2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

A Reservoir Computing Approach for Learning Forward Dynamics of Industrial Manipulators

Athanasios S. Polydoros and Lazaros Nalpantidis

***Abstract*— Many robot learning algorithms depend on a model of the robot's forward dynamics for simulating potential trajectories and ultimately learning a required task. In this paper, we present a data-driven reservoir computing approach and apply it for learning forward dynamics models. Our proposed machine learning algorithm exploits the concepts of dynamic reservoir, self-organized learning and Bayesian inference. We have evaluated our approach on datasets gathered from two industrial robotic manipulators and compared it on both step-by-step and multi-step trajectory prediction scenarios with state-of-the-art algorithms. The evaluation considers the algorithms' convergence and prediction performance on joint and operational space for varying prediction horizons, as well as computational time. Results show that the proposed algorithm performs better than the state-of-the-art, converges fast and can achieve accurate predictions over longer horizons, which makes it a reliable, data-efficient approach for learning forward models.**

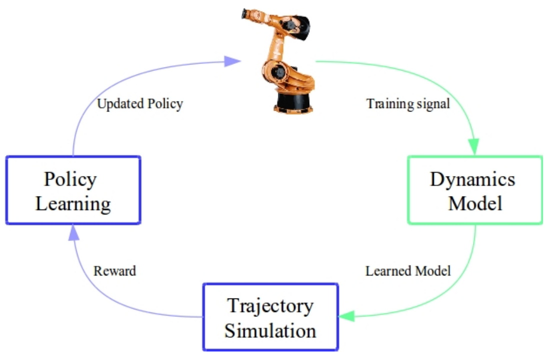

Fig. 1: Overview of a model-based RL algorithm. The dynamics model receive a training signal from the robot's sensors, then the learned model is used for simulating the trajectory that derives from the current policy. The reward of the trajectory is propagated to the policy learning algorithm which updates the policy and applies it on the robot.

I. INTRODUCTION

Modeling robots' dynamics has been an appealing topic of scientific research over the last decades with numerous proposed approaches and algorithms [1], [2]. Such models can represent a robot's embodiment, its interaction with the environment and can be used for control and simulation [3]. In particular, forward dynamics—also known as direct dynamics—provide the state of the joints given the applied forces, and can therefore be used for simulating the effects of actions on the robot state. Some applications of forward dynamics include operational space control [4], motor control [5] and formation of smooth trajectories [6]. Recently, trying to let robots learn tasks faster and minimize their interactions with the environment [7], model-based Reinforcement Learning (RL) has started gaining popularity [8]–[10]. Forward dynamics models are an essential part of model-based RL [11], as shown in Fig. 1, because the performance of such algorithms depends on the prediction accuracy of the model.

Traditionally, the derivation of forward models can be done analytically by exploiting physics equations and properties of the robot's structure. However, this approach can be problematic since it requires precise knowledge of robot properties such as inertial matrices of the links, type of joints, masses, friction coefficients, centrifugal forces e.t.c. These are difficult to calculate and, even worse, they may change depending on the wear and tear of the robot. An alternative approach, in order to overcome such problems, is the use of data-based models. In those cases—this work is one of them—the models are learned by employing machine learning approaches which use information gathered from the sensors for inferring the mapping that best approximates the model [12].

In this paper we focus on learning forward dynamics directly from sensor data. To achieve this, we draw inspiration from our previous work [13]—where we were learning inverse dynamics models—but develop an enhanced methodology to target this new, different problem. Our approach is based on reservoir computing—which, to the best of the authors' knowledge, has never been applied before in this problem. The proposed algorithm, we call this version PC-ESN++, is an enhanced version of the Principal-Components Echo State Network (PC-ESN) algorithm—as presented in [13] for learning inverse dynamics—and belongs to the class of deep learning algorithms [14]. The network consists of a self-organized layer which decorrelates the inputs, a fixed recurrent network (dynamic reservoir) and the learning rule is an iterative formulation of Bayesian regression.

In contrast to our previous work, the enhanced version illustrated in this paper avoids any assumptions about the sensor noise; instead, the noise is solely inferred by the algorithm. This fact reduces the number of parameters requiring tuning and results in a different learning rule that makes the algorithm more adaptable to different robotic

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}@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.

The authors would like to thank Mikkel Rath Pedersen for his help in gathering the dataset from the KUKA LWR iiwa.

manipulators. Furthermore, given the nature of our targeted problem—learning forward dynamics models—we have been able to simplify the structure of the network by omitting the connections from the output back to the reservoir. As a result of this, prediction errors are not propagated back to the network making the algorithm more reliable in long-term predictions. Long horizon predictions suffer from the accumulation and propagation of errors, can lead the model to unknown states and cause instability. Furthermore, we add a forgetting factor, the leaky integrator, which reduces the dynamic memory of the reservoir. This module makes the model more capable to handle long prediction horizons.

