**2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**
**September 24–28, 2017, Vancouver, BC, Canada**

# Online Multi-Target Learning of Inverse Dynamics Models for Computed-Torque Control of Compliant Manipulators

Athanasios S. Polydoros[1,2], Evangelos Boukas[2] and Lazaros Nalpantidis[2]

*Abstract*— **Inverse dynamics models are applied to a plethora of robot control tasks such as computed-torque control, which are essential for trajectory execution. The analytical derivation of such dynamics models for robotic manipulators can be challenging and depends on their physical characteristics. This paper proposes a machine learning approach for modeling inverse dynamics and provides information about its implementation on a physical robotic system. The proposed algorithm can perform online multi-target learning, thus allowing efficient implementations on real robots. Our approach has been tested both offline, on datasets captured from three different robotic systems and online, on a physical system. The proposed algorithm exhibits state-of-the-art performance in terms of generalization ability and convergence. Furthermore, it has been implemented within ROS for controlling a Baxter robot. Evaluation results show that its performance is comparable to the built-in inverse dynamics model of the robot.**

## I. Introduction

Computed-torque controllers are among the most popular approaches for the control of robotic manipulators, since they provide the necessary torques in order to achieve a desired joint-space trajectory. Another popular approach for control is the operational space controller and both of those require a model of the robot's inverse dynamics [1].

Modeling inverse dynamics can be challenging for various types of robotic systems such as the compliant light-weight manipulators, whose utilization has significantly increased over the last years.The main advantage of compliant manipulators is their ability to operate more safely alongside humans.In order for compliant manipulators to achieve this characteristic, their stiff actuators have been replaced, or are operating in conjunction, with compliant actuators which include elastic elements, such as springs. Furthermore, inverse dynamics modeling can be challenging in biologically inspired [2] and modular robotics.

In most cases, inverse dynamics models derive analytically. In the majority of robotics systems, this requires the precise knowledge of physical properties that are, at least, hard to be calculated. Also, some of those properties may change during the robot's lifetime due to wear and tear which can result to modeling inaccuracies on which analytical models can not be adapted.

[1]Intelligent & Interactive Systems (IIS), Department of Computer Science, Universitat Innsbruck, 6020 Innsbruck, Austria. athanasios.polydoros@uibk.ac.at

[2] Robotics, Vision and Machine Intelligence (RVMI) Lab., Department of Mechanical, Production and Management Engineering, Aalborg University Copenhagen, Denmark {athapoly,eb,lanalpa}@make.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.

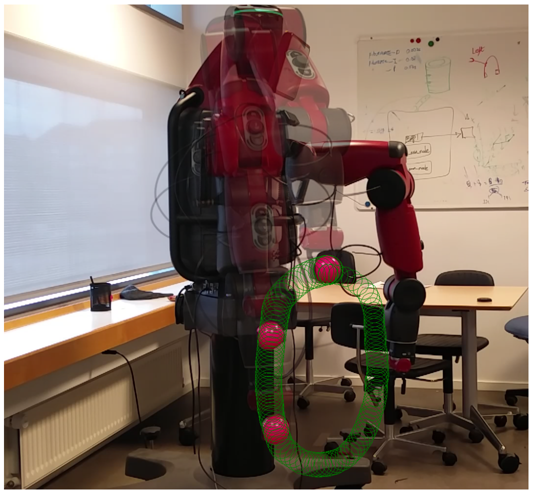


Fig. 1: Baxter robot executing a zero-shape trajectory utilizing the proposed online inverse dynamics learning algorithm.

An alternative approach to cope with such issues is the utilization of machine learning algorithms for modeling the inverse dynamics models [3]. In such cases, models are learned directly from data provided by the system's sensors. Thus, knowledge of the robot's physical properties is not required for the derivation of the inverse dynamics model. Another advantage of learned models is that they can adapt to changes of the inverse dynamics that can occur during the robot's operation. In order to achieve such a level of adaptability, the machine learning algorithm needs to be able to perform fast online updates of the system's model and also achieve fast convergence. Therefore, this work proposes a machine learning algorithm for online modeling of robotic manipulators' inverse dynamics and demonstrates its implementation within a computed-torque control scheme, on a real compliant robot.

In order to meet those requirements, we build upon our previous work in [4], [5] where the dynamics learning algorithm—the Principal Components Echo-State Network (PC-ESN)—has exhibited characteristics, such as fast convergence, low training time and high accuracy, which make it applicable to online learning [6]. Differently to our previous work, we hereby focus on ensuring efficient implementation of PC-ESN to physical, multiple Degrees of Freedom (DoF), robotic systems and thus we propose a new variant of the PC-ESN incorporating the required modifications. The enhanced algorithm differs from both its previous variations [4], [5] in terms of its learning rule, which can now perform multi-target training. Thus, a single learning model is used for modeling the inverse dynamics of all the manipulator's joints, instead of maintaining one model per joint. Moreover, the proposed inverse dynamics learning algorithm has been integrated in a computed-torque control scheme in order to perform joint-space trajectories on a compliant robotic manipulator, as depicted in Fig. 1.

