Learning and Adaptation of Inverse Dynamics Models: A Comparison

Kevin Hitzler[1], Franziska Meier[2], Stefan Schaal[2] and Tamim Asfour[1]

***Abstract*— Performing tasks with high accuracy while interacting with the real world requires a robot to have an exact representation of its inverse dynamics that can be adapted to new situations. In the past, various methods for learning inverse dynamics models have been proposed that combine the well-known rigid body dynamics with model-based parameter estimation, or learn directly on measured data using regression. However, there are still open questions regarding the efficiency of model-based learning compared to data-driven approaches as well as their capabilities to adapt to changing dynamics. In this paper, we compare the state-of-the-art inertial parameter estimation to a purely data-driven and a model-based approach on simulated and real data, collected with the humanoid robot Apollo. We further compare the adaptation capabilities of two models in a pick and place scenario while a) learning the model incrementally and b) extending the initially learned model with an error model. Based on this, we show the gap between simulation and reality and verify the importance of modeling nonlinear effects using regression. Furthermore, we demonstrate that error models outperform incremental learning regarding adaptation of inverse dynamics models.**

I. Introduction

Humans are able to move their body through space with high accuracy, even when moving fast, lifting heavy objects or using tools. To achieve similar capabilities, a robot needs to learn a representation of its own body with its dynamic properties that can be adapted to new situations and environmental interactions.

In the context of robot control, this is known as the inverse dynamics problem. Given a desired trajectory with joint positions, velocities and accelerations, a dynamics model is used to predict the torques (or efforts) that have to be applied on each individual joint [1]. Deriving such an inverse dynamics model analytically, e.g. by making use of the rigid-body dynamics formulation, is not sufficient as it does not consider nonlinearities like backlash or friction and thus, would lead to large tracking errors. In such cases, the robot would constantly have to track its current position and compensate the errors with high-gain feedback control. This makes it dangerous to interact with the real world and impossible to work in human-centered environments. Additionally, there are cases in which the dynamic parameters are only partially known in advance or might not even be available at all, so that they need to be identified at run-time. Current state-of-the-art methods address these problems by estimating the inertial parameters of a robot [2] or learning the inverse dynamics model directly from measured data using regression, see e.g. [3], [4], [5]. However, these methods strongly rely on the data used for model inference. Hence, they might only approximate the robot's dynamics from previously seen tasks or workspace configurations. Due to the fact that the robot's dynamics always depends on the current state, offline learning would require large amounts of training data from a well-explored state space. For robots with many degree-of-freedom (DoF), this is not easily feasible.

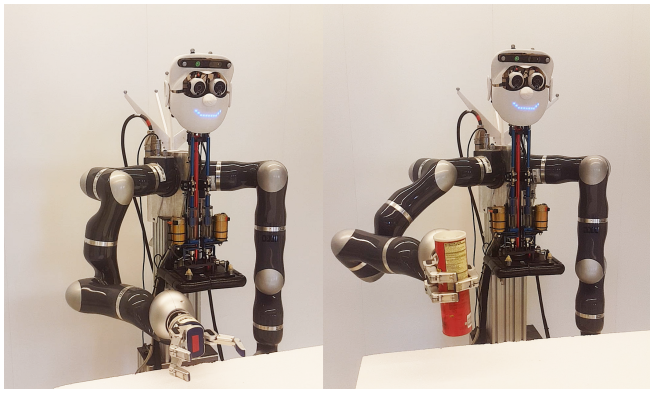


Fig. 1. The humanoid robot Apollo with 7 DoFs in each arm, executing a pick and place task without (left) and with an object (right)

Furthermore, one has to keep in mind that the robot's dynamics can change during task execution, for example when using tools or lifting heavy objects. This means that the robot would either have to learn new dynamics or continuously adapt its existing model. A promising way to overcome these limitations is to *incrementally learn* the inverse dynamics model online [5], [6] or *learn an error model* on top of an existing one [7], [8], [9]. However, there are many challenges when it comes to online learning as the robot has to adapt to the new dynamics without losing the ability of its previously learned model, whereas many error model approaches assume that learning can be done in a task-specific way. Hence, the robot needs to have knowledge about when to switch to a certain task [8], [10].

While regression-based learning methods for inverse dynamics have been explored in the past [11], there are still many open questions regarding the efficiency of model-based learning compared to data-driven approaches as well as their capabilities to adapt to new situations with changing

The research leading to these results has received funding from the German Research Foundation (DFG: Deutsche Forschungsgemeinschaft) under Priority Program on Autonomous Learning (SPP 1527) and from the German Federal Ministry of Education and Research (BMBF) under the project ROBDEKON (13N14678). The research was conducted in collaboration between the Max-Planck-Institute of Intelligent Systems, the University of Southern California and the Karlsruhe Institute of Technology.

[1]Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Karlsruhe, Germany. {hitzler, asfour}@kit.edu

[2]Max-Planck-Institute of Intelligent Systems, Germany and Computational Learning and Motor Control Lab, University of Southern California, USA. Franziska Meier is currently with Facebook AI Research and Stefan Schaal is currently with Google X.

dynamics. In this work, we target both challenges. First, we compare three different methods for inverse dynamics learning, the state-of-the-art inertial parameter estimation [2], the model-based Dynamic Bézier Map [12] and a purely data-driven approach that uses feedforward neural networks [13]. The methods are evaluated 1) in simulation and 2) two of them based on real data collected with the humanoid robot Apollo (Fig. 1) with 7 DoFs in each arm. After that, we compare the adaptation capabilities of the previously learned models in a pick and place scenario, executed on a real robot, while a) learning the models incrementally and b) extending the models with an error model.