The *contribution* of this paper is three-fold. We apply a deep learning algorithm for the first time on the problem of learning forward dynamics which outperforms the current state-of-the-art algorithms in prediction performance and matches their convergence, both on step-by-step and multi-step trajectory prediction for various horizons. Moreover, the proposed algorithm is much less computationally demanding than the state-of-the-art since it is memory-less, i.e. it does not need to retain sensors' data in memory. Finally, we make publicly available the datasets we gathered from two industrial robots, the Rethink Robotics Baxter robot and the KUKA LWR iiwa arm[1].

II. Related work

Physics-based models that describe forward dynamics are widely used as deterministic models [15], [16]. The main disadvantage of such models is that they contain many factors that are difficult to be analytically expressed, such as friction and dynamics of elastic joints. Another disadvantage is that these models do not take into account potential changes of the robot's environment. On the other hand, machine learning models do not suffer from those problems because they are based only on sensor data. Thus, a workaround is to create hybrid models by combining physics models and machine learning algorithms for inferring hard to model quantities [17], [18]. Nevertheless, pure machine learning algorithms can be used to infer a mapping M such that $M(s_t, u_t) \mapsto s_{t+1}$, where s_t is the state of the robot at time step t, u_t are the applied commands and s_{t+1} is the predicted state for the next time step.

A widely used deterministic algorithm is the Locally Weighted Linear Regression (LWLR) [19], which creates a nonparametric model that fits linear regressions locally on training data. The predictions of the model's regression coefficients are made using the ordinary least square estimator, weighted by a kernel that provides a measurement of the similarity between new inputs and learned data. Thus, the data-points that are closer to the input affect the prediction more. LWLR is memory-based; it needs to keep all the training inputs in memory, which makes it a computationally expensive approach.

Stochastic machine learning models are more popular for modeling forward dynamics since they provide a level of uncertainty for their predictions. This characteristic is useful in model-based RL because the policy can be learned taking into account the noise of the system. An extension of LWLR is the Locally Weighted Bayesian Regression (LWBR), which is employed in [20] for learning the forward dynamics of a cart-pole system and in [21] for learning the forward model of an helicopter. LWBR combines Bayesian inference and LWLR. The regression coefficients are given a Gaussian prior and the noise is assumed to be described as a Gamma distribution. The weighted inputs are used for the derivation of the regression coefficients according to the learning rule of Bayesian regression. The model's output is a Student's t-distribution over the future state.

The state-of-the-art approach for learning the forward models—and in particular stochastic models—are the Gaussian Processes (GP), applied in [8], [9] for learning the forward dynamics of complex platforms like high-DOF robotic manipulators and biped robots. GP is a non-parametric method like LWBR and thus, there is not any assumption about the function M that maps current states and actions to future states. This makes it a powerful learning method. Furthermore, it employs the kernel trick for projecting the inputs into a high-dimensional space and is defined by its mean m and a kernel (covariance function) k. The mean of the GP is assumed to be zero in most cases and a very common choice for kernels are those that belong to the exponential family. Some kernels depend on parameters—the hyper-parameters of a GP. The hyper-parameters can be tuned by a variety of methods, including greedy search over the hyper-parameters' space and—more commonly—marginal likelihood-based methods. The prediction of the GP is a Gaussian distribution over the future state.

Our proposed method, the PC-ESN++, belongs to the class of non-parametric generative models such as LWBR and GP. Its main advantage over those methods is that it is memory-less; it does not keep the past inputs' in memory, which significantly decreases its computational requirements. This is achieved due to its incremental learning rule. Moreover, contrary to LWBR, it projects the inputs on high dimensional space—like GP—by using a recurrent neural network with fixed weights (dynamic reservoir). This makes our approach computationally cheaper compared to the use of kernels. Finally, the fading memory property of the dynamic reservoir can reduce accumulated errors when predicting the trajectory over long horizons.

