Advantages and limitations of reservoir computing on model learning for robot control

Polydoros, Athanasios; Nalpantidis, Lazaros; Krüger, Volker

Publication date:
2015

Document Version
Accepted author manuscript, peer reviewed version

Advantages and Limitations of Reservoir Computing on Model Learning for Robot Control

Athanasios S. Polydoros, Lazaros Nalpantidis and Volker Krüger

***Abstract*— In certain cases analytical derivation of physics-based models of robots is difficult or even impossible. A potential workaround is the approximation of robot models from sensor data-streams employing machine learning approaches. In this paper, the inverse dynamics models are learned by employing a learning algorithm, introduced in [1], which is based on reservoir computing in conjunction with self-organized learning and Bayesian inference. The algorithm is evaluated and compared to other state of the art algorithms in terms of generalization ability, convergence and adaptability using five datasets gathered from four robots in order to investigate its pros and cons. Results show that the proposed algorithm can adapt in real-time changes of the inverse dynamics model significantly better than the other state of the art algorithms.**

I. Introduction

The use of models for the representation of robots' embodiment and their interaction with the environment is common in the field of intelligent robotics for action control and prediction [2]. Models can be derived analytically based on the physics and structure of the robot. However, such methods can not cope with changes of the robot structure and dynamic environments. Furthermore, analytical computation of models, especially on low-cost manipulators with elastic actuators, is too difficult or even impossible.

In order to overcome such problems towards the development of adaptive and cognitive robots, models should instead be learned online using streams of sensory data. Model learning can be generally defined as a process where an agent can infer the characteristics of its structure and environment. Thus, data-based model learning algorithms have become popular for being able to accurately model complex robotic systems. Most of the existing approaches can be classified in three classes: Direct Modeling, Indirect Modeling and Distal Teacher Learning [2].

We deal with this problem—learning robot models in real-time through data-streams—by introducing a novel machine learning algorithm, dubbed Principal-Components Echo State Network (PC-ESN). The presented algorithm can be applied within a Direct Modeling scheme, as diagrammatically illustrated in Fig. 1 for learning robot control. A forward model is learned using observed inputs and outputs as training signals, while a feed-back controller is employed for compensating errors.

All the authors are with the Robotics, Vision and Machine Intelligence (RVMI) Lab., Department of Mechanical and Manufacturing Engineering, Aalborg University Copenhagen, Denmark {athapoly,lanalpa,vok}@m-tech.aau.dk

This work has been supported by the European Commission through the research project "Sustainable and Reliable Robotics for Part Handling in Manufacturing Automation (STAMINA)", FP7-ICT-2013-10-610917.

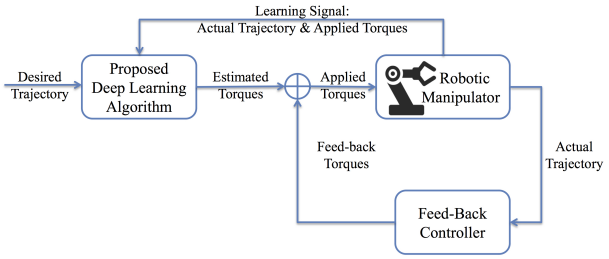


Fig. 1. A direct model for learning inverse dynamics using the proposed algorithm. The desired joints' positions, velocities and accelerations are fed into the algorithm, which provides an estimation of the required torques. Those torques are corrected by the feed-back controller's signal. The feed-back torques are a linear combination of the actual position and velocity of the manipulator, weighted by error constants. The sensors' measurements that derive from the applied torques are used as training signal of the algorithm [1].

As thoroughly presented in [1], the structure of the network consists of four layers: the input, the output, and two hidden layers—a self-organized and a recursive reservoir. The self-organized layer decorrelates the inputs by approximating their principal components using Generalized Hebbian Learning (GHL). The reservoir projects the uncorrelated inputs to high dimensional space and provides a fading memory. The Bayesian linear regression is recursively applied as learning rule for updating the connections between the reservoir and output layer. We argue that the algorithm belongs to the class of deep learning methods since it models a desired mapping using a large set of non-linear transformations of the input data [3] Also, the recursive reservoir is considered to be a deep neural network because, if folded out in time, it corresponds to a feed-forward network with indefinite number of layers [4].