The contribution of this paper is two-fold. *First*, we present a novel iterative Bayesian learning rule for the PC-ESN which is able to handle multiple outputs. This is a desired characteristic for online learning of inverse dynamics; it allows the use of a single dynamics model for the whole manipulator and significantly reduces the burden of the hardware implementation. The *second* contribution of the paper is the design and actual implementation of a control scheme that employs the PC-ESN for online, inverse dynamics learning. This control scheme has been implemented within the Robotics Operational System (ROS) and tested on the compliant Baxter robot (Rethink Robotics).

## II. Related Work

The inverse dynamics models of robotic manipulators can be analytically defined by the Newton-Euler equation of motion that relates the exerted torques $\boldsymbol{\tau}$ and the angular position $\mathbf{q}$, velocity $\dot{\mathbf{q}}$ and acceleration $\ddot{\mathbf{q}}$ of the joints. The models also depend on the inertia matrix $\mathbf{M}$, the centripetal and Coriolis torques $\mathbf{C}$ and gravitational forces $\mathbf{G}$.

Thus, the analytical derivation requires knowledge about moments of inertia, friction, gravitational and centripetal forces which can be challenging to define. Also they are subject to change depending on the wear and tear of the robot. This fact raises the need for machine learning approaches that approximate the dynamics model only based on sensors' streams. Additionally, such methods should be able to operate on-the-fly in order to ensure adaptability to dynamic changes that can occur during manipulation tasks.

In inverse dynamics learning, a function $f_{inv}(\cdot)$ is approximated by the machine learning algorithm such that:

$$f_{inv}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) + \boldsymbol{\varepsilon} = \boldsymbol{\tau} \tag{1}$$

where $\boldsymbol{\varepsilon}$ is the sensors' noise. This learning process corresponds to a regression analysis, which can be solved using a large variety of machine learning algorithms [7], [8].

A common characteristic of all online inverse dynamics learning algorithms is that they create local models on the training data. Furthermore, the majority of proposed models are probabilistic and thus generate a probability distribution over their predictions, instead of a single-point estimate. Moreover, most of the related work has been based on Gaussian Process (GP) regression [9] and numerous approaches have been proposed for reducing its computational complexity.

In [10] the authors propose the Locally Weighted Projection Regression (LWPR) for learning inverse dynamics models, which is a local method for approximating nonlinear mappings in high-dimensional space. Its main advantages are, the low computational complexity and the projection regression technique for dealing with the curse of dimensionality. However, it suffers from the large amount of tunable parameters that can be challenging to optimize. An enhancement of LWPR is presented in [11] for increasing its generalization ability by introducing prior knowledge.

Furthermore, numerous extensions of GPR have been proposed for operating with real-time data streams. In [12] the authors propose Local GP (LGP) for real-time learning where multiple Gaussian Process models are created on clusters of similar data which derive from a distance metric. The models are updated iteratively for each novel training sample. Also the method is capable to exclude samples from training based on their information gain. The model has been evaluated on a Barrett WAM robot and achieved better performance compared to other state of the art approaches. In [13], a modification of the Sparse Online Gaussian Process algorithm is illustrated, that makes feasible incremental updates of both the hyper-parameters and the model

An incremental variant of Sparse Spectrum Gaussian Process Regression (I-SSGPR) is proposed in [14]. I-SSGPR has fixed computational complexity that depends on the number of randomly sampled features that are used as a computationally cheap way of approximating kernels. The algorithm was evaluated on data collected from different robots and provides the lowest prediction error compared to other state of the art algorithms. Another GP based approach is introduced in [15]. The authors deal with the problem of computational complexity by creating multiple local approximations of GPs which are able to adapt to non-stationery data.

In [16] the authors present an hybrid method which is based on [17] and [12] for learning the inverse dynamics of a musculoskeletal manipulator. They use an Eco-State Network (ESN) for projecting the state of the manipulator into high-dimensions. The high-dimensional state is fed as input to a local GPR. Another ESN-GPR hybrid approach is presented in [18] where sparse GP approximations and recursive kernels are used for reducing the computational complexity. The hyper-parameters are optimized by the stochastic natural gradient descent rule. In [19], authors replace the PID controller, which generates the correction feedback torques, with an adaptive gradient descent learning algorithm. Therefore, the tracking error of the joints is minimized online instead of being fixed during the motion.