III. The PC-ESN++

The PC-ESN++ is a deep neural network that consists of two hidden layers, an input and an output layer. The inputs are the position, velocity and applied torques at each time-step, which are propagated through a feed-forward network to the self-organized layer. That layer decorrelates them according to the Generalized Hebbian Learning (GHL) rule. The decorrelated inputs are fed to the second hidden layer, a fixed Recurrent Neural Network (RNN), which projects them to high-dimensions. At the final step we iteratively apply Bayesian Linear Regression for inferring the weights

[1]Our recorded datasets are made publicly available at `https://bitbucket.org/athapoly/datasets/`

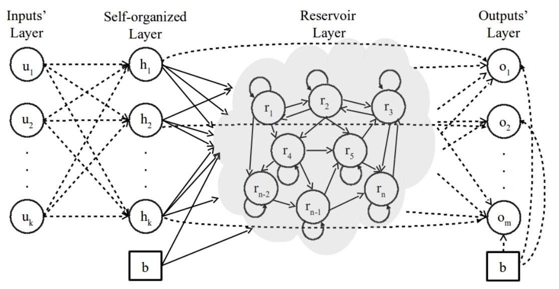

Fig. 2: Overview of the PC-ESN++ deep neural network. The neurons are represented as nodes, the weights as arcs and the bias of nodes as rectangles. The dashed lines signify weights that are updated during the learning phase.

between the RNN and the output layer. The structure of the network is illustrated in Fig. 2

Even though the method we are proposing here is inspired by our previous work [13], several adaptions and extensions needed to be performed in order for it to be applicable to the forward dynamics learning problem. Changes include both the learning rule and the structure of the used network. The new learning rule avoids the assumption that the noise of the system is known by putting a prior distribution over the noise of the target values. This results in different update rules for the weights connecting the RNN with the output layer and an additional posterior distribution over the noise. Furthermore, the structure of the used network has changed, omitting the feed-back connections from the output layer to the RNN. This change has the advantage of reducing the propagation of possible errors back to the network. Finally, we introduced a leaky integrator which acts as a forgetting module and reduces the dynamic memory of the reservoir.

A. Generalized Hebbian Learning

The values $\mathbf{h}$ of the self-organized layer nodes derive from a linear combination between the inputs $\mathbf{u}$ and the weights connecting the input layer and the self-organized layer. This can be written in matrix form as in (1):

$$\mathbf{h}_{t+1} = \mathbf{W}_t^{in}\mathbf{u}_t \tag{1}$$

where the nodes values of the input and self-organized layer are represented by the column vectors $\mathbf{u}$ and $\mathbf{h}$ respectively. The inputs' weights are represented by the triangular matrix $\mathbf{W^{in}}$ with elements w_{jk}^{in} and therefore, its entry in (j,k) is the weight from the input node k to the self-organized layer node j. The inputs' matrix is updated according to the GHL rule. GHL belongs to the class of unsupervised learning [22] and constitutes an extension of Oja's rule to multiple outputs.

The update rule is illustrated in (2) where $\mathrm{LT}\left[\cdot\right]$ denotes a lower-triangular matrix and the learning rate η_t decreases as $t \to \infty$.

$$\Delta\mathbf{W}_{t+1}^{in} = \eta_t(\mathbf{u}_t\mathbf{h}_{t+1}^T - \mathrm{LT}\left[\mathbf{h}_{t+1}\mathbf{h}_{t+1}^T\right]\mathbf{W}_t^{in}) \tag{2}$$

Thus, the values of the self-organized nodes are an approximation of the inputs' principal components. The decorrelation of the inputs is important since the algorithm is fed with high-frequency samples of sensors' values which are highly correlated. Further insights on the impact of the self-organized layer on the prediction accuracy of such a RNN, alongside additional details on its derivation can be found in [13].

B. Dynamic Reservoir

The values of the self-organized nodes are propagated to an Echo State Network (ESN). This ESN includes, in addition to the version of [13], a leaky-integrator which at each time-step reduces the memory of the reservoir by a certain amount. This element is expected to reduce the accumulated errors in the case of multi-step prediction and make the model more capable of handling long horizons. Also, ESN has been found to perform well in non-linear system identification applications [23].

ESN belongs to the class of RNNs which—unlike simple feed-forward networks—is able to approximate dynamical systems [24]. Another useful characteristic of RNNs is that their state is a unique representation of the inputs' history—due to the recurrent connections—and therefore have a dynamic memory. Despite those advantages, adaptation of the recurrent weights is a computationally expensive task. This issue is solved in ESNs by setting constant recurrent weights following a process that ensures the Echo State property. This property introduces a fading memory to the system, which should make ESN capable of coping with accumulated errors in long-term predictions. The derivation of the recurrent weights is described in more detail in [13].