The main *contribution* of this work is to present the advantages and limitations of reservoir computing as a solution to the robot model learning problem. For this purpose we use the results derived in [1] since it is, to the best of the authors knowledge, the first pure reservoir computing algorithm proposed for real-time robot model learning. Our approach extents the applicability of model learning methods to noisy data, obtained by robots without accurate force/torque sensors. Furthermore, our approach can quickly converge to new situations generating the appropriate control signals, due to the fading memory of the reservoir. Thus, it is able to adapt to changes of the environment (e.g. handling different objects, or picking and releasing objects) or the robot itself (e.g. due to mechanical wear). Contrary to other popular machine learning methods, such as kernel-based methods, the presented approach does not require an a priori selection

of kernel or hyperparameter optimization. What makes the proposed algorithm particularly appealing for real-time robot control is that its complexity is independent of the number of training samples, since they are not retained but rather used to recursively update it. Thus, a minimum update frequency of the model, during operation, can be guaranteed. Finally, as an additional contribution, we make publicly available three new datasets for inverse dynamics model learning, captured from two industrial robots.

II. State of the art

A benchmark problem in model learning is the modeling of robotic manipulators inverse dynamics. The inverse dynamics problem involves the computation of the required joints' torques in order to achieve a desired motion (position, velocity, acceleration). The inverse dynamics relationship can be expressed as:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} \tag{1}$$

where $\mathbf{q}, \dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are the joints' angular position, velocity and acceleration respectively. $\mathbf{M}$ is the inertia matrix, $\mathbf{C}$ the centripetal and Coriolis torques. Finally, $\mathbf{G}$ is the effect of the gravity at the system and $\boldsymbol{\tau}$ is the vector of the applied torque command. The disadvantage of such a physics-based model is that parameters, like friction and moments of inertia, are hard to get defined [5]. Two main approaches have been proposed for the derivation of these parameters—the dynamic parameter identification [6] and adaptive control [7].

The derivation of the physics-based dynamics model of (1) is based on assumptions regarding the structure of the robotic manipulator and the type of its joint. Those assumptions do not necessarily hold in the case of light-weight and compliant manipulators. Thus, the necessity of data-driven model learning of the inverse dynamics mapping becomes prominent. In this case, the machine learning algorithm has to approximate a function $f(\cdot)$ such that:

$$f(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) + \boldsymbol{\varepsilon} = \boldsymbol{\tau} \tag{2}$$

where $\boldsymbol{\varepsilon}$ is the noise of the manipulator's system. Thus, the approximation of the inverse dynamics mapping provided by (2) corresponds to a regression problem, which can be solved by a large variety of machine learning algorithms [8]–[15]. Their common characteristic is that they are nonparametric. In this paper, we focus on real-time algorithms that are developed for sequential torque estimation and model adaptation.

Locally Weighted Projection Regression (LWPR), introduced in [8], is a local model which approximates nonlinear mappings in high-dimensional space. Its computational complexity depends linearly on the amount of the training instances. The algorithm copes with the curse of dimensionality by performing a projection regression. A drawback of this approach is the large number of free parameters which are hard to optimize. Furthermore, the authors in [13] introduced prior knowledge in LWPR in order to increase the algorithm's generalization ability.

A large portion of the literature is focused on employing kernel-based methods for the estimation of the inverse dynamics mapping by employing approaches, such as Gaussian Process Regression (GPR) and Support Vector Regression (SVR).

Local Gaussian Process (LGP), introduced in [11], handles the problem of real-time learning by building local models on similar inputs, based on a distance metric and uses the Cholesky decomposition for incrementally updating the kernel matrix.

In [15] the authors propose a real-time algorithm, dubbed SSGPR, which incrementally updates the model using GPR as learning method. The model is capable of learning nonlinear mappings by using random features mapping for kernel approximation whose hyperparameters are automatically updated.

A hybrid algorithm that combines both reservoir computing and GPR is presented in [12]. The search space is reduced by employing an online goal babbling method. The manipulator's state is represented by a recurrent Echo State network, and this state is used as input to the Local GPR algorithm.

