 500, 1000, and 2000 random features (i.e., parameter D). Additionally, we included KRLS as a reference method, since it is an (approximate) upper bound on the performance of RFRLS. A minor deviation from the experimental setup, as described in Section III-B, is that for RFRLS[500] on the Barrett dataset we report the results using optimal hyperparameter configurations for each distinct output. With

this limited number of random features, results using a shared hyperparameter configuration for all outputs were unsatisfactory.

The results in Fig. 1 show that, contrary to our expectations, KRLS often outperforms GPR by a significant margin. These two methods are functionally identical and can be expected to show similar performance. These deviations indicate that different hyperparameter configurations were used, which can be explained by the non-convexity of the marginal likelihood optimization problem (cf. Section III-B). Hence, we have to be cautious when interpreting the comparative results on these datasets with respect generalization performance. The comparison between KRLS and RFRLS, trained using identical hyperparameters, remains valid and gives an indication of the approximation quality of RFRLS. As expected, the performance of RFRLS steadily improves as the number of random features increases for all datasets. Furthermore, RFRLS[1000] is often sufficient to obtain satisfactory predictions on all datasets. RFRLS[500], on the other hand, performs poorly on the Barrett dataset, despite using distinct hyperparameter configurations for each degree of freedom. In this case, RFRLS[1000] with a shared hyperparameter configuration is more accurate and requires less overall time for prediction.

Fig. 2 shows the prediction times for all 7 outputs. Note that the timing results for GPR, LWPR, and LGP were measured on different hardware, and consequently may vary up to a constant factor when compared to our timing results. Nevertheless, Fig. 2 shows that the batch methods (i.e., GPR and KRLS) scale linearly with the number of training samples. Both LWPR and LGP scale sublinearly, though still outperformed by RFRLS due to its constant time complexity. When combined with the prediction results from Fig. 1, we can confirm that parameter D effectively trades computation for prediction accuracy. This property is particularly useful for real-time control, as it allows to maximize accuracy given strict timing constraints.

Remark 2: The low prediction errors of KRLS in general are due to training and test sets being subsampled interleavingly from the same motion, and they are therefore strongly correlated. Combined with dense sampling, it is inevitable that there is a similar training sample for each test sample. To investigate this issue further, we performed additional experiments using Kernel Nearest Neighbor (1-KNN) [21]. This method simply returns as predicted output the label associated with the most similar training sample and therefore lacks generalization ability. Consequently, 1-KNN is critically dependent on congruence between training and test distributions, dense sampling, and noise-free outputs. Also 1-KNN, when used with the same kernel as KRLS, outperforms previously published results in nearly all cases. This confirms that the similarity between training and test samples is very high, given correct scaling of the input features, and small predictions errors therefore do not necessarily indicate good generalization performance.

---

[5]Published results for other methods (e.g., ν-SVR) are comparable to the considered methods and have been left out for conciseness.
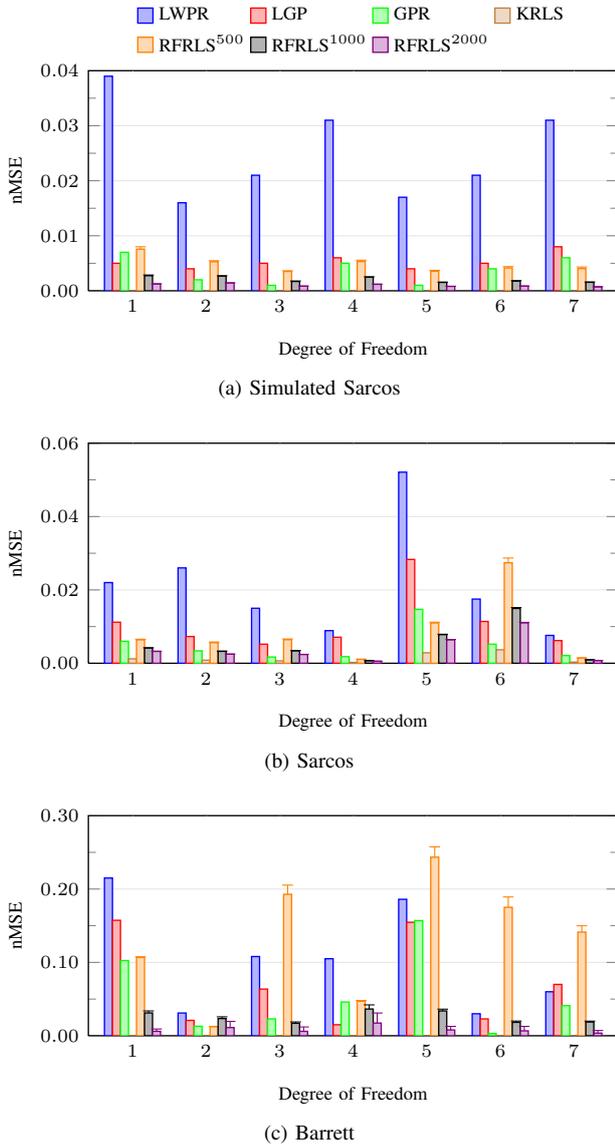
Fig. 1. Prediction error per degree of freedom for the (a) Simulated Sarcos, (b) Sarcos, and (c) Barrett datasets. The results for LWPR, GPR, and LGP are taken from Nguyen-Tuong et al. (cf. [6]). The mean error over 25 runs is reported for RFRLS with $D \in \{500, 1000, 2000\}$, whereas error bars mark a distance of one standard deviation. Note that in some cases the prediction errors for KRLS are very close to zero and therefore barely noticeable.