The reservoir depends on four parameters: the number of nodes, its sparsity, its spectral radius and the leak rate. The reservoir's size is related to the number of training instances; small reservoirs are preferable for large numbers of training samples [25]. The sparsity affects the computational time, while the spectral radius and the leak rate of the reservoir affect its memory. The state of the reservoir is updated according to (3):

$$\mathbf{r}_{t+1} = -\zeta\mathbf{r}_t + g\left(\mathbf{W}^{res}\mathbf{r}_t + \mathbf{W}^{self}\mathbf{h}_{t+1}\right) \tag{3}$$

where the values of the reservoir at time step $t+1$ are denoted as $\mathbf{r}_{t+1}$. $\mathbf{W}^{self}$ 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 ζ is the leaking rate that causes a reservoir memory leak at each time step. Both $\mathbf{W}^{self}$ and $\mathbf{W}^{res}$ have fixed weights.

The state of the output layer is derived from a linear combination of the nodes connected to them and is calculated as:

$$\mathbf{o}_{t+1} = \mathbf{W}^{train}\mathbf{c}_{t+1}. \tag{4}$$

where all the adaptable weights and the nodes that are connected to the output layer $\mathbf{o}_{t+1}$ have been concatenated in the matrix $\mathbf{W}^{train}$ and the vector $\mathbf{c}_{t+1}$ respectively.

Thus, it is clear that inference of $\mathbf{W}^{train}$ constitutes a linear regression problem and can be updated by iteratively applying Bayesian linear regression, as explained in the following subsection.

C. Iterative Bayesian Linear Regression

In this work we avoid making any assumptions about the sensor noise; instead, the noise is solely inferred by the algorithm. This fact reduces the number of parameters requiring tuning and results in a different learning rule that makes the algorithm more adaptable to different robotic manipulators.

The goal of our learning rule is to infer the appropriate weights such that the output $\mathbf{o}$ of the PC-ESN++ approximates the true state of the robot $\mathbf{s}$. For notation simplicity we limit the derivation of the learning rule to a single output, but it can be easily generalized to multiple outputs. Thus, the weights towards the node o are represented by the row vector $\mathbf{w}^{train}$ and the forward dynamics model for a single joint at time step t can be rewritten as a regression problem such that $\mathbf{w}_t^{train}\mathbf{c}_t + \varepsilon = s_t$, where ε is noise.

Regression problems can be solved using deterministic methods such as ordinary least squares estimation and ridge regression. The Bayesian approach, which is used in PC-ESN++, derives a joint probability distribution over the regression coefficients and the variance of the noise $p\left(\mathbf{w}_t^{train}, \sigma^2 | D\right)$. The likelihood of the regression model can be written as a conditional probability distribution according to (5) and it is assumed to be a normal distribution with mean $\mathbf{w}^{train}\mathbf{c}_t$ and variance σ^2.

$$p\left(\tau_t | \mathbf{c}_t, \mathbf{w}^{train}, \sigma^2\right) = \mathcal{N}(\tau_t | \mathbf{w}^{train}\mathbf{c}_t, \sigma^2) \tag{5}$$

In (5), σ^2 is the unknown variance of the noise ε, while τ is the target value of the training sample. A conjugate prior of the normally distributed likelihood is the Normal-Inverse Gaussian distribution (NIG) which is used to represent the prior knowledge about the joint distribution of the weights and noise variance, as in (6):

$$\begin{gathered} p\left(\mathbf{w}_t^{train}, \sigma^2\right) = \\ \mathcal{NIG}(\mathbf{w}^{train}, \sigma^2 | \mathbf{w}_{t-1}^{train}, \mathbf{V}_{t-1}, \alpha_{t-1}, \beta_{t-1}) \end{gathered} \tag{6}$$

where the prior of the regression coefficients' covariance matrix is $\mathbf{V}_{t-1}$ and α, β respectively are asymmetry and scale parameters of the distribution that represents the belief before receiving a training signal at $t-1$.

When a new training signal becomes available at time step t, the posterior probability distribution is calculated by recursively applying the Bayes rule as:

$$\begin{gathered} p\left(\mathbf{w}_t^{train}, \sigma^2 | D\right) \propto \\ \mathcal{NIG}(\mathbf{w}^{train}, \sigma^2 | \mathbf{w}_{t-1}^{train}, \mathbf{V}_{t-1}, \alpha_{t-1}, \beta_{t-1}) \cdot \\ \mathcal{N}(\tau_t | \mathbf{w}_{t-1}^{train}, \mathbf{c}_{t-1}, \sigma^2). \end{gathered} \tag{7}$$

Given that the likelihood and the prior distributions are conjugate and by applying the Bayesian rule on linear Gaussian systems, the joint posterior probability is NIG as illustrated in (8):

$$\begin{gathered} p\left(\mathbf{w}_t^{train}, \sigma^2 | D\right) = \\ \mathcal{NIG}(\mathbf{w}^{train}, \sigma^2 | \mathbf{w}_t^{train}, \mathbf{V}_t, \alpha_t, \beta_t) \end{gathered} \tag{8}$$