In this paper, we use a pure reservoir computing algorithm, as presented in [1], in order to identify its potential compared to other state of the art methods. Contrary to the described kernel-based methods [11], [15], in the reservoir computing algorithm there is no need for kernel selection and hyperparameter optimization. Furthermore, its complexity does not depend on the amount of the training instances, due to the recursive updating rule. Thus, the data-stream is not kept in memory, like in [8], [11], [12]. Also, the learning rule depends only on two parameters, which control the fit of the model on the training data. This attribute makes fine-tuning easier compared to methods with many free parameters such as [8]. Finally, the proposed method is adaptive to changes of the inverse dynamics mapping that occur when the manipulator handles various loads.

III. Overview of the algorithm

The illustration of the deep neural network's structure is presented in Fig. 2. The connections between the input and the self-organized layer are feed-forward and they are updated based on the GHL rule. The self-organized layer is directly connected to the output layer and the reservoir, which consists of a large number of fully interconnected neurons. The connections between the reservoir neurons and the outputs' feed-back connections are constant and derived so as to ensure the Echo-State property of the reservoir. The connections towards the output layer are updated by applying Bayesian Linear Regression. A more detailed presentation of the applied learning algorithm can be found in [1].

A. The self-organized layer

The values of the nodes in the self-organized layer depend on the values of the inputs and of the weights. In the case of inverse dynamics modeling, the inputs are the position, velocity and acceleration of each joint. Thus, the nodes' values s of the self-organized layer are calculated as:

$$\mathbf{s}_{t+1} = g(\mathbf{W}^{in}\mathbf{u}_t) \tag{3}$$

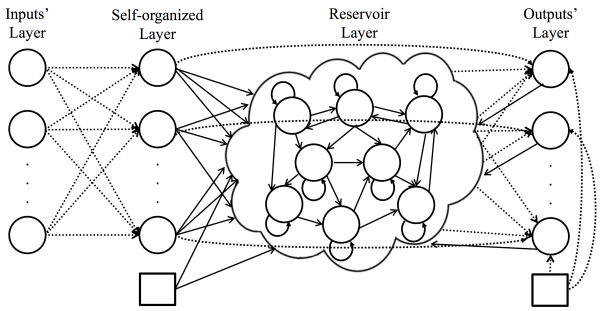


Fig. 2. Structure of the deep neural network. Nodes correspond to neurons and arcs to weights that describe cause-effect relationships between neurons. Solid arcs represent constant weights, contrary to doted arcs that change according to a learning rule. Rectangles represent the bias on the reservoir and output layer.

where $\mathbf{W^{in}}$ is the inputs' weights matrix whose value in entry (s, k) is the weight from the input node k to the self-organized node s and $g(\cdot)$ is the neurons' activation function, the hyperbolic tangent. The inputs are represented by the column vector $\mathbf{u}$ and the neurons of the self-organized layer by the column vector $\mathbf{s}$.

The GHL is an unsupervised learning rule [16] and it is used for the adaptation of the inputs' weights $\mathbf{W^{in}}$ as presented in 4. This learning rule is a generalization of Oja's rule, belongs to the family of Hebbian Learning algorithms.

$$\Delta \mathbf{W}^{in} = \eta_t(\mathbf{u}\mathbf{s}^T - \mathrm{LT}\left[\mathbf{s}\mathbf{s}^T\right]\mathbf{W}^{in}) \tag{4}$$

Thus, GHL yields the entire set of the eigenvectors for a processed data-stream. The operator $\mathrm{LT}[\cdot]$ converts a matrix to lower-triangular and η is the learning step. If the learning rate decreases after each time-step and for infinite time-steps, the rows of an initially random inputs' weights matrix $\mathbf{W^{in}}$, would correspond to the eigenvectors of the inputs' covariance matrix. Furthermore, the GHL rule assumes that the inputs are centered, therefore the inputs' mean value is updated at each time step and subtracted from the input vector. Since the data-stream consists of high-frequency samples of the manipulator's position, velocity and acceleration, the inputs of the algorithm can be significantly correlated. Using such a transformation, the inputs get uncorrelated which results to a better prediction accuracy.

B. The Reservoir

The uncorrelated inputs are fed to the second hidden layer, a Recurrent Neural Network (RNN). This type of network is selected, instead of a feedforward network, because RNNs can approximate dynamical systems [17]. Furthermore, due to the recurrent connections their state represents the history of the inputs, thus they have a dynamic memory. Those characteristics make them powerful tools for time-series prediction and therefore can be applied on real-time dynamics approximation.