II. Fundamentals

In the following, we describe the basic concept of inverse dynamics models in the context of feedforward robot control and provide an overview of existing state-of-the-art approaches. After introducing the necessary theory, the most interesting methods are evaluated in detail.

A. Inverse Dynamics in Feedforward Control

The inverse dynamics equation represents the mapping between the robot's motion expressed in generalized coordinates (i.e. joint positions q, velocities $\dot{q}$ and accelerations $\ddot{q}$) and the generalized forces τ (i.e. joint torques) that are required to achieve a particular motion. The symbols q, $\dot{q}$ and $\ddot{q}$ as well as τ denote n-dimensional vectors where n is the robot's number of DoFs. The inverse dynamics of a robot with n joints can then be described as

$$\tau = H(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) + \varepsilon(q,\dot{q},\ddot{q}) \tag{1}$$

where $H(q)$ is the symmetric and positive-definite inertia matrix, $C(q,\dot{q})\dot{q}$ is the centripetal and Coriolis force, $g(q)$ is the gravitational component and $\varepsilon(q,\dot{q},\ddot{q})$ are nonlinear backlash and friction effects [14]. In order to control the robot, e.g. by using feedforward torque control, the generalized forces have to be transmitted to torque commands and sent to the motor controller. Typically, the motor commands are computed as

$$\tau = \tau_{FF} + \tau_{FB} \tag{2}$$

where τ_{FF} is the feedforward and τ_{FB} the feedback component. Given a desired motion with desired positions q_d, velocities $\dot{q}_d$ and accelerations $\ddot{q}_d$, the feedforward component τ_{FF} can be predicted by an assumed inverse dynamics model. In common linear feedback controllers (PD), the feedback term can be described as

$$\tau_{FB} = K_p e + K_v \dot{e} \tag{3}$$

where $e = q_d - q$ is the tracking error and K_p and K_v represent the feedback gains. Selecting the correct gains can be seen as a trade-off between tracking the desired trajectory with the inverse dynamics model and keeping the system stable [1].

There exist various methods to derive an inverse dynamics model from the above equation of motion when neglecting $\varepsilon(q,\dot{q},\ddot{q})$. The most commonly used are the Lagrange and the Newton-Euler formulation, derived from the rigid body dynamics [14]. However, applying these methods can lead to inaccurate inverse dynamics models, since many of the robot's parameters are usually not known in advance and must be assumed such as the inertial parameters (i.e. the mass, center of mass and inertia tensor matrix). Furthermore, when making use of the rigid body dynamics, nonlinear backlash and friction effects are neglected so that the feedforward command results in

$$\tau_{FF} = H(q_d)\ddot{q}_d + C(q_d,\dot{q}_d)\dot{q}_d + g(q_d) \tag{4}$$

which is only a rough approximation of the equation of motion. Thus, it can lead to poor torque predictions and cause imprecise movements of the robot.

B. Learning Inverse Dynamics

For the reasons given above, many efforts have been made to identify the robot's dynamics parameters at runtime, or learn the inverse dynamics model directly from measured data. State-of-the-art inertial parameter estimation, for example, learn the inertial parameters of a robot by re-formulating the well-known Newton-Euler equations into a linear combination of known and unknown parameters and solving a least-squares problem [2], [15]. With the prior knowledge of the rigid body dynamics, the resulting dynamics model is able to learn fast and generalize well on simulated data, but it does not consider any non-linearities and may lead to poor results on a real robot.

Nonlinear regression methods such as LWPR [5], GPR [11] and ν-SVR [4] overcome this problem by directly learning from measured data of a robot. The local learning method LWPR is very efficient due to its low computational costs but often performs poorly in unseen regions, i.e. they suffer from insufficient extrapolation capabilities [16], whereas the global learning methods GPR and ν-SVR can learn more accurate models but need much more time for training [11]. Due to the fact that regression methods only learn based on previously seen data, they can only approximate the real function of the robot's inverse dynamics.

The so-called "Dynamic Bézier Map" (DBM) proposed in [12] allow to learn an exact encoding of a robot's inverse dynamics, represented by Lagrangian equations. Thus, they can provide both high accuracy in trained regions, i.e. good interpolation, as well as good extrapolation capabilities. But this comes at a high cost as they require a large amount of training data, especially if the training data is noisy.

The need to improve the inter- and extrapolation capability of inverse dynamics models as well as their ability to adapt to new situations, raises the following questions: Which approach is suited best for inverse dynamics learning? What are the limitations of model-based and data-driven approaches? How can the models perform in changing dynamics and adapt to new situations? To answer these questions, we compare three different methods for inverse dynamics learning: 1) the state-of-the-art inertial parameter estimation that makes use of the prior knowledge about rigid body dynamics, 2) a purely data-drive approach with feedforward neural networks which directly learns from measured data, and 3)

a model-based approach that tries to learn an exact encoding of the robot's inverse dynamics. In the following, we first present the learning of inverse dynamics models. After that, we show how the initially learned models can be adapted to changing dynamics using incremental and error model learning.

III. Inverse Dynamics Models

In general, learning an inverse dynamics model can be described as finding the function $f(q,\dot{q},\ddot{q})$ that maps the applied torques τ to the target values $y=\tau$, given an observed trajectory with joint positions q, velocities $\dot{q}$ and accelerations $\ddot{q}$. If the learned function is sufficiently accurate, it can then be used to predict the feedforward motor commands for a desired motion with $f(q_d,\dot{q}_d,\ddot{q}_d)=\tau_{FF}$. The following sections provide an overview of the inertial parameter estimation, the regression-based neural networks and the model-based Dynamic Bézier Map.