The location of the NIG distribution $\mathbf{w}_t^{train}$ corresponds to the most probable value of the weights and is derived from:

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

where $\mathbf{V}_t$ is updated at each time-step according to the relationship $\mathbf{V}_t = \left(\mathbf{V}_{t-1}^{-1} + \mathbf{c}_t\mathbf{c}_t^T\right)^{-1}$, the initial weights $\mathbf{w}_0$ is a zero vector, and $\mathbf{V}_0 = \mathbf{I}$. The asymmetry and scale parameters of the posterior distribution are updated according to (10) and (11) respectively:

$$\alpha_t = \frac{n}{2} \tag{10}$$

$$\beta_t = \frac{1}{2}\left(\mathbf{w}_{t-1}^{train}\mathbf{V}_{t-1}\mathbf{w}_{t-1}^T + \tau^2 - \mathbf{w}_t^{train}\mathbf{V}_t\mathbf{w}_t^T\right) \tag{11}$$

where n is the number of samples at time step t.

The posterior predictive distribution for a new input $\mathbf{c}_*$ is a Student's t-distribution as illustrated in (12):

$$p\left(o_* | \mathbf{c}_*, D\right) = \mathcal{T}\left(\mathbf{W}_t^{train}\mathbf{c}_*, \tfrac{\beta_t}{\alpha_t}\left(I + \mathbf{c}_*\mathbf{V}_t\mathbf{c}_*{}^T\right), 2\alpha_t\right) \tag{12}$$

where o_* is the predicted output. Finally, the marginal posterior distributions over the unknown trained weights $\mathbf{w}^{train}$ and variance σ^2 are given by (13) and (14) respectively:

$$p\left(\mathbf{w}^{train} | D\right) = \mathcal{T}(\mathbf{w}_t^{train}, \frac{\beta_t}{\alpha_t}, \mathbf{V}_t, 2\alpha_t) \tag{13}$$

$$p\left(\sigma^2 | D\right) = \mathcal{IG}(\alpha_t, \beta_t) \tag{14}$$

Thus, the proposed learning rule infers iteratively both the regression coefficients and the variance of the sensors' noise.

IV. Evaluation Results

The proposed algorithm has been evaluated on two self-recorded datasets gathered from two different robotic manipulators, the KUKA Light Weight Robot (LWR) and Rethink Robotics Baxter (we used one of its arms, equipped with a parallel gripper). The two robots have very different characteristics when it comes to accuracy, repeatability and stiffness, making them an interesting couple to examine. The datasets contain trajectories generated during the execution of pick and place tasks. The pick and place locations were drawn randomly from two non-overlapping areas with size 50×50 cm each, as illustrated in Fig. 3. The robots were considered to have full executed a task by starting and finishing at the same location. An overview of the datasets is presented in Table I.

The performance of PC-ESN++ is compared with two other state-of-the-art non-parametric generative methods, the LWBR and GPs, as presented in Sec. II. The comparison is

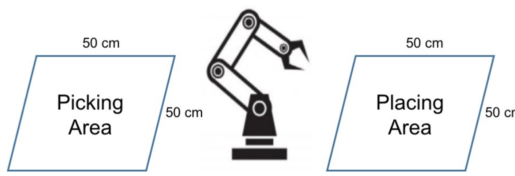

Fig. 3: Sketch of the set-up used for the creation of datasets. The exact pick and place locations were randomly chosen.

TABLE I: Description of the evaluation datasets

Dataset	Trajectories	Total # of Samples	Sampling Frequency	Motion Type	DoF
Baxter	10	19295	100 Hz	Pick & Place	7
KUKA	10	20068	120 Hz	Pick & Place	7

performed in terms of prediction accuracy and convergence in two cases—a step-by-step (Sec. IV-A) and a full trajectory prediction scenario (Sec. IV-B). Furthermore—trying to find a middle ground between these two extreme cases—the algorithms are compared in terms of operational space error for different prediction horizons (Sec. IV-C). Moreover we evaluate the computational time of the algorithms in Sec. IV-D.The inputs of the algorithms are the position, velocity and applied torques at a time step t, while the prediction output is the position and velocity at the next time step $t+1$.

A. *Step-by-step prediction*

In this evaluation scenario we assume that the actual position and velocity are available after each time step and they are used as inputs for predicting the state of the joints at the next time step. The step-by-step evaluation can be useful for model-based RL algorithms that update their policy after every single step and not at the end of the trajectory. Tables II and III illustrate the nMSE of the three evaluated algorithms in the predicted joints' position and velocity averaged over all joints and all cross-validation sets. The cross-validation sets are derived using a leave-one-out approach.