Despite their advantages, the application of RNNs, as machine learning tools, was limited because of the computational expensive, gradient-based, update rules. The difficulty of training rises from the recurrent connections between the nodes of the network. Thus, in order to apply training rules such as back-propagation, which is widely used in feedforward networks, the recurrences have to be unfold through time.

This drawback is solved by introducing the dynamic reservoir. A reservoir is a large set of recurrently connected nodes. The connections are fixed, except those from the reservoir towards the output layer (readout connections). The reservoir has two main functions, it expands the uncorrelated signal $\mathbf{s}$ non-linearly to a high-dimensional space and it preserves a memory of that signal. The values of the fixed weights $\mathbf{W}^{res}$, that interconnect the reservoir nodes, are set by following the procedure described in Algorithm 1. This procedure yields a reservoir consisting of nodes with the Echo Sate property [18] regardless how the other, also fixed, weights are set.

The Echo State property has as result that the state $\mathbf{r}$ of the reservoir is unique for each different input sequence $\mathbf{s}_{0\to T}$. This fact, in conjunction with the short term memory embedded in the state of the reservoir and the fixed connections, makes this type of reservoir appropriate for real-time model learning.

The global parameters for setting the Echo State reservoir are the sparsity, the number of nodes and the spectral radius. A large number of reservoir nodes is recommended if there are not enough available training instances [19]. However, this is not the case in model learning of manipulator inverse dynamics, since the inputs can be sampled with frequencies up to 1 KHz. Thus, a good approximation of the inverse dynamics can be achieved with a relatively small number of reservoir nodes.

The sparsity of the reservoir affects the computational time and slightly the performance. The spectral radius α should be less than 1 in order to ensure the Echo State property for any input. Its value depends on the relation between the history of the inputs $\mathbf{s}$ and the target value $\mathbf{u}$. Large values of the spectral radius are recommended when the history of the data-stream is significant for the derivation of torques; otherwise, small values are preferable. Thus, for modeling the inverse dynamics, small spectral radius is preferred.

In the proposed model, the values of the reservoir nodes are updated according to:

$$\mathbf{r}_{t+1} = g\left(\mathbf{W}^{res}\mathbf{r}_t + \mathbf{W}^{self}\mathbf{s}_{t+1} + \mathbf{W}^{fb}\mathbf{o}_t\right) \tag{5}$$

where $\mathbf{r}_{t+1}$ is the state of the reservoir at time step $t+1$, $\mathbf{W}^{self}$ is the weights matrix connecting the nodes of the self-organized layer with the reservoir, $\mathbf{W}^{res}$ are the connections between the nodes of the reservoir and $\mathbf{W}^{fb}$ the feed-back connections from the output $\mathbf{o}_t$ to the reservoir nodes.

The values of the output layer's nodes are a linear combination of the nodes connected with them and are calculated as follows:

$$\mathbf{o}_{t+1} = \mathbf{W}^{out}\mathbf{r}_{t+1} + \mathbf{W}^{dir}\mathbf{s}_{t+1} \tag{6}$$

Where $\mathbf{W}^{out}$ are the weights from the reservoir to the output

and $\mathbf{W}^{dir}$ the weights from the self-organized layer to the output.

Algorithm 1 Pseudocode for creating the reservoir weights matrix

Input: $\mathbf{W}_0$ random sparse matrix, $w_{ij} \sim \mathcal{U}(-1, 1)$
Input: desired spectral radius : $\alpha < 1$
Output: Matrix $\mathbf{W}^{res}$ with the Echo-State Property

$\mathbf{W}_1 \Leftarrow \frac{1}{\lambda_{\max}}\mathbf{W}_0$ ▷ $\lambda_{\max}$ *largest absolute eigenvalue of* $\mathbf{W}_0$
$\mathbf{W}^{res} \Leftarrow \alpha\mathbf{W}_1$ ▷ $\mathbf{W}^{res}$ *has now spectral radius* α