Our proposed algorithm does not create local models for training, but it exploits the fading memory characteristic of the Echo-State Networks (ESN) [20] for achieving high adaptability and convergence. Furthermore, it creates a single model for all joints without assuming any similarity between those. While separate models for each joint are created in the kernel-based methods. This fact can significantly reduce the implementation complexity of our algorithm on a physical robotic system. Even though a plethora of dynamics learning algorithms applied on robotic systems exists, the discussion in the relevant literature on implementation details and approaches is rather limited. Therefore, in this paper we provide information about the integration of the proposed algorithm on a physical robotic manipulator using the Robotics Operating System (ROS).

## III. Online Learning with PC-ESN

The PC-ESN belongs to the class of generative approaches and—similarly to kernel methods—it projects the inputs to a high-dimensional space. It consists of four neuron's layers: the input layer, in which the sensors' stream is fed, the output layer that provides the prediction of the dynamics, and two hidden layers. The first of the two is a self-organized layer for inputs' decorrelation, while the second is a Recurrent Neural Network (RNN) with a large number of neurons. The structure of the network is presented in Fig. 2.

In the rest of this section, we briefly present the structure and characteristics of the deep neural network's hidden layers (Sec III-A) followed by an elucidating description of the paper's contributions. Thus, the novel aspect of our learning approach, namely the iterative multivariate Bayesian regression is presented in Sec. III-B while the proposed implementation method is described in Sec. III-C.

### A. The hidden layers

The first hidden layer is a self-organized layer that consists of the same amount of nodes as the input layer in which the sensors' stream is fed. The nodes' values $\mathbf{h}$ derive from a linear combination between the inputs $\mathbf{u}_t$ and the corresponding $k \times k$ weights' matrix $\mathbf{W^{in}}$ as illustrated in (2) where $g(\cdot)$ is a sigmoid function.

$$\mathbf{h}_{t+1} = g(\mathbf{W}^{in}\mathbf{u}_t) \tag{2}$$

The adaptation rule of the weights' matrix is unsupervised Hebbian-based. Namely the Generalized Hebbian Learning (GHL) rule is used, which is an extension of Oja's rule to multiple outputs. The weights which are updated according to Oja's rule approximate the first eigenvector of the input stream [21]. Thus, the output of the network is an approximation of the inputs' first principal component. Since GHL generalizes Oja's rule to multiple outputs, the weights matrix $\mathbf{W}^{in}$ has been proven to approximate the inputs' eigenvectors for large amount of data [22]. The update rule

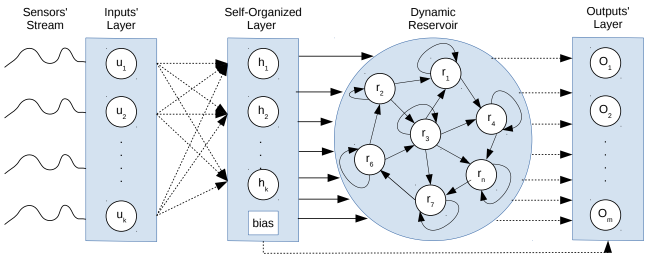


Fig. 2: Illustration of the PC-ESN components. Nodes represent the neurons and the bias of the network while arcs represent its weights. Fixed weights are represented with solid arcs while weights that change according to a learning rule are represented with dashed arcs.

is illustrated in (3) where $\mathrm{LT}[\cdot]$ denotes a lower-triangular matrix and the learning rate $\eta_t$ decreases as $t \to \infty$.

$$\Delta\mathbf{W}^{in}_{t+1} \Leftarrow \eta_t(\mathbf{u}_t\mathbf{h}^T_{t+1} - \mathrm{LT}\left[\mathbf{h}_{t+1}\mathbf{h}^T_{t+1}\right]\mathbf{W}^{in}_t). \tag{3}$$

The impact of the self-organized layer on the prediction accuracy of the algorithm is illustrated in [4].

The second hidden layer is an Echo-State Network (ESN) which is a special case of RNNs with fixed weights. These weights are calculated in such a way as to ensure the Echo-State property. The ESNs have characteristics that make them applicable to problems of online dynamics modeling. First, they are more capable of approximating nonlinear dynamical systems compared to feed-forward networks [23]. Furthermore, the state of the network is unique for each sequence of inputs due to the recurrent connections [24]. Furthermore, ESNs have fixed recurrent weights that are defined only once and initialized so as to guarantee the Echo-State property [25]. Thus, only the weights towards the output layer are trained, which significantly reduces the computational complexity.

A difference between the presented architecture and the one proposed in [4] is that the feedback connections from the output layer to the reservoir are omitted. This change is motivated from [5] where it was found that such network architecture can propagate wrong estimations back to the reservoir resulting to poor performance.