The PC-ESN++ performs better than LWBR and GPs both in position and velocity prediction. Particularly, our proposed algorithm gives an average error of $2.04 \cdot 10^{-6}$ on predicting joint positions and 0.03 on joint velocities when considering both datasets. On the other hand, the state-of-the-art GP algorithm has an average of $2.59 \cdot 10^{-5}$ and 0.3 respectively. Thus, PC-ESN++ decreased the prediction error on joint positions by 21.21% and on joint velocities by 90% compared to GP and even more compared to LWBR. Furthermore, its performance does not vary significantly between joints.

Another desirable characteristic of machine learning approaches is fast convergence—their ability to learn the dynamics model with as less training data as possible. Fig. 4 illustrates the convergence of PC-ESN++, LWBR and GP on a single joint of the Baxter robot. The algorithms are trained with an increasing number of trajectories and tested always on the same trajectory. Both GPs and PC-ESN++ converge fast and as a result they are able to learn the forward dynamics from the first trajectories. On the contrary, LWBR needs more training samples in order to achieve its optimal performance.

B. *Full trajectory prediction*

In this scenario we evaluate the ability of the algorithms to predict a full trajectory. This kind of problems are challenging and a number of approaches have been proposed for dealing with long-term time-series predictions [26]. In

TABLE II: One-step prediction error (nMSE) of the evaluated algorithms averaged over all joints and all cross-validation sets for the KUKA LWR dataset.

KUKA LWR				
Algorithm	Position Error		Velocity Error	
	Mean	St.Dev	Mean	Variance
PC-ESN++	$1.67 \cdot 10^{-6}$	$1.60 \cdot 10^{-6}$	0.04	0.01
GP	$4.96 \cdot 10^{-6}$	$6.95 \cdot 10^{-6}$	0.27	0.38
LWBR	$1.03 \cdot 10^{-4}$	$1.57 \cdot 10^{-4}$	1.61	1.62

TABLE III: One-step prediction error (nMSE) of the evaluated algorithms averaged over all joints and all cross-validation sets for the Baxter dataset.

Rethink Robotics Baxter				
Algorithm	Position Error		Velocity Error	
	Mean	Variance	Mean	Variance
PC-ESN++	$2.40 \cdot 10^{-6}$	$1.80 \cdot 10^{-6}$	0.02	0.01
GP	$4.68 \cdot 10^{-5}$	$8.93 \cdot 10^{-5}$	0.33	0.26
LWBR	$2.67 \cdot 10^{-4}$	$5.03 \cdot 10^{-4}$	1.96	1.96

this paper we restrict ourselves to testing all algorithms employing simply a recursive strategy; trying more sophisticated approaches falls outside the scope of this work. In the training phase the algorithms learn to make one-step predictions using the ground-truth as inputs. In the testing phase the inputs are the predicted values of the previous time step. Thus, the algorithms are only based on their past predicted values, and an obvious problem in this case is that errors are accumulated over the trajectory.

As shown in Tables IV and V, the full trajectory prediction performance of all the algorithms is significantly worse compared to the step-by-step scenario. This comes as no surprise since the algorithms are making future predictions based on their own, imperfect, previous predictions. However, even in this scenario the PC-ESN++ performs better than GP and LWBR, even though its prediction over the joints fluctuates more compared to its performance in the step-by-step scenario. In detail, PC-ESN++ had an average prediction error of 1.25 on joints' positions and 1.34 on joints' velocities

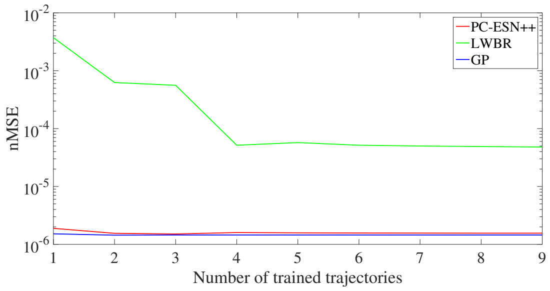

Fig. 4: Step-by-step prediction convergence of the evaluated algorithms. Only the base joint of Baxter is illustrated for clarity. The convergence is similar for all other joints.

TABLE IV: Full trajectory prediction error (nMSE) of the evaluated algorithms averaged overall joints and all cross-validation sets for the KUKA dataset

KUKA LWR				
Algorithm	Position Error		Velocity Error	
	Mean	Variance	Mean	Variance
PC-ESN++	1.01	0.85	1.48	1.21
GP	1.71	0.96	1.67	0.92
LWBR	2.42	1.25	3.59	3.52

TABLE V: Full trajectory prediction error (nMSE) of the evaluated algorithms averaged overall joints red and all cross-validation sets for the Baxter dataset.