The derivation of the required weights corresponds to a linear regression problem, thus the weights $\mathbf{W}^{out}$ and $\mathbf{W}^{dir}$ are updated by applying Bayesian linear regression. For notation simplicity, all the weights to the output are concatenated in a single matrix $\mathbf{W}^{train}$ and their corresponding nodes values to the vector $\mathbf{c}$. As a result, (6) can be written as:

$$\mathbf{o}_{t+1} = \mathbf{W}^{train}\mathbf{c}_{t+1} \tag{7}$$

C. Recursive Bayesian Linear Regression

The employed learning rule has to optimize the weights $\mathbf{W}^{train}$ so that the output of the algorithm, $\mathbf{o}$, approximates the output of function $f(\cdot)$ in (2). In order to keep the notation simple, the presentation of the learning rule is confined in the case of a single joint but can be easily generalized for all the manipulator's joints. Thus, the weights towards the output node o are represented as a row vector $\mathbf{w}^{train} = \begin{bmatrix} w_{o1} & w_{o2} & \cdots & w_{oN} & w_{bias} \end{bmatrix}$. The inverse dynamics model of a single joint at time step t can be rewritten as:

$$\mathbf{w}_t^{train}\mathbf{c}_t + \varepsilon = \tau_t \tag{8}$$

The regression problem in (8) can be solved using methods such as ordinary least squares estimation and ridge regression. Applying the Bayesian approach to this problem, a probability distribution over the regression coefficients $p\left(\mathbf{w}_t^{train}|D\right)$ is derived instead of a simple point estimation. By assuming Gaussian likelihood and prior distributions, and applying the Bayesian rule on linear Gaussian systems, then the posterior is also a Gaussian distribution with mean $\mathbf{w}_{t+1}^{train}$ and covariance matrix $\mathbf{V}_{t+1}$ which are recursively derived from (9) and (10) respectively by setting the initial weights $\mathbf{w}_0$ to a zero vector and $\mathbf{V}_0 = \phi^2\mathbf{I}$.

$$\mathbf{w}_{t+1}^{train} = \mathbf{V}_{t+1}\mathbf{V}_{t-1}^{-1}\mathbf{w}_t + \frac{1}{\sigma^2}\mathbf{V}_{t+1}\mathbf{c}_t\tau_t \tag{9}$$

$$\mathbf{V}_{t+1} = \left(\mathbf{V}_{t-1}^{-1} + \frac{1}{\sigma^2}\mathbf{c}_t\mathbf{c}_t^T\right)^{-1} \tag{10}$$

Thus, at each time step the weights' covariance matrix is updated based on the nodes' values that are connected with the output node. Since the Bayesian rule is applied recursively, the weights' posterior distribution at time step t is used as prior distribution at time step $t+1$. From (9) and (10) is clear that the algorithm's computational complexity for training depends on the number of nodes that are connected to the output. Therefore, in order to reduce the complexity, a Cholesky decomposition can be applied on the covariance matrix $\mathbf{V}$ for calculating its rank update and inverse.

TABLE I
DESCRIPTION OF THE DATASETS USED FOR EVALUATION (DATASETS MARKED WITH ^{P} ARE PUBLICLY AVAILABLE, WHILE DATASETS WITH ^{N} ARE NEW SELF-RECORDED ONES)

Dataset	Samples	Training	Testing	Motion Type	DoF
Sarcos^P	19122	13622	5500	Rhythmic	7
Barrett^P	18572	13572	5000	Rhythmic	7
BaxterRand^N	20000	15000	5000	Random	7
BaxterRepeat^N	8918	6000	2918	Rhythmic	7
UR10^N	12352	N/A	N/A	Pick&Place	6

IV. EVALUATION RESULTS

We evaluated our proposed algorithm and compared it against three state of the art real-time learning algorithms, namely the LWPR, LGP and SSGPR, presented in [8], [11] and [15] respectively. The evaluation and comparison was performed using five different—both publicly available and self-recorded—datasets of four robots. Each of these datasets consist of tuples of position, velocity, acceleration, and applied torque values for all joints at each time step. More details about the datasets can be found in Table I. The datasets of the Sarcos and Barret robots are publicly available [11] and correspond to rhythmic, repetitive movements. Beyond these datasets, typically used in the relevant literature, we have captured and tested three new datasets on two low-cost industrial robots. The two Baxter robot datasets were generated by sampling (at 120 Hz sampling frequency) a random and a rhythmic, movement respectively. The UR10 dataset consists of samples obtained from a Universal Robots UR10 robot during a pick-and-place manipulation task of a 4 Kg object with a sampling frequency of 120 Hz[1].