Four parameters are required for the initialization of the ESN: the amount of nodes, its sparsity, spectral radius and leak rate which, contrary to the kernels' hyper-parameters can all be easily tuned based on the characteristics of the modeled system [26]. The size of the ESN affects both the prediction performance of the algorithm and its computational complexity. Furthermore, the main impact of sparsity is on computational time; usually reservoirs with $90\%$ sparse connections are used. The spectral radius and the leak rate affect the memory and values smaller than $1$ are assigned. A more detailed discussion about the impact of those parameters is presented in [4].

The nodes of the ESN $\mathbf{r}_{t+1}$ are updated according to:

$$\mathbf{r}_{t+1} = g\left(\mathbf{W}^{res}\mathbf{r}_t + \mathbf{W}^{self}\mathbf{h}_{t+1}\right) \tag{4}$$

where matrices, $\mathbf{W}^{self}$ and $\mathbf{W}^{res}$ have fixed weights and represent the connections from the self-organized layer to the ESN and the nodes of the reservoir respectively. The process for initializing those weights' is presented in [4]

The state of the output layer $\mathbf{o}_{t+1}$ is given as follows:

$$\mathbf{o}_{t+1} = \mathbf{W}^{out}\mathbf{r}_{t+1} + \mathbf{W}^{dir}\mathbf{h}_{t+1} \tag{5}$$

Where $\mathbf{W}^{out}$ and $\mathbf{W}^{dir}$ are the weights' which connect the ESN and the self-organized layer with the output respectively. For notation simplicity, all the updated weights are concatenated in the matrix $\mathbf{W}^{train}$ and the nodes' values of the ESN and self-organized layer to the vector $\mathbf{c}$ which has length $m$ equal to the sum of the self-organized and reservoir nodes. As a result, (5) can be written as:

$$\boldsymbol{v}_{t+1} = \mathbf{W}^{train}\mathbf{c}_{t+1}. \tag{6}$$

## B. Multi-target training

The previous learning approach of PC-ESN [4] can only be applied for predicting a single output. This can be problematic for implementing PC-ESN online on a robotic system since a separate model for every joint has to be created and therefore models have to be trained in parallel for achieving an online learning system. This fact increases the implementation difficulty since those models has to be synchronized both for training and prediction. Therefore, in order to make the implementation easier, we create a single model for dynamics of the whole manipulator that is able to be trained for multiple targets.

In the multivariate case, the regression problem is expressed as in (7) where the target value is a column vector of length $d$ similarly to the noise vector $\boldsymbol{\epsilon}$ and the trained weights is a $d \times m$ matrix.

$$\mathbf{W}_t^{train}\mathbf{c}_t + \boldsymbol{\epsilon} = \boldsymbol{v}_t \tag{7}$$

Furthermore, it is assumed that the noise is modeled by a multivariate Normal distribution $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ with zero mean vector and unknown variance matrix $\boldsymbol{\Sigma}$. Consequently, the likelihood is also a multivariate Normal distribution and given in (8).

$$p\left(\boldsymbol{v}_t|\mathbf{c}_t, \boldsymbol{W}_t^{train}, \boldsymbol{\Sigma}\right) \sim \mathcal{N}(\hat{\boldsymbol{v}}_t|\mathbf{W}_t^{train}\mathbf{c}_t, \boldsymbol{\Sigma}) \tag{8}$$

At the next step, a prior distribution has to be set over the trained weights. In the previous approaches, the trained weights were a vector of values and the prior over those could be modeled as multivariate Normal distribution. In this case the trained weights are a matrix and hence their prior distribution (see (9)) is the Matrix-Normal ($\mathcal{MN}$) which is conjugate with the likelihood.

$$p(\boldsymbol{W}) \sim \mathcal{MN}\left(\boldsymbol{W}_t^{train}, \boldsymbol{\Sigma}, \boldsymbol{K}\right) \tag{9}$$

The Matrix-Normal distribution depends on three parameters, the mean matrix $\boldsymbol{W}_t^{train}$ and two covariance matrices, the row covariance matrix $\boldsymbol{\Sigma}$ and the $m \times m$ column covariance matrix $\boldsymbol{K}$. Random variables from a Matrix-Normal distribution can be sampled by converting it to a multivariate Normal as shown in (10) where $vec(\boldsymbol{W}_t^{train})$ derives by stacking the columns of matrix $\boldsymbol{W}_t^{train}$ creating a vector and the $\otimes$ operator denotes the Kronecker tensor product.

$$p\left(vec(\boldsymbol{W}_t^{train})\right) \sim \mathcal{N}\left(vec(\boldsymbol{W}_t^{train}), \boldsymbol{K}^{-1} \otimes \boldsymbol{\Sigma}\right) \tag{10}$$