A. Inertial Parameter Estimation

The inertial parameter estimation, further referred to as *PEST*, makes use of the recursive Newton-Euler algorithm derived by the rigid body dynamics. More specifically, the Newton-Euler equations are re-formulated in a linear equation to only depend on the known parameters (i.e. the robot kinematics and applied torques) and unknown parameters (including the mass, center of mass and inertia matrix of the robot's links). This way, the linear equation can be described as

$$\tau = K\psi \tag{5}$$

where τ denotes the applied forces/torques on each link, K is the *regression matrix* with the prior knowledge of the robot's kinematic motion and ψ are the 10 *inertial parameters* of each link [2]. Unfortunately, the inertial parameters cannot be estimated by simply applying least squares, defined as

$$\psi = (K^T K)^{-1} K^T \tau \tag{6}$$

because the term $K^T K$ has a loss of rank and thus, is not invertible [2]. In other words, some of the inertial parameters are completely unidentifiable, whereas others are only identifiable in linear combinations [15]. There are multiple solutions to identify the important parameters, e.g. by using the well-known principal component analysis or the QR-decomposition method. The PEST model in this work implements the singular value decomposition (SVD) of the regression matrix, defined as

$$K = U\Sigma V^T \tag{7}$$

where U and V are orthogonal matrices and Σ is the diagonal matrix of ordered singular values ρ on the diagonal [15]. To eliminate the irrelevant parameters, all singular values that are below the threshold $\rho_i \leq 0.001$ are set to 0. The solution of the least squares problem can then be described as in [15]:

$$\Delta\psi = (V\Sigma^{+}U^T)\tau. \tag{8}$$

Note that if the model is trained with perfect data, i.e. data without noise and nonlinear backlash or friction, the parameter estimation is able to find a perfect solution for the inverse dynamics problem. However, due to the fact that the term $K^T K$ from Equation 6 is not of full rank, we only calculate a pseudo-inverse of the *regression matrix K*. Consequently, the identified inertial parameters $\Delta\psi$ typically do not match the physical inertial parameters ψ of the robot.

TABLE I
FFNN parameters for each joint of the NN model

Joint	FFNN structure	η	Batch size	Epochs
1	$3d,100,1$	0.001	32	100
2	$3d,100,100,1$	0.001	32	100
3	$3d,100,1$	0.0007	32	100
4	$3d,100,1$	0.005	32	100
5	$3d,100,1$	0.001	32	100
6	$3d,100,1$	0.001	32	100
7	$3d,100,1$	0.001	32	100

B. Feedforward Neural Networks

The regression-based inverse dynamics model in this work, further referred to as *NN* model, implements multiple feedforward neural networks. Feedforward neural networks (FFNN) are well-known for their nonlinear regression capability [13]. Thus, they pose an interesting alternative for learning a robot's inverse dynamics, particularly with regard to nonlinear backlash and friction effects. Contrary to the inertial parameter estimation, the NN model does not have explicit knowledge about the robot, aside its number of DoFs.

For every robot joint, the NN model trains a separate neural network that directly learns from measured data. The input of each FFNN is the robot's state, i.e. the positions q, velocities $\dot{q}$ and accelerations $\ddot{q}$ of all joints. Consequently, the input layer takes $3d$ inputs where d is the number of DoFs. The predicted value is the torque τ of the robot's joints so that the output layer results in size 1. To find a suitable network architecture, we trained the FFNNs multiple times with different training parameters and network configurations. The results were evaluated by analyzing the training and validation loss. Table I shows the final structure of the FFNN per joint as well as its training parameters (i.e. the *learning rate* η, *batch size* and number of *epochs*). The robot's first joint, for example, takes an input size of $3d$, has one hidden layer with 100 neurons and an output size of 1. All networks consist of fully connected layers and use the nonlinear *ReLU* activation function [13], apart from the *linear* output layer.

C. Dynamic Bézier Map

The work in [12] presents a completely different approach that can be used for both learning the kinematics as well as the dynamics model of a robot. The Dynamic Bézier Map (DBM) is a parametrizable model that makes use of the tensor product of rational Bézier functions, a method from the field of computer aided geometrical design, to learn the inverse dynamics of a robot. To this end, the DBMs encode a combination of squared trigonometric functions to

represent the Lagrangian formulation from Hollerbach [17]. In this way, they are able to learn an exact representation of the inertia matrix $H(q)$, the centripetal and Coriolis force $C(q,\dot{q})\dot{q}$ and the gravity component $g(q)$ from the equation of motion (Equation 1).

To demonstrate the DBM abilities, the authors of [12] use a dynamic simulator in the experiments with canceled nonlinearities. The results show that the DBM algorithm is able to provide high accuracy with good extrapolation capabilities. This also applies to situations where the model is trained with noisy data. But these features come at a high cost, as the number of model parameters increases drastically with the robot's number of joints. This limits applications to a small number of DoFs, especially in the case of noisy data, where a large amount of training samples is required.

Due to the fact that DBM show both good interpolation and extrapolation capabilities even in the presence of noise, they pose an interesting alternative for learning the robots inverse dynamics. Therefore, we evaluate the DBM approach in detail and compare it to the inertial parameter estimation and regression-based neural networks. Since we use an exact implementation of the Dynamic Bézier Map, we refer to [12] for further details of the method.