The components that affect the prediction performance of the proposed deep network are the self-organized layer and the size of the reservoir. In Fig. 3 is illustrated the impact of the self-organized layer on the model's accuracy. It becomes clear that the decorrelation of the inputs, performed in the first hidden layer, results in much better torques' estimation compared to a structure with only the reservoir layer.

The most important parameter of the proposed deep neural network is the size of the reservoir because it affects both the approximation performance and the computational load. Thus, an appropriate trade-off between the reservoir's size and the approximation error has to be derived. Fig. 4 shows the normalized Mean Square Error (nMSE) obtained for various sizes of the reservoir in three datasets. Based on these results, it can be deduced that using a reservoir with 600 nodes provides a reasonable trade-off, since neither the mean, nor the deviation of the nMSE change significantly when more nodes are considered.

[1]As part of this work, we have made publicly available our three new datasets at: `https://bitbucket.org/athapoly/datasets`

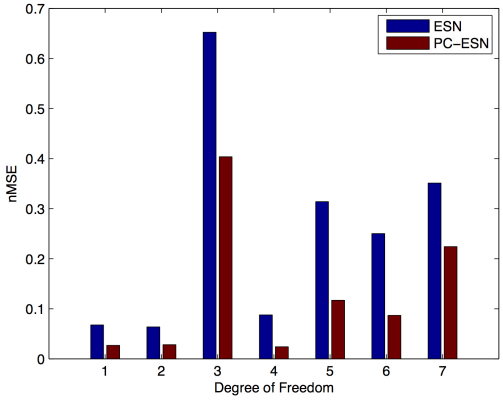


Fig. 3. Impact of including (or not) the self-organized layer in the proposed model. The prediction error (nMSE) was evaluated on the BaxterRand dataset. Blue bars (ESN) correspond to a network with only a reservoir layer, consisting of 100 nodes, while red bars (PC-ESN) corresponds to the proposed structure.

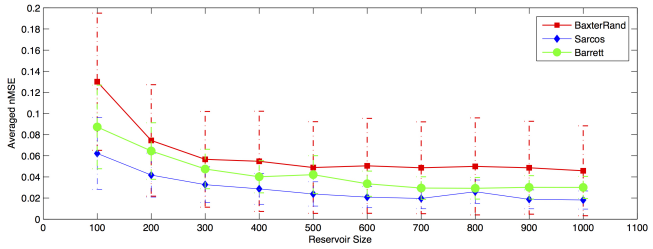


Fig. 4. Impact of the reservoir size on the error of torques' estimation for the proposed algorithm. The presented nMSE results are averaged over all joints, while the error bars indicate the standard deviations.

The generalization ability of the proposed PC-ESN algorithm was evaluated by assessing the estimation accuracy of the desired torques obtained on novel data using a trained model. In Fig. 5 the proposed algorithm was compared to the three other state of the art algorithms, on the Sarcos dataset. The results exhibit that the generalization ability of PC-ESN is comparable to that of the other algorithms.

We further evaluated and compared the convergence of the algorithms. This exhibits how fast the algorithms can learn the inverse dynamics model. The comparison results, shown in Fig. 6, were obtained using all the samples contained in the

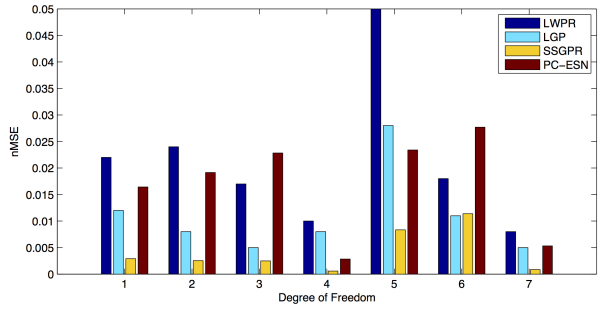
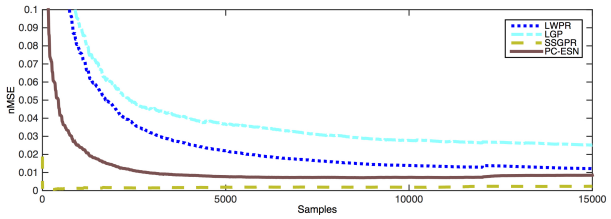
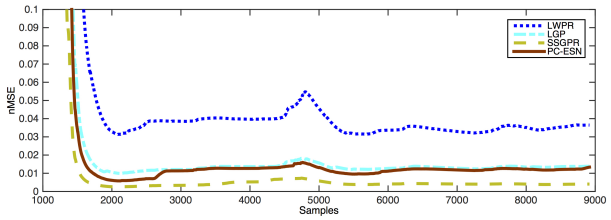