The parameter of the mean is initialized as a zero matrix while the two covariance matrices are initialized as unit matrices at $t = 0$. The posterior distribution (11) is also a Matrix-Normal and derives by multiplying the likelihood with the prior, similarly to the previous cases.

$$p(\boldsymbol{W}|\boldsymbol{\Sigma}, \boldsymbol{v}_t) \sim \mathcal{MN}\left(\boldsymbol{W}_{t+1}^{train}, \boldsymbol{\Sigma}, \boldsymbol{K}_{t+1}\right) \tag{11}$$

The update rule for the trained weights derives from the mean of the posterior distribution in (12) and the updated raw-covariance matrix in (13).

$$\boldsymbol{W}_{t+1}^{train} = \left(\boldsymbol{v}_t{\mathbf{c}_t}^T + \boldsymbol{W}_t\boldsymbol{K}_t\right)\left(\mathbf{c}_t{\mathbf{c}_t}^T + \boldsymbol{K}_t\right)^{-1} \tag{12}$$

$$\boldsymbol{K}_{t+1} = \left(\mathbf{c}_t{\mathbf{c}_t}^T + \boldsymbol{K}_t\right) \tag{13}$$

Similarly, the noise covariance $\boldsymbol{\Sigma}$ derives by selecting an appropriate conjugate prior. In this case, the prior distribution of the noise covariance matrix is selected to be the Inverse-Wishart (14) which is defined by two parameters, the positive-definite $d \times d$ scale matrix $\boldsymbol{\Sigma}$ and the degree of freedom which are initialized as: $\boldsymbol{\Sigma}_0 = \boldsymbol{I}_d$ and $q = d$.

$$p\left(\boldsymbol{\Sigma}|q\right) \sim \mathcal{W}_q^{-1}(\boldsymbol{\Sigma}_0) \tag{14}$$

By applying Bayesian inference, the posterior mean is calculated as:

$$\begin{gathered}\boldsymbol{\Sigma}_{t+1} = \left(\boldsymbol{v}_t{\boldsymbol{v}_t}^T + \boldsymbol{W}_t\boldsymbol{K}_t\boldsymbol{W}_t^T\right) - \\ \left(\boldsymbol{v}_t{\mathbf{c}_t}^T + \boldsymbol{W}_t\boldsymbol{K}_t\right)\boldsymbol{K}_{t+1}^{-1}\left(\boldsymbol{v}_t{\mathbf{c}_t}^T + \boldsymbol{W}_t\boldsymbol{K}_t\right)^T\end{gathered} \tag{15}$$

The posterior predictive distribution is also analytically tractable and it is given by a multivariate t-distribution as:

$$p\left(\hat{\boldsymbol{v}}|\mathbf{u}^*, D\right) \sim \mathcal{T}\left(\boldsymbol{W}_{t+1}^{train}\mathbf{c}_*, \alpha\boldsymbol{\Sigma}_{t+1}, q+1\right) \tag{16}$$

where $\alpha = \left(1 + \mathbf{c}_*^T\boldsymbol{K}_{t+1}\mathbf{c}_*\right)$. The weights' matrix is updated by a new training signal iteratively at each time step $t$ and Bayesian updates are taking place recursively. Thus, the posterior distribution over the weights at time step $t$ is used as prior distribution at $t+1$.

The algorithm's computational complexity, as shown in the update rules of the weights and the two covariance matrices (12), (13), (15), does not depend on the number of training samples but only on the number of nodes that are connected to the output. Due to this characteristic, a constant training time is achieved which is desirable for the application of learning dynamics models to robotics, since the sensor's produce hundreds of training signals each second.

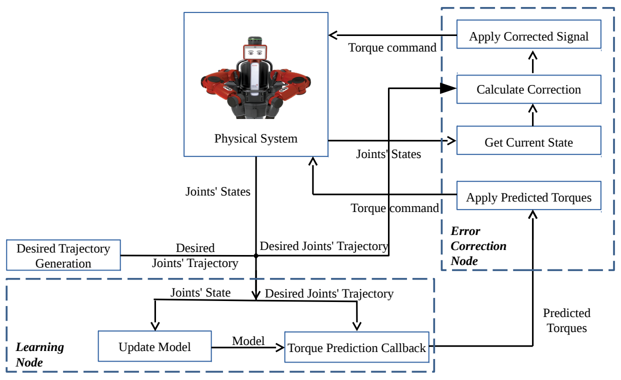


Fig. 3: Flow diagram of the hardware implementation for inverse dynamics model learning

## C. Hardware Implementation

In this subssection we describe the implementation of the PC-ESN on a compliant robotic manipulator—namely on one of the arms of the Baxter robot (Rethink Robotics)—using ROS for learning its inverse dynamic model. The specific robot is an interesting case for dynamics' model learning since it a employs a Series Elastic Actuators (SEAs) [27] and, moreover, its shoulder-pitch joint is spring-driven. Those characteristics can render the analytical derivation of the model difficult, resulting in a poor inverse dynamics model and consequently inaccurate trajectories.