IV. Adaptation of Inverse Dynamics Models

Learning an inverse dynamics model *once* is not sufficient as the dynamics of a robot might change when interacting with the real world. Thus, the inverse dynamics model has to *continuously* learn, i.e. adapt or extend its existing knowledge, to cope with new situations. In the remainder of this section, we discuss two approaches for adapting the inverse dynamics, the incremental learning and the error model learning.

A. Incremental Learning

One way to adapt the inverse dynamics model is to learn the model *incrementally* during task execution. This means that the model is trained with every new sample (the extreme case) or with a small number of samples received from the data stream of the current task. However, a key challenge of incremental inverse dynamics learning is to extend the underlying model without losing the ability of its previously learned model. Furthermore, it requires a computationally efficient learning algorithm that is able to estimate the model's parameters at run-time.

In this work, we apply the well-known δ-rule [18], also referred to as the *Widrow-Hoff rule*, to update the weights of the NN model as it allows an incremental adaptation of the model parameters in an efficient way with low computational costs [13]. To optimize the incremental learning process, we performed multiple experiments on the NN model with different *batch sizes*, number of *epochs* and *adaptation rates*, i.e. number of data points the model is incrementally trained with. Choosing the correct training parameters can be seen as a trade-off between adapting to the new dynamics and not losing the knowledge of the previously learned batch model. At the same time, we ensure that the training parameter candidates are within a reasonable range regarding their training time and computational costs. The best training configuration for incremental learning was performed with an *adaptation rate* of 100, a *batch size* of 32 and learning 5 *epochs* which is also applied in the experiments. The average training time of a single FFNN in this configuration is 0.68 seconds. However, the experiments were conducted on a single core of a CPU without code optimization and thus, could further be improved for online learning.

B. Error Model Learning

Another way to adapt a robot's inverse dynamics, is to learn an *error model* on top of an existing dynamics model, as e.g. proposed in [8] or [9]. Error models benefit from the fact that they do not directly depend on the underlying dynamics model. Instead, they only make use of the model's predictions but are learned separately. In particular, the feedforward command sent to the motor controller can be interpreted as a combination of the torque predictions from the existing dynamics model and the learned error model. Let τ_{pred} be the torque prediction of the inverse dynamics model, then the feedforward command τ_{FF} can be expressed as

$$\tau_{FF}(q,\dot{q},\ddot{q}) = \tau_{pred}(q,\dot{q},\ddot{q}) + \tau_{err}(q,\dot{q},\ddot{q},\omega) \tag{9}$$

where ω are error model parameters and τ_{err} are the predicted error torques [9]. Note that both models depend on the actual state of the robot, i.e. the positions q, velocities $\dot{q}$ and accelerations $\ddot{q}$. Given the applied torques τ sent to the robot and its observed state $(q,\dot{q},\ddot{q})$, the error model can be learned by minimizing the loss $\mathscr{L}$ with respect to the predicted torques from the inverse dynamics model [8]:

$$\mathscr{L} = \sum_{(q,\dot{q},\ddot{q},\tau)\in\mathscr{D}} \| \underbrace{\tau - \tau_{pred}(q,\dot{q},\ddot{q})}_{\hat{\tau}_{err}} - f_{err}(q,\dot{q},\ddot{q},\omega)\|^2 \tag{10}$$

where $f_{err}(q,\dot{q},\ddot{q},w)$ is the function learned by the nonlinear error model, given its model parameters ω and the current robot state $(q,\dot{q},\ddot{q})$. For each sample in the data set $\mathscr{D}$, the inverse dynamics model first has to predict the torques τ_{pred}. After that, the predicted torques τ_{pred} are subtracted from the applied torques τ so that the difference $\tau - \tau_{pred} = \hat{\tau}_{err}$ can then be used as the target value to train the inverse dynamics error model [8].

Similar to the work of [8], we employ feedforward neural networks to model the error of each individual robot joint. Our error model uses the same network structures as the NN model described in Table I, because the target function of the NN model is very similar to the one that has to be approximated by the error model. The error models are then learned incrementally to extend the existing PEST and NN models during task execution.

V. Experiments

In this section, we first describe the general data acquisition process for our experiments. Following this, we compare the learning performance of PEST, NN and DBM based on simulated data including noise-free and noisy data sets of a

reduced 4-DoF model of the humanoid robot Apollo. The PEST and NN model are then evaluated on real data of the 7-DoF Apollo arm. Finally, the adaptation capabilities of the inverse dynamics models are compared to each other on real data of multiple pick and place tasks by extending the PEST and NN model with error models and learning the NN model incrementally.

A. Data Acquisition

1) Simulated data without noise: The simulated data is collected in the software package SL [19] with a model of Apollo. The trajectories are generated by executing a sine task on all robot joints simultaneously such that they cover a sufficiently rich robot state space. To acquire a wide range of data with well-varying dynamics, the sine task is executed multiple times with different velocities which results in slow (low frequency) and fast (high frequency) movements of the robot and thus, leads to different forces/torques acting on the robot's joints. The sine motion is executed 8 times with frequencies ranging from 0.1Hz and 0.8Hz, where the velocity is increased by 0.1Hz in each task. Every sine task has an execution time of 10 seconds and the data is measured with a sample rate of 500Hz. To limit the number of training samples for the batch learning experiments, the Apollo robot model is reduced to 4 DoFs, only considering the most upper links. Due to the fact that the joint velocities, accelerations and torques depend on the succeeding links, they are not directly used from SL. Instead, the velocities are computed by differentiating over the measured positions through a convolution filter with a window size of 3. The accelerations are then calculated in the same way, using the velocities. Given the joint positions q, the velocities $\dot{q}$ and accelerations $\ddot{q}$, the ground truth torques τ_{FF} are computed with the recursive Newton-Euler algorithm (RNEA) [20]. The total number of data points N from 8 sine tasks executed with different velocities results in $N = 8 \times 4.500 = 36.000$.