Rethink Robotics Baxter				
Algorithm	Position Error		Velocity Error	
	Mean	Variance	Mean	Variance
PC-ESN++	1.48	1.21	1.20	0.76
GP	1.6	1.5	1.93	1.60
LWBR	3.52	3.37	1.61	1.98

when considering both datasets. The corresponding errors of GP are 2.51 and 2.765 respectively. Thus, the proposed approach decreased the prediction error by 50.2% on joints' positions and by 51.54% on joints' velocities compared to GP and even more compared to LWBR. Finally, PC-ESN++ converges faster than the other algorithms, as illustrated in Fig. 5. It needs three trajectories, while LWBR and GP need more in order to achieve their optimal performance.

C. *Varying prediction horizons*

In this part of the evaluation we explored how the algorithms perform in situations falling between the aforementioned two extreme cases. We varied the considered predictions horizons from 100 steps to 1000 steps, in increments of 100 steps. The evaluation was performed on the Baxter dataset, using 9 trajectories for training and 1 for testing—all algorithms were trained and tested with the same trajectories. For each horizon we evaluated the performance of all algorithms in terms of operational space error. The results are illustrated in Fig. 6. It can be seen that the error

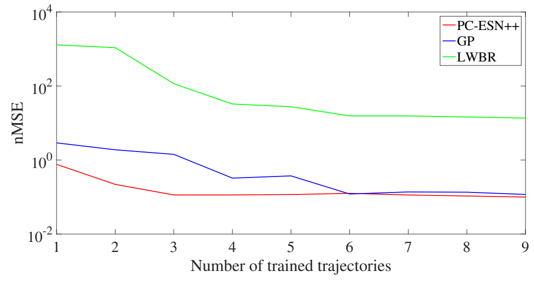

Fig. 5: Full trajectory prediction convergence of the evaluated algorithms. Only the base joint of Baxter is illustrated for clarity. The convergence is similar for all other joints.

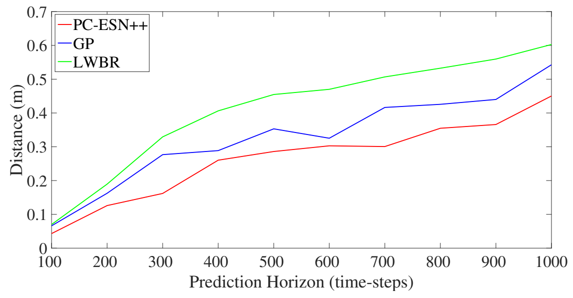

Fig. 6: Operational space error averaged over the trajectory of the Baxter robot for different prediction horizons. The error is the distance between the actual and the predicted position of the end-effector.

of PC-ESN++ in the operational space is lower than both GP and LWBR for all horizons. Furthermore, all the algorithms converge to low error for short horizons but longer ones are handled better by PC-ESN++.

D. *Computational Time*

Computational time is a significant characteristic of model learning algorithms. Fast updates of the model makes the algorithms usable in real-time, which can significantly improve the prediction accuracy. In this section we evaluated the algorithms in terms of computational time for an increasing amount of training data. Fig.7 illustrates this comparison where PC-ESN++ requires a constant amount of time, independent from the amount of data. The other methods depend on the number of training instances—they are memory-based and their computational time grows with the amount of considered data. Small fluctuations of the time are caused by operating system noise, while the sharp reduction noticeable in GP happens because the algorithm employs a more computationally efficient method after a certain amount of data.

V. CONCLUSIONS & DISCUSSION

In this paper we applied a novel memory-free and reservoir-based approach for learning the forward dynamics of robotic manipulators using only data from sensors. The

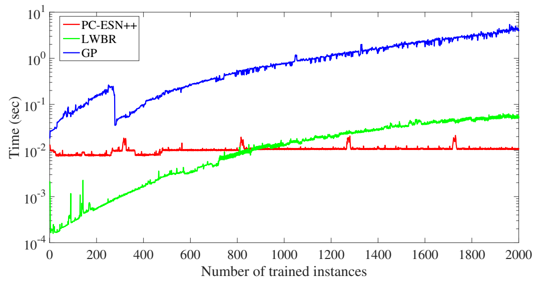

Fig. 7: Computational time of the evaluated methods for performing a model-update as a function of the training instances. All results were measured on an Intel core i7 @ 2.4 Ghz processor with 12 GB of RAM.

decorrelated outputs of the self-organized layer are propagated to a dynamic reservoir which projects them to high dimensions by a non-linear combination of fixed weights. Leaky integrators are included in the reservoir as forgetting modules. The output weights are updated by applying at each iteration the Bayes rule without making any assumption about the noise of the system.