The implemented software architecture, as illustrated in Fig. 3 consists of two nodes, namely the learning node and the feed-back node which run in parallel. The former is responsible for both the training and the prediction of the model. It consists of two threads that are executed in parallel. The first thread continuously receives the training signal from the robot which is used for model updates. By having a thread which continuously receives the training signal of the model it is ensured that the model is up-to-date with the state of the robot. The second thread is activated when a trajectory request is received and uses the trained model in order to predict the torques, which are then applied on the robot. The trajectory requests contain the desired time-stamped positions, velocities and accelerations for each joint and in this case they are generated using the MoveIt motion planning framework.

Furthermore, the model, used in the prediction thread, is continuously updated with learning signals that are generated during the trajectory execution. This characteristic enhances the ability of the system to operate on-line and adapt faster.

The predicted torques alongside with the desired trajectory are fed to the feed-back loop which compensates for any errors made in the model's prediction. At each time-step of the trajectory, the error between the actual and the desired joints' position and velocities is calculated and the correction torque signal derives as in (17). The terms $\Delta\boldsymbol{p}$, $\Delta\dot{\boldsymbol{p}}$ represent the error in joints position and velocities respectively, while $\boldsymbol{z}_p$, $\boldsymbol{z}_{\dot{p}}$ the corresponding error gains, which are set empirically. The correction torques are added to the predicted torques of the model and applied to the robot.

$$\Delta\boldsymbol{\tau} = \boldsymbol{z}_p\Delta\boldsymbol{p} + \boldsymbol{z}_{\dot{p}}\Delta\dot{\boldsymbol{p}} \tag{17}$$

## IV. Evaluation Results

The enhanced version of PC-ESN has been evaluated both on batch (Sec. IV-A, IV-B) and online learning (Sec. IV-C). In batch learning the proposed method has been compared to three other state of the art methods, the LWPR, the LGP and the I-SSGPR in terms of generalization ability, convergence and prediction accuracy. The evaluation datasets were recorded from two different robots, the Sarcos and the Baxter while performing rhythmic and point-to-point motions respectively.

Furthermore, PC-ESN has been implemented on the real Rethink Baxter robot for evaluating its online learning performance. The specific robot is a challenging case for inverse dynamics learning not only due to its SEAs which are used to measure the joints' torque but also due to its spring-driven shoulder pitch joint. The online learning of PC-ESN has been assessed in terms of operational space error during a variety of trajectory executions.

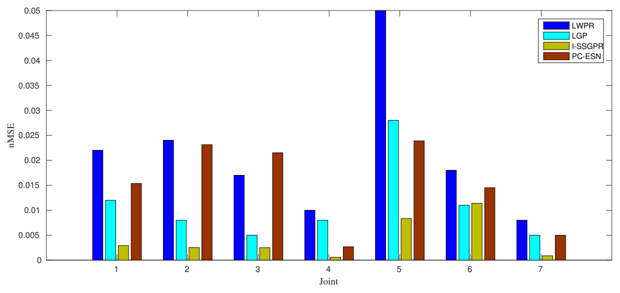
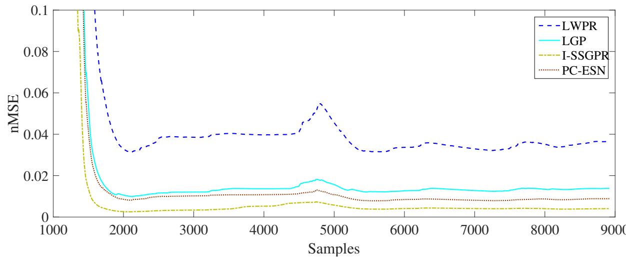
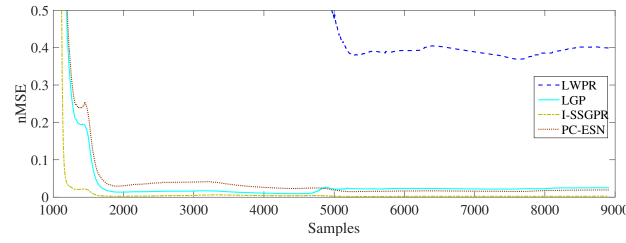


Fig. 4: Normalized-Mean Square Error (nMSE) of the enhanced PC-ESN and other state-of-the-art algorithms on the Sarcos dataset in terms of Generalization ability.