To evaluate the inverse dynamics models with respect to both their *interpolation* and *extrapolation* capabilities, we split the data into different training and test sets. The *interpolation* experiment evaluates whether the models are able to predict the forces/torques from a range of data points that lie in the trained region. Thus, the models are trained on frequencies ranging from 0.1Hz to 0.4Hz and 0.7Hz to 0.8Hz, whereas the tests are performed on data from 0.5Hz and 0.6Hz. In case of *extrapolation*, the models are tested on data samples that lie outside the trained region. Accordingly, the training set consists of data from 0.1Hz to 0.6Hz and the models are tested on 0.7Hz and 0.8Hz. An overview of the *simulated data* sets used for inter- and extrapolation is depicted in the first two rows of Table II. Note that both data sets are generated based on the positions from SL and thus, represent perfect data from a simulated robot without noise.

2) Simulated data with noise: To evaluate the inverse dynamics models under more realistic conditions, the data is modified to simulate the inaccuracies of a robot's position sensing. This is achieved by truncating the positions after the 5^{th} decimal point. As before, the velocities and accelerations

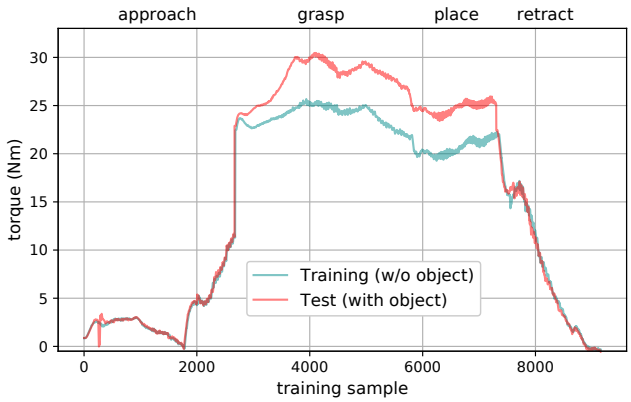


Fig. 2. Measured torques on the first robot joint during execution of a pick and place task with (cyan) and without an object (red)

are then computed by a convolution filter. To evaluate the prediction errors of the inverse dynamics models, the RNE-torques are calculated based on the *unmodified* SL positions. Note, if we used the *modified* SL positions to calculate the torques, the methods would only have to learn the mapping of the RNE-equations. Or in other words, learning the inverse dynamics would be the same as in our experiments with noise-free data, apart from slightly different positions – whereas learning from noisy positions and accurate RNE-torques allows us to evaluate the errors made by the models due to learning from inaccurate robot states. The composition of the noisy training and test set is given in Table II.

3) Real data: Instead of generating a new data set, this work makes use of existing measurements from [21] which were collected on the real humanoid robot Apollo with 7 DoFs in each arm. Hence, it comprises both noisy data as well as nonlinear backlash and friction effects. The data is measured during multiple *pick and place tasks* which include the following stages: 1) *approach* target object, 2) *grasp* and lift object, 3) move to target position and *place* object, 4) *retract* arm from object.

First, the pick and place task is conducted without an object such that the robot is able to learn its own inverse dynamics. After that, an object with a mass of 851 g is used (see Fig. 1) to which the robot has to adapt its inverse dynamics model. In the real data experiments, the robot learns from observations of its own state, i.e. q, $\dot{q}$ and $\ddot{q}$ at time step t, after sending the motor commands τ to the robot in time step $t-1$. Hence, one training sample of the inverse dynamics models can be described as $(q_t, \dot{q}_t, \ddot{q}_t, \tau_{t-1})$. Each pick and place task is repeated 10 times and lasts about 9-10 seconds. The data is sampled with 1kHz so that the total number of data points is 90000 per task.

In this work, the real data set is used for both evaluating the *batch learning* as well as the *adaptation* of inverse dynamics models. In the *batch learning* experiments, the models are first trained on tasks *without the object*. Then, the models are evaluated on the test set *with the object* to evaluate their accuracy after changing the robot's dynamics. Fig. 2 shows the measured torques of the first robot joint while executing the pick and place task without (cyan) and

TABLE II
DATA SETS USED FOR LEARNING AND ADAPTATION

Experiment description	Training set [Hz]	Test set [Hz]
Sim. data w/o noise (Interpolation)	0.1-0.4, 0.7-0.8	0.5, 0.6
Sim. data w/o noise (Extrapolation)	0.1-0.6	0.7, 0.8
Sim. data with noise (Interpolation)	0.1-0.4, 0.7-0.8	0.5, 0.6
Sim. data with noise (Extrapolation)	0.1-0.6	0.7, 0.8
Real data (Batch learning)	without object	with object
Real data (Continuous learning)	with object	–

with (red) an object. Between the training sample 3.000 and 7.500, while grasping and lifting the object, the torques yield much higher values due to the additional object weight. In the *adaptation* experiments, the initially learned models are learned *continuously* to adapt to the weight of the object. Thus, the data set with the object will be used as a training set in context of continuous learning, as shown in Table II.