Fig. 5. Evaluation of generalization ability. All algorithms were trained with the Sarcos training set. The illustrated error corresponds to the estimated torques on the testing set. LWPR and LGP results are taken from [11].



(a) Barrett



(b) BaxterRepeat

Fig. 6. Estimation error on Barrett and BaxterRepeat dataset. Only one joint is plotted for clarity of presentation. The models were updated on-the-fly; the torques were estimated for a given input and then the models were updated using the target value of the torque.

Barrett and BaxterRepeat datasets. In both cases the proposed PC-ESN converged faster that LWPR and LGP, while SSGPR demonstrated the best performance.

A further set of experiments investigated the ability of the algorithms to adapt to changes of the inverse dynamics model. Such changes can occur during object manipulation. Data from a non-compliant robot were used because such robots adapt the applied torques in order to compensate external loads. Thus, the adaptability is evaluated on samples collected from the non-compliant UR10 robot during a pick and place operation. The dataset is extremely noisy because UR10 is not equipped with torque sensors; the torques are rather approximated based on the motors' current measurements. The results, illustrated in Fig. 7, show that the proposed algorithm is more stable than LWPR and SSGPR and similar to the LGP, while always producing the smallest error of them all.

The final set of experiments investigated the appropriateness of PC-ESN for real-time learning. All 8918 samples of the BaxterRepeat dataset were considered and we calculated the mean time and standard deviation, as measured on an Intel core i7 @ 2.4 Ghz processor with 12 GB of RAM. In Table II the times required specifically for training, prediction and hyperparameters optimization are presented for all four considered algorithms. Even though PC-ESN is slower than the other algorithms in training, it exhibits a very small standard deviation of measured execution times. This is because its complexity depends only on the size of the reservoir. The prediction time is less than 0.6 msec which allows the real-time control of a manipulator. Another advantage of the proposed algorithm is that no hyperparameter optimization procedure is required. As a result, our algorithm can be used for real-time model learning.

TABLE II

MEAN TIME (MSEC) AND STANDARD DEVIATION (MSEC) FOR TRAINING, PREDICTION AND HYPERPARAMETER OPTIMIZATION

Algorithm	Training	Prediction	Optimization
LWPR	5.73 ± 0.55	5.7 ± 0.31	Not Required
LGP	11.10 ± 6.43	5.44 ± 0.03	$36\,520 \pm 850$
SSGPR	0.36 ± 0.02	0.34 ± 0.01	$274\,170 \pm 30\,230$
PC-ESN	30.38 ± 0.05	0.56 ± 0.04	Not Required

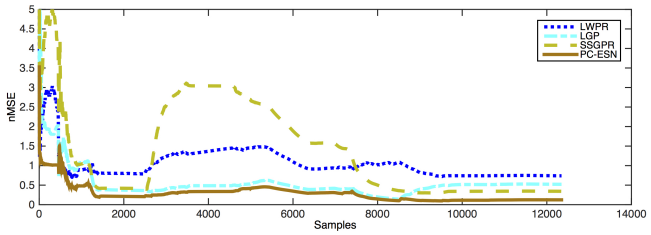


Fig. 7. Torque estimation error on a single joint of the UR10 dataset. Large error fluctuations are caused because of low adaptability.

V. DISCUSSION AND CONCLUSION

In this paper we focused on the evaluation results of the reservoir computing algorithm introduced in [1] in order to initiate a discussion about the advantages and disadvantages of such approaches on model learning. The performance of the algorithm is evaluated on two publicly available and three self-recorded datasets from four robots (Sarcos, Barrett, Baxter and UR10). The evaluations included the generalization ability, the convergence, the adaptability, the training and prediction time of the algorithms.