B. Incremental Learning

A primary advantage of incremental learning over the more common batch approach is the ability to adapt continuously to changing conditions. This is particularly important if the i.i.d. sampling assumption is violated, which is often the case for real-life (robotic) problems. We therefore compare incremental learning with the more common batch learning paradigm on the James and iCub datasets. For the batch case, we trained a KRLS with RBF kernel on the initial 15000 training samples. This number of training samples is sufficient to learn the dynamics model when sampled i.i.d. (cf. [2]). We compare this method with incremental RFRLS as detailed in Algorithm 1, exclusively using the training samples for hyperparameter optimization (cf. Section III-
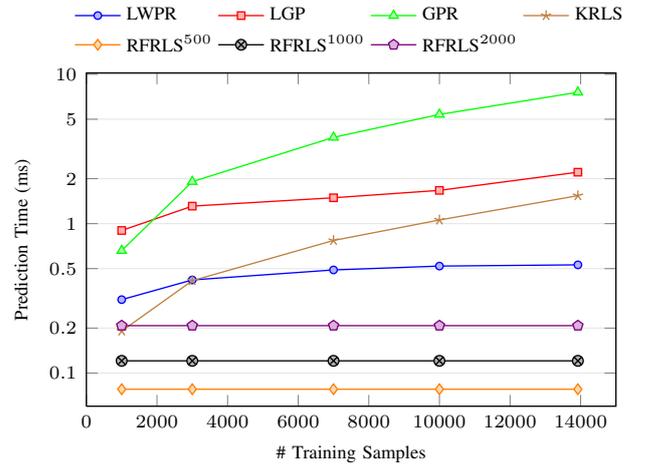

Fig. 2. Prediction time for all seven output labels with respect to the number of training samples. The results for LWPR, GPR, and LGP are taken from Nguyen-Tuong et al. (cf. [6]) and could vary by a constant factor as compared to KRLS and RFRLS.

B). During testing, analogous to real-life employment of the learning method, the incremental method thus starts without any learned knowledge.

Fig. 3 shows how the average nMSE develops as test samples are predicted in sequential order using both methods. RFRLS, starting without any prior training, requires between 5000 and 10000 samples to achieve performance comparable to KRLS. The performance of KRLS, on the other hand, decreases over time. In particular on the iCub dataset it suffers a number of large errors, causing the average nMSE to show sudden jumps. This is a direct consequence of the unavoidable fact that train and test samples are not drawn *independently* from the same distribution. Incremental RFRLS, on the other hand, is largely unaffected by these changes and demonstrates stable predictive performance. This is not surprising, as (1) it is able to adapt to changing conditions, and (2) it eventually has trained on significantly more samples than KRLS.

Furthermore, Fig. 3 shows that 200 random features are sufficient to achieve satisfactory performance on either dataset. In this case, model updates of RFRLS require only 400μs, as compared to 2ms and 7ms when using 500 or 1000 random features, respectively. These timing figures make incremental RFRLS suitable for high frequency control loops.

V. CONCLUSIONS

Machine learning is an interesting alternative to modeling robot dynamics analytically, as it avoids the need to estimate kinematic and dynamic parameters and can handle non-linearities inherent to real robotic systems. In this paper, we propose the RFRLS algorithm to learn dynamics models incrementally. An experimental comparison with state of the art batch and incremental methods shows that RFRLS is highly competitive in terms of prediction accuracy. A primary advantage of our method compared to other incre-
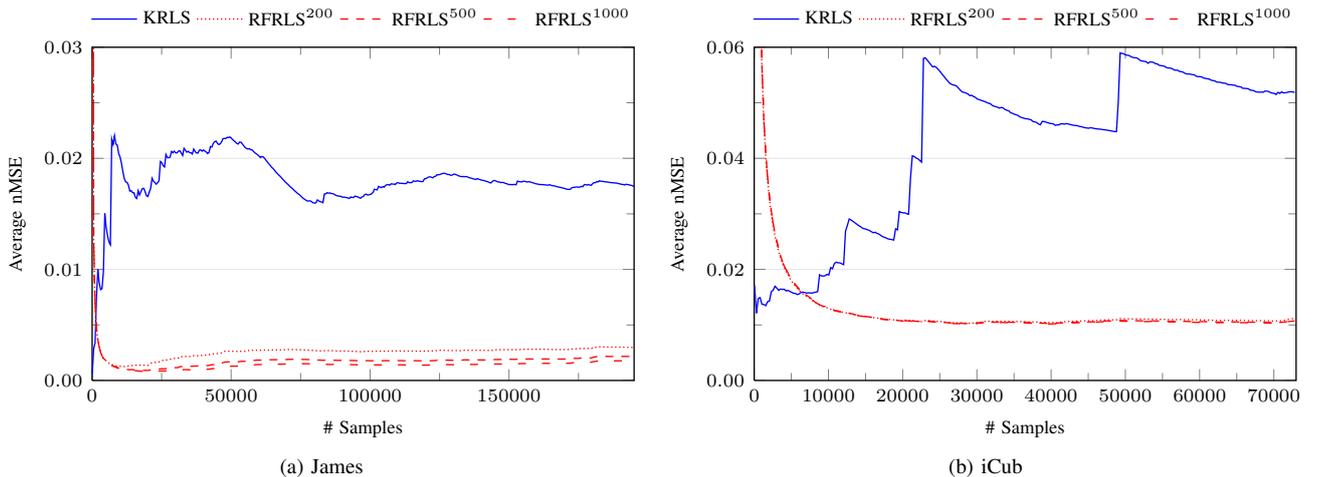
Fig. 3. Average prediction error with respect to the number of test samples of KRLS and incremental RFRLS with $D \in \{200, 500, 1000\}$ on the (a) James and (b) iCub datasets. The error is measured as the nMSE averaged over the force and torque output components. The standard deviation over 25 runs of RFRLS is negligible in all cases, for clarity we report only the mean without error bars.