B. Batch Learning

In the batch learning experiments, PEST, NN and DBM models are trained on the full training set and evaluated on the test set (Table II). The inverse dynamics models are then compared with respect to their *Normalized Mean Squared Error*, defined as $\text{NMSE} = \text{MSE}(P)/\text{Var}(T)$ where P are the predictions of an inverse dynamics model and T are the target values. The NMSE is calculated for each robot joint separately. After that, the mean NMSE of all links is computed to get the total NMSE of the model.

1) Simulated data: The batch learning experiment results on simulated data are shown in Fig. 3a. On the *interpolation set without noise*, all methods learn good models of the robot's inverse dynamics. The PEST and DBM model are able to encode an exact representation of the inverse dynamics as their torque prediction errors are very close to zero. The NN model is able to interpolate the training data and finds a good solution as its prediction error decreases below a NMSE of 0.1, indicated by the black line.

The *extrapolation experiment without noise* illustrates that PEST as well as DBM are able to learn a generalized inverse dynamics model, because both methods show good extrapolation capabilities. Furthermore, they only have slightly higher prediction errors than in the interpolation experiment. Accordingly, they are even able to predict the torques from robot states that lie outside the trained region. The NN model's extrapolation on the contrary, is not as good as its interpolation. After training, its NMSE on the test set is greater than 0.1. Thus, the NN can only predict torques accurately if they are in a certain range of the trained region.

In the *interpolation experiment with noise*, all inverse dynamics models show good results but suffer from the noisy data as their prediction errors on the interpolation set increase almost up to a NMSE of 10^{-2}. Compared to the interpolation experiment without noise, the prediction errors of the PEST and the DBM model differ the most, because they are no longer able to encode the exact inverse dynamics. The NN model does not change too much in its learning behavior as it is still able to interpolate the data. However, it can be seen that on the noisy data set, the NN model reaches the same prediction accuracy as DBM and PEST.

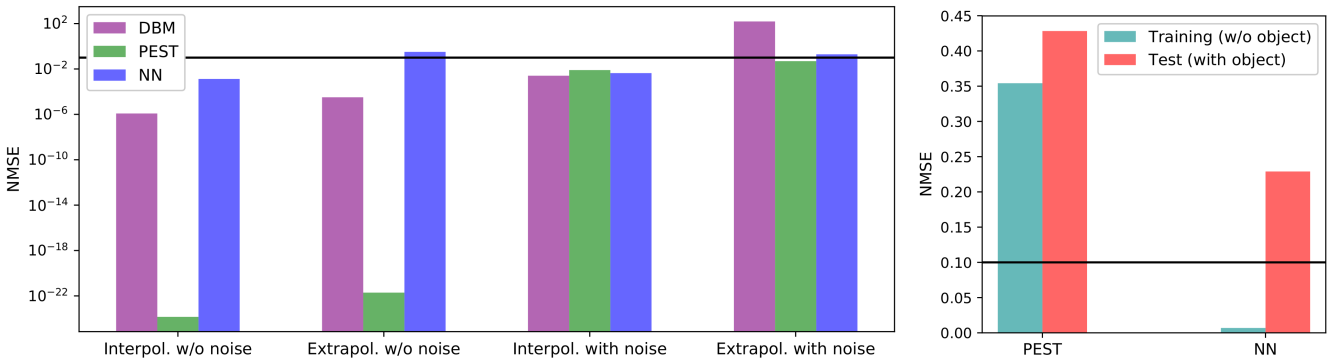
The *extrapolation experiment with noisy data* evaluates both the model's capability of learning a generalized model from a restricted training region as well as their ability to cope with noisy data. The noisy extrapolation experiment results in Fig. 3a show that all of the model's extrapolation capabilities suffer heavily from noisy data. The predictions of the PEST are now very close, but still below a NMSE of 0.1. The DBM on the contrary, show bad extrapolation capabilities on the noisy data set. That is, with a NMSE greater than 10^2. The prediction error of the NN model did not change much compared to the other experiments. Although it is not able to interpolate, due to the training set in this experiment, the NN model almost reaches a NMSE of 0.1 which is very close to the accuracy of the PEST model. At the same time, one should keep in mind that the PEST makes use of the prior knowledge of the robot's kinematics, whereas the neural networks do not comprise any prior knowledge.

2) Real data: In the real data experiments, we first compare the PEST and the NN model after learning from data of a pick and place task *without an object*, executed on the humanoid robot Apollo with 7 DoFs. After that, the previously learned models are tested on the same task performed *with an object* to examine the models' capabilities regarding changing dynamics. The DBM model is excluded from the real experiments as the number of training samples required to encode a DBM increases exponentially with the number of DoFs.

Fig. 3b shows the real data experiment results. It can be seen that the PEST model is not able to encode the inverse dynamics very well as its NMSE on the training set is 0.35. This is due to the nonlinear backlash and friction effects which cannot be encoded by the PEST and thus, results in a big gap between simulation and reality. The NN method on the contrary, is able to learn the robot's inverse dynamics very well with a NMSE of 0.007. This performance is consistent with simulated data experiments from above as it also reflects the good interpolation capabilities of the NN model. On the test set with the object, however, both models are far beyond a NMSE of 0.1, because the object in the robot's hand results in a different inverse dynamics of the robot. As a consequence, the inverse dynamics models need to be learned continuously during task execution to adapt to the changing dynamics.