The generalization ability of PC-ESN is similar to the state of the art. The strengths of our algorithm are better exploited, the more frequently the model is updated. This happens due to the recurrent structure of the network where the errors are accumulated if the model is not regularly updated. Therefore, in all the real-time learning evaluations, PC-ESN exhibits better performance than LWPR and LGP. Furthermore, it converges fast in both noise-free (Barrett) and noisy (Baxter) datasets. The most important characteristic is the high adaptability which is important for applying model learning in real life applications. Adaptive learning algorithms can cope with dynamic changes of the modeled mapping. This mapping, changes e.g. in object manipulation tasks—picking and releasing objects. In such tasks the proposed approach outperforms all other considered state of the art algorithms, as it was shown to exhibit better adaptability.

In the initial tests the impact of the network's components on the prediction performance was evaluated. The most important component is the self-organized layer which decorrelates the inputs and provides better prediction performance compared to a network with only the reservoir layer. Furthermore, it was illustrated that reservoirs consisting of more than 600 nodes do not improve the estimation error. Using this size of reservoir, the weights' update frequency is approximately 30 Hz and the prediction frequency is 1 KHz. Such an update and prediction frequency, in conjunction with the presented performance, imply that the proposed algorithm is applicable for online learning of the inverse dynamics on real robotic manipulators.

As future work, we intend to extend the application of the proposed PC-ESN algorithm on a real compliant robotic manipulator using a direct model approach for online learning. We expect that such an implementation will be able to perform various manipulation tasks.

REFERENCES

[1] A. S. Polydoros, L. Nalpantidis, and V. Krüger, "Real-time deep learning of robotic manipulator inverse dynamics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015.

[2] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–40, Nov. 2011.

[3] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[4] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2013, pp. 190–198.

[5] H. Olsson, K. J. Åström, C. Canudas de Wit, M. Gäfvert, and P. Lischinsky, "Friction models and friction compensation," *European journal of control*, vol. 4, no. 3, pp. 176–195, 1998.

[6] J. Wu, J. Wang, and Z. You, "An overview of dynamic parameter identification of robots," *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 5, pp. 414–419, 2010.

[7] M. W. Spong and R. Ortega, "On adaptive inverse dynamics control of rigid robots," *IEEE Transactions on Automatic Control*, vol. 35, no. 1, pp. 92–95, 1990.

[8] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional space," in *International Conference on Machine Learning (ICML)*, 2000.

[9] D. Nguyen-Tuong, B. Scholkopf, and J. Peters, "Sparse online model learning for robot control with support vector regression," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2009, pp. 3121–3126.

[10] J. S. de la Cruz, W. Owen, and D. Kulic, "Online learning of inverse dynamics via gaussian process regression," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2012, pp. 3583–3590.

[11] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, no. 15, pp. 2015–2034, 2009.

[12] C. Hartmann, J. Boedecker, O. Obst, S. Ikemoto, and M. Asada, "Real-time inverse dynamics learning for musculoskeletal robots based on echo state gaussian process regression." in *Robotics: Science and Systems*, 2012.

[13] J. S. de la Cruz, D. Kulić, and W. Owen, "Online incremental learning of inverse dynamics incorporating prior knowledge," in *Autonomous and Intelligent Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6752, pp. 167–176.

[14] Y. Choi, S.-Y. Cheong, and N. Schweighofer, "Local online support vector regression for learning control," in *International Symposium on Computational Intelligence in Robotics and Automation*, 2007, pp. 13–18.

[15] A. Gijsberts and G. Metta, "Real-time model learning using incremental sparse spectrum gaussian process regression," *Neural Networks*, vol. 41, pp. 59–69, 2013.

[16] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural networks*, vol. 2, no. 6, pp. 459–473, 1989.

[17] K. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural networks*, vol. 6, no. 6, pp. 801–806, 1993.

[18] I. B. Yildiz, H. Jaeger, and S. J. Kiebel, "Re-visiting the echo state property," *Neural networks*, vol. 35, pp. 1–9, 2012.

[19] M. Lukoševičius, "A practical guide to applying echo state networks," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 659–686.