(a) Joint 1



(b) Joint 2

Fig. 5: Convergence of the compared algorithms on Baxter dataset.

## A. Generalization

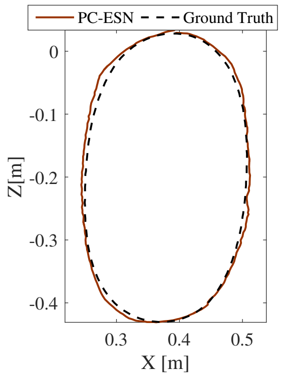
The generalization performance of the algorithm has been evaluated on the publicly available Sarcos dataset which consists of 13922 training and 5569 testing samples. The results are illustrated in Fig. 4. PC-ESN appears to have similar performance with other learning algorithms which have been applied on physical robotics systems for inverse dynamics modeling. The I-SSGPR appears to have the best overall generalization accuracy while LWBR and PC-ESN perform similarly. We must stress here that the two latter methods are the only which can be trained over multiple targets. Thus they handle a higher dimensional learning problem compared to I-SSGPR and LGP.

TABLE I: Prediction Error of the evaluated methods on the Baxter Dataset.

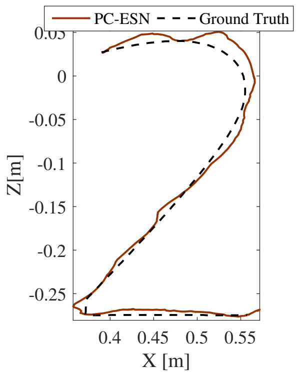
| *Algorithms* | *nMSE of Predicted Torques on Baxter Dataset* | | | | | | |
|---|---|---|---|---|---|---|---|
| | **Joint 1** | **Joint 2** | **Joint 3** | **Joint 4** | **Joint 5** | **Joint 6** | **Joint 7** |
| **LWBR** | 0.035 | 0.398 | 0.031 | 0.056 | 0.027 | 0.043 | 0.061 |
| **LGP** | 0.013 | 0.025 | 0.048 | 0.002 | 0.002 | 0.001 | 0.053 |
| **I-SSGPR** | 0.005 | 0.01 | 0.011 | 0.002 | 0.003 | 0.001 | 0.037 |
| **PC-ESN** | 0.008 | 0.019 | 0.008 | 0.009 | 0.006 | 0.004 | 0.048 |

TABLE II: Mean Absolute Operational Space Error (in meters) for each trajectory

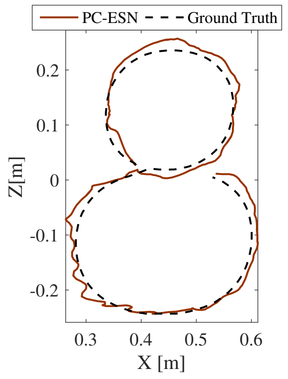
| *Inverse Dynamics Model* | *Shape of trajectory* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **0** |
| **PC-ESN** | 0.009 | 014 | 0.018 | 0.011 | 0.019 | 0.016 | 0.015 | 0.018 | 0.019 | 0.014 |
| **Built-in Model** | 0.007 | 0.09 | 0.01 | 0.008 | 0.008 | 0.008 | 0.07 | 0.01 | 0.008 | 0.009 |
| **Baseline – Error Correction only** | 0.252 | 0.31 | 0.28 | 0.25 | 0.23 | 0.33 | 0.35 | 0.33 | 0.34 | 0.265 |



(a) 0-Shape Trajectory



(b) 2-Shape Trajectory



(c) 8-Shape Trajectory

Fig. 6: Trajectories created through a computed-torque control scheme with PC-ESN for online inverse dynamics learning

## B. Convergence

In order evaluate the algorithm's convergence, a publicly available[1] dataset has been used. It consists of 8918 samples and was captured using the Baxter robot while executing an 8-shaped trajectory [4]. In this evaluation, the models are updated at each time step in order to approximate as closely as possible an online learning scenario. Thus, once a new input is available, the models provide a prediction and then they are updated with the corresponding target value. Their performance on the first two joints of the robot throughout the whole trajectory is illustrated in Fig. 5. The rationale behind the selection of the specific joints is that they are the ones that move the most during the trajectory and additionally that joint 2 is a spring-joint (Fig.5(b)).

In Fig.5 is demonstrated that PC-ESN converges similarly to the LGP and I-SSGPR while LWBR under-performs on modeling the spring-joint. Furthermore, Table I summarizes the nMSE of the algorithms over all the robot's joints. In this case, the proposed algorithm performs similarly to LGP and I-SSGPR and also exhibits better relative performance compared to the case of generalization. Thus, PC-ESN achieves its best possible prediction performance when it is regurarly updated.