The performance of the algorithm was evaluated on two self-recorded datasets gathered from two industrial robots (KUKA LWR and Rethink Robotics Baxter), which we make publicly available. The datasets consist of 10 different trajectories recorded during the execution of pick and place tasks. The proposed algorithm was evaluated and compared with other state-of-the-art algorithms in terms of converge and prediction accuracy in both joint and operational space. The evaluation considered two different scenarios: a step-by-step and a multi-step trajectory prediction.

The prediction performance and convergence of PC-ESN++ is better than the state-of-the-art in both datasets and for both evaluated scenarios. Furthermore, our algorithm can handle better the accumulated errors in multi-step predictions due to the fading memory of the reservoir and the leaky integrator. Another important characteristic of PC-ESN++ is its low demand for computational resources. Its complexity only depends on the size of the reservoir, while the complexity of the other algorithms depend on the number of training samples. This allows us to perform learning on large datasets without additional burden.

Even if the effect of long term predictions on accuracy was ameliorated by the use of leaky integrators, the need for a strategy that handles better the accumulated errors becomes apparent [27]. Thus, as future work, we intend to investigate the impact of different multi-step prediction approaches that have been proposed in the field of time-series forecasting for the prediction of robots' dynamics.

Finally, we intend to apply the proposed PC-ESN++ within a model-based RL algorithm for learning typical tasks performed by industrial robotic manipulators. We expect that such an algorithm would be able to learn tasks in a small number of iterations and achieve a level of accuracy that will allow it to handle manufacturing problems.

References

[1] R. Featherstone and D. Orin, "Robot dynamics: equations and algorithms," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, 2000, pp. 826–834 vol.1.

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

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

[4] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: A theoretical and empirical comparison," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 737–757, 2008.

[5] D. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Networks*, vol. 11, no. 78, pp. 1317 – 1329, 1998.

[6] Y. Wada and M. Kawato, "A neural network model for arm trajectory formation using forward and inverse dynamics models," *Neural Networks*, vol. 6, no. 7, pp. 919 – 932, 1993.

[7] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[8] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *IEEE International Conference on Robotics and Automation*. IEEE, 2014, pp. 3876–3881.

[9] A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann, "Model-based contextual policy search for data-efficient generalization of robot skills," *Artificial Intelligence*, Dec. 2014.

[10] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2944–2952.

[11] J. Kober, J. a. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, Aug. 2013.

[12] O. Sigaud, C. Salaün, and V. Padois, "On-line regression algorithms for learning mechanical models of robots: a survey," *Robotics and Autonomous Systems*, vol. 59, no. 12, pp. 1115–1129, 2011.

[13] 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.

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

[15] A. El-Fakdi and M. Carreras, "Policy gradient based Reinforcement Learning for real autonomous underwater cable tracking," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sept. 2008, pp. 3635–3640.

[16] S. Ross and J. A. Bagnell, "Agnostic system identification for model-based reinforcement learning," *Proceedings of the 29th Interational Conference on Machine Learning*, pp. 1703–1710, 2012.

[17] R. Koppejan and S. Whiteson, "Neuroevolutionary reinforcement learning for generalized helicopter control," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09*. New York, New York, USA: ACM Press, July 2009, p. 145.

[18] G. Boone, "Efficient reinforcement learning: model-based Acrobot control," *Proceedings of International Conference on Robotics and Automation*, vol. 1, 1997.

[19] C. G. Atkeson, "Nonparametric model-based reinforcement learning," in *Advances in Neural Information Processing Systems*, 1998, pp. 1008–1014.

[20] J. Schneider, "Exploiting model uncertainty estimates for safe dynamic control learning," *Advances in Neural Information Processing Systems*, 1997.

[21] J. Bagnell and J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," *Robotics and Automation, IEEE International Conference on*, vol. 2, pp. 1615—-1620, 2001.

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

[23] H. Jaeger, "Adaptive nonlinear system identification with echo state networks," in *Advances in neural information processing systems*, 2002, pp. 593–600.

[24] 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.

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

[26] A. Sorjamaa, J. Hao, N. Reyhani, Y. Ji, and A. Lendasse, "Methodology for long-term prediction of time series," *Neurocomputing*, vol. 70, no. 16, pp. 2861–2869, 2007.

[27] A. Venkatraman, M. Hebert, and J. A. Bagnell, "Improving multi-step prediction of learned time series models." in *AAAI*, 2015, pp. 3024–3030.