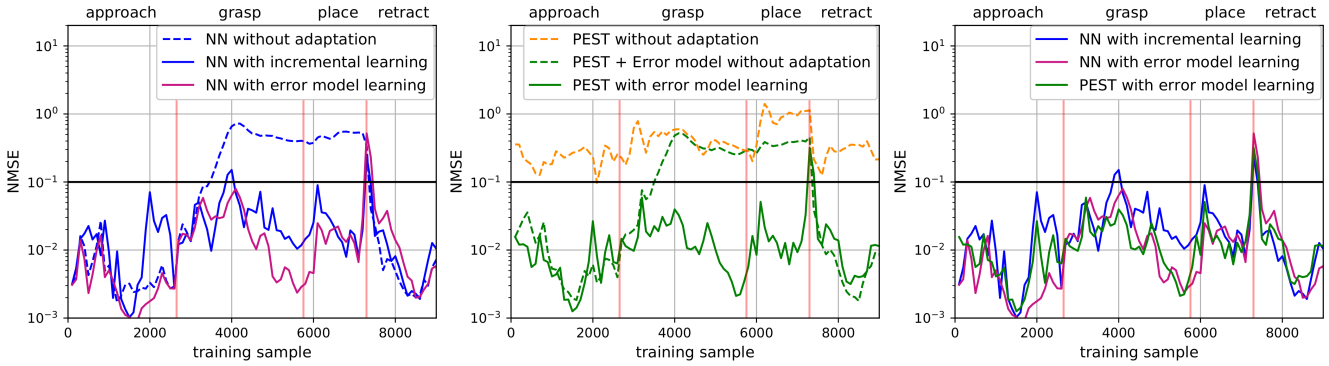
C. Continuous Learning

We evaluate the adaptation capabilities of the NN and the PEST method using *incremental* and *error model learning*. Similar to the real data experiments, the NN and PEST batch models are first trained on the pick and place task *without the object*. For the NN batch model, we use a slightly different structure as its performance could further be improved on real data. The FFNN configuration for each individual joint is $(21, 100, 100, 100, 1)$ while batch learning is performed for



(a) (b)

Fig. 3. Batch learning performed on simulated data (Fig. 3a) of a reduced 4-DoF Apollo model, considering the the inter- and extrapolation capabilities of the DBM (purple), the PEST (green) and the NN model (blue) on noise-free and noisy data. In the real experiments (Fig. 3b), the PEST and NN model are trained on real data of the full 7-DoF Apollo arm executing a pick and place task without an object (cyan) and tested on a data set with an object (red).



(a) (b) (c)

Fig. 4. Continuous learning experiments performed on real data of a pick and place task. The red lines indicate the task's stage changes (e.g. from approach to grasp). Fig. 4a shows the learning performance of the NN batch model without adaptation (dashed blue), using incremental learning (solid blue) and error model learning (purple). In Fig. 4b, the performance of the pure PEST batch model (dashed orange) is compared to its extension with an error model with (solid green) and without (dashed green) adaptation. Fig. 4c shows the comparison of the NN model with incremental learning (blue), the NN model with error model learning (purple) and the PEST model with error model learning (green).

200 *epochs* with a *learning rate* of $\eta = 0.001$. Similar to the NN and PEST batch models, the error models are pre-trained on data *without the object* using the predictions of the NN and PEST models, respectively. After that, the models are continuously learned on data of the pick and place task *with the object* using an *adaptation rate* of $\lambda = 100$, i.e. the models always learn based on the next λ data points received from the data stream. In every training step, the models predict the torques on the small subset S. Based on these predictions, the NMSE is computed for each joint as $\text{NMSE} = \text{MSE}(S)/\text{Var}(T)$ where T represents the target values (i.e. applied torques) of the pick and place task.

The results of the continuous learning experiments in Fig. 4a show that the *NN batch model without adaptation* (dashed blue) performs well during the approach and the retract stage in which the robot does not interact with the object. During grasping and placing the object, however, the robot's inverse dynamics changes due to the additional weight so that the NMSE of the NN model without adaptation increases drastically. *The incrementally learned NN model* (solid blue) on the contrary, is able to adapt to the changing dynamics of the robot. One can see that its NMSE slightly increases up to 0.1 during grasping, but recovers after 4000 training samples. In the subsequent place stage, the robot keeps moving its end-effector with the object so that the adapted model still fits the robot's inverse dynamics. The NMSE of the NN model with incremental learning has its highest peak at the end of the place stage. This could be due to the fact that it adapted to the new dynamics with the object and thus, has to re-learn the dynamics of the robot without the object. The combination of the *NN with error model learning* (purple) shows the best prediction results. Even at the beginning of the grasping stage where the robot's inverse dynamics model changes, the combined NN error model stays below a NMSE of 0.1 and continuously decreases. Similar to the incrementally learned model, the NN with error model learning has its highest NMSE at the end of the place stage, but is able to recover and adapt to the changing dynamics.

Fig. 4b shows that the pure *PEST model without adaptation* (dashed orange) yields very high prediction errors throughout the pick and place task. Similar to the batch learning experiments on real data, this emphasizes the importance of modeling nonlinear effects which the PEST

cannot cope with. However, in *combination with the pre-trained error model* (dashed green), the torque predictions improve significantly for the approach and retract stage as the regression-based error model is able to correct the prediction errors. Due to the fact that the combined PEST error model (dashed green) does not adapt during task execution, it has high prediction errors in the grasp and place stage. As expected, the combined *PEST with error model learning* (solid green) yields the best results. It is able to compensate the errors of the pure PEST model and, at the same time, adapts to the changing dynamics of the grasp and place stage. Similar to the adaptation experiments of the NN model (Fig. 4a), PEST with error model learning has the highest prediction errors at the end of the place task. Apart from this, the model remains below a NMSE of 0.1 and even decreases below a NMSE of 10^{-2} after grasping the object.