mental methods, such as LWPR or LGP, is the constant update complexity with respect to the number of samples. Furthermore, the tradeoff between accuracy and computational cost can be regulated using a single parameter. These properties make RFRLS particularly suited for use in control loops with strict timing requirements.

In addition, we demonstrate the advantage of incremental learning over traditional batch learning for modeling robot dynamics. In this problem, consecutive samples are strongly dependent and there may additionally be concept drift. This lack of i.i.d. sampling is particularly problematic for batch learners, as past training samples may not be representative for future test data. Incremental methods, on the other hand, alleviate this problem by learning from all available samples. The resulting training data (i.e., the past test data) is therefore more likely to be representative for future samples. Experiments on two realistic large scale datasets show that prediction errors of incremental RFRLS are indeed significantly lower as compared to batch KRLS.

ACKNOWLEDGMENTS

REFERENCES

[1] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modelling and Control*. John Wiley & Sons, New York, USA, 2005.
[2] M. Fumagalli, A. Gijsberts, S. Ivaldi, L. Jamone, G. Metta, L. Natale, F. Nori, and G. Sandini, "Learning to exploit proximal force sensing: A comparison approach," in *From Motor Learning to Interaction Learning in Robots*, 2010, pp. 149–167.
[3] D. Nguyen-Tuong, J. Peters, and M. Seeger, "Computed torque control with nonparametric regression models," in *Proceedings of the 2008 American Control Conference (ACC 2008)*, June 2008, pp. 212–217.
[4] S. Vijayakumar, A. D'souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural Computation*, vol. 17, no. 12, pp. 2602–2634, 2005.
[5] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2005.
[6] D. Nguyen-Tuong, M. W. Seeger, and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.
[7] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, pp. 55–67, 1970.
[8] R. Rifkin, G. Yeo, and T. Poggio, "Regularized least squares classification," in *Advances in Learning Theory: Methods, Model and Applications*, vol. 190, 2003, pp. 131–154.
[9] C. Saunders, A. Gammerman, and V. Vovk, "Ridge regression learning algorithm in dual variables," in *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 515–521.
[10] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, June 2004.
[11] O. Dekel, S. Shalev-Shwartz, and Y. Singer, "The forgetron: A kernel-based perceptron on a budget," *SIAM Journal on Computing*, vol. 37, no. 5, pp. 1342–1372, 2008.
[12] Y. Engel, S. Mannor, and R. Meir, "Sparse online greedy support vector regression," in *ECML '02: Proceedings of the 13th European Conference on Machine Learning*, 2002, pp. 84–96.
[13] V. I. Bogachev, *Measure Theory*. Dordrecht, The Netherlands: Springer, 2007.
[14] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in Neural Information Processing Systems 20*, 2008, pp. 1177–1184.
[15] ——, "Uniform approximation of functions with random bases," in *Allerton Conference on Communication Control and Computing (Allerton08)*, September 2008, pp. 555–561.
[16] Å. Björck, *Numerical Methods for Least Squares Problems*. Philadelphia: SIAM, 1996.
[17] A. H. Sayed, *Adaptive Filters*. Wiley-IEEE Press, 2008.
[18] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, pp. 2275–2285, 2003.
[19] L. Jamone, F. Nori, G. Metta, and G. Sandini, "James: A humanoid robot acting over an unstructured world," in *International Conference on Humanoid Robots*, 2006.
[20] N. G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. J. Ijspeert, M. C. Carrozza, and D. G. Caldwell, "iCub: the design and realization of an open humanoid platform for cognitive and neuroscience research," *Advanced Robotics*, vol. 21, no. 10, pp. 1151–1175, 2007.
[21] K. Yu, L. Ji, and X. Zhang, "Kernel nearest-neighbor algorithm," *Neural Processing Letters*, vol. 15, pp. 147–156, 2002.