## C. Online evaluation

Additionally to the previous evaluations which have been performed offline on captured datasets, the proposed algorithm has been employed online within a computed-torque control scheme for trajectory execution on a Rethink Baxter robot. The on-line performance of the PC-ESN, has been evaluated on 10 number-shaped trajectories (0-9) where learning and prediction take place on-the-fly i.e. while the trajectories are executed. Furthermore, its performance is compared to the built-in dynamics model of the Baxter robot and with a baseline controller. The performance of the models is measured in terms of Mean Absolute Error (MAE) in the Cartesian space which is presented in Table **??** and illustrations of selected trajectories are presented in Fig. 6. The measurements of the PC-ESN error are recorded after 30 repetitions of the trajectories.

The PC-ESN provides errors which range from 0.9 cm for simple trajectories to 1.9cm for more complex trajectories while the performance of built-in controller fluctuates less.

[1]`https://bitbucket.org/athapoly/datasets/`

Overall the proposed dynamics learning method does not yield trajectories that are more than 1 cm off than the built-in model although that such a comparison is unfair due to implementation issues. The robot's built-in controller is executed in a real-time loop which is proprietarily locked. Thus, the presented control scheme (Fig 3) can suffer from fluctuations of the system's latency. Such latencies were observed to be at least one order of magnitude larger than the internal ones and to cause motion jitter as the one observed in Fig.6(c)

## V. Discussion and conclusion

In this paper an online multi-target learning method for inverse dynamics modeling has been presented. The algorithm decorrelates the input feed and projects it to a high-dimensional space through a recurrent neural network. The learned model is updated once a new learning signal is available by iteratively applying Bayesian multivariate linear regression. The learning rule is capable of training the model over multiple outputs which makes the algorithm easily implementable on physical robotic systems.

The PC-ESN has been evaluated both offline and online. Offline evaluation was performed on publicly available datasets from two different robots. The proposed method was compared with three other state-of-the-art methods in terms of generalization ability and convergence. Online evaluation was performed by integrating the learning algorithm within a computed-torque controller for performing trajectory execution using the compliant Baxter robot. The online performance of the algorithm is comparable to the built-in inverse dynamics model of the robot, despite the fact that the last is running in a real-time loop which users cannot access. This fact makes the built-in controller capable of compensating fast the inaccuracies of the dynamics model. Despite this advantage, which makes the comparison unfair, the small difference between the performance of those control modules provides grounds for assuming that the learning approach would at least perform same as the analytical approach if both were implemented within a real-time control loop. Last but not least, it has to be pointed out that the built in controller is carefully tuned and optimized for the specific robot, while the proposed data driven approach requires no a priori knowledge and thus can be deployed in various different robots. Furthermore, the advantages of the learned model would be clear in cases where the robot manipulates heavy parts and on high-speed motions. Those cases will be further investigated in our future work.

## References

[1] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. Wiley New York, 2006, vol. 3.

[2] K. Narioka and K. Hosoda, "Motor development of an pneumatic musculoskeletal infant robot," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 963–968.

[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] 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.

[5] A. S. Polydoros and L. Nalpantidis, "A reservoir computing approach for learning forward dynamics of industrial manipulators," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[6] ——, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[7] 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.

[8] A. Droniou, S. Ivaldi, V. Padois, and O. Sigaud, "Autonomous online learning of velocity kinematics on the icub: A comparative study," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 3577–3582.

[9] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning*. MIT Press, 2006.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] F. Meier and S. Schaal, "Drifting gaussian processes with varying neighborhood sizes for online model learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 264–269.

[16] 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.

[17] S. P. Chatzis and Y. Demiris, "Echo state gaussian process," *Neural Networks, IEEE Transactions on*, vol. 22, no. 9, pp. 1435–1445, 2011.

[18] H. Soh and Y. Demiris, "Spatio-temporal learning with the online finite and infinite echo-state Gaussian processes." *IEEE transactions on neural networks and learning systems*, vol. 26, no. 3, pp. 522–36, 2015. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/25720008

[19] F. Meier, D. Kappler, N. Ratliff, and S. Schaal, "Towards robust online inverse dynamics learning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 4034–4039.

[20] H. Jaeger, "Short term memory in echo state networks," 2001.

[21] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, no. 3, pp. 267–273, 1982.

[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] 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.

[24] H. Jaeger, "Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the" echo state network" approach," 2002.

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

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

[27] G. A. Pratt and M. M. Williamson, "Series elastic actuators," in *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, vol. 1. IEEE, 1995, pp. 399–406.