Fig. 4c shows the comparison of all three adaptation models. Again, it can be seen that the models are able to adapt to the changing dynamics of the pick and place task. The highest NMSE of all models occurs while placing the object on the table. This could be due to the abrupt change of the robot's dynamics or because the models completely adapted to the dynamics with the object. The *incrementally learned NN model* (blue), however, yields much higher prediction errors on average than the *adaptation models with error model learning* (purple and green). This is especially true for the grasp stage where its highest prediction error exceeds a NMSE of 0.1, whereas the combined error models stay below a NMSE of 0.1. In case of error model learning, the results of the combined PEST as well as the combined NN are very similar. Both error models improve the batch model's accuracy significantly during task execution.

VI. Conclusion

In this paper, we compared three representative methods for inverse dynamics learning, the state-of-the-art PEST, the model-based DBM and the purely data-driven NN model. In simulation, PEST and DBM learn exact representations of the robot's inverse dynamics. In presence of noise, all models suffer heavily, whereby the PEST and the NN model still yield plausible prediction results regarding extrapolation. The experiments on the real data set, however, demonstrate the big gap between simulation and reality. Due to nonlinear backlash and friction effects, the PEST model performs poorly on real data, whereas the regression-based NN method still learns a reasonable inverse dynamics. We demonstrated that learning an inverse dynamics model *once* is not enough as the robot's dynamics changes while interacting with the real world. Instead, the dynamics models need to be adapted at run-time. The continuous learning experiments show that incremental learning the NN model is one way to deal with the adaptation problem. However, combining the PEST or NN with error model learning leads to even better results.

In the future, we plan to extend our experiments with complex manipulation tasks while learning the inverse dynamics models online. At the same time, we would like to perform a comprehensive hyperparameter search for the NN and error model including different network architectures and recurrent structures. Furthermore, the impact of prior knowledge could be investigated, as e.g. proposed in [22].

References

[1] J. J. Craig, *Introduction to Robotics: Mechanics and Control*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 1989.

[2] C. H. An, C. G. Atkeson, and J. M. Hollerbach, "Estimation of inertial parameters of rigid body links of manipulators," in *1985 24th IEEE Conference on Decision and Control*, pp. 990–995, Dec 1985.

[3] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning, Cambridge, MA, USA: MIT Press, Jan. 2006.

[4] B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning, Cambridge, MA, USA: MIT Press, Dec. 2002.

[5] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space," in *Proc. 17th Int. Conf. Mach. Learn. (ICML)*, pp. 1079–1086, 2000.

[6] D. Nguyen-Tuong and J. Peters, "Local gaussian process regression for real-time model-based robot control," in *2008 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 380–385, Sept 2008.

[7] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *2010 IEEE Int. Conf. on Robotics and Automation*, pp. 2677–2682, May 2010.

[8] D. Kappler, F. Meier, N. Ratliff, and S. Schaal, "A new data source for inverse dynamics learning," in *2017 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 4723–4730, IEEE, 2017.

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

[10] L. Jamone, B. Damas, and J. Santos-Victor, "Incremental learning of context-dependent dynamic internal models for robot control," in *2014 IEEE Int. Symp. Intell. Contr. (ISIC)*, pp. 1336–1341, Oct 2014.

[11] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schölkopf, "Learning inverse dynamics: a comparison," in *European Symposium on Artificial Neural Networks*, 2008.

[12] S. Ulbrich, M. Bechtel, T. Asfour, and R. Dillmann, "Learning robot dynamics with kinematic bézier maps," in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 3598–3604, Oct 2012.

[13] S. S. Haykin, *Neural networks and learning machines*. Upper Saddle River, NJ: Pearson Education, third ed., 2009.

[14] R. Featherstone and D. E. Orin, "Dynamics," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), ch. 2, pp. 35–65, Springer Berlin Heidelberg, May 2008. ISBN 978-3-540-23957-4.

[15] J. Hollerbach, W. Khalil, and M. Gautier, "Model Identification," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), ch. 2, pp. 321–344, Springer Berlin Heidelberg, May 2008. ISBN 978-3-540-23957-4.

[16] J. S. de la Cruz, D. Kulic, and W. Owen, "Learning inverse dynamics for redundant manipulator control," in *2010 Int. Conf. on Autonomous and Intelligent Systems, AIS 2010*, pp. 1–6, June 2010.

[17] J. M. Hollerbach, "A recursive lagrangian formulation of maniputator dynamics and a comparative study of dynamics formulation complexity," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 10, pp. 730–736, Nov 1980.

[18] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 IRE WESCON Convention Record, Part 4*, (New York), pp. 96–104, Institute of Radio Engineers, 8 1960.

[19] S. Schaal, "The sl simulation and real-time control software package," tech. rep., University of Southern California, Los Angeles, CA, 2009.

[20] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computation scheme for mechanical manipulators," *J. Dyn. Syst. Meas. Contr.*, vol. 102, 06 1980.

[21] F. Meier, D. Kappler, and S. Schaal, "Online learning of a memory for learning rates," in *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 2425–2432, IEEE, 2018.

[22] J. K. Gupta, K. Menda, Z. Manchester, and M. Kochenderfer, "A general framework for structured learning of mechanical systems," *arXiv preprint arXiv:1902.08705*, 2019.