Multibody System Dynamics
https://doi.org/10.1007/s11044-024-10024-2

RESEARCH

Data-driven inverse dynamics modeling using neural-networks and regression-based techniques

Maciej Pikuliński[1] · Paweł Malczyk[1] · Ronald Aarts[2]

Received: 9 November 2023 / Accepted: 9 August 2024

Abstract
This research proposes a novel approach for the residual modeling of inverse dynamics employed to control a real robotic device. Specifically, we use techniques based on linear regression for residual modeling while a nominal model is discovered by physics-informed neural networks such as the Lagrangian Neural Network and the Feedforward Neural Network. We introduce an efficient online learning mechanism for the residual models that utilizes rank-one updates based on the Sherman–Morrison formula. This enables faster adaptation and updates to effects not captured by the neural networks. While the time complexity of updating the model is comparable to other successful learning methods, the method excels in prediction complexity, which depends solely on the model dimension. We propose two online learning strategies: a weighted approach that gradually diminishes the influence of past measurements on the model, and a windowed approach that sharply excludes the oldest data from impacting the model. We explore the relationship between these strategies, offering recommendations for parameter selection and practical application. Special attention is given to optimizing the computation time of the weighted approach when recomputation techniques are implemented, which results in comparable or even lower execution times of the weighted controller than the windowed one. Additionally, we assess other methods, such as the Woodbury identity, QR decomposition, and Cholesky decomposition, which can be implicitly used to update the model. We empirically validate our approach using real data from a 2-degrees-of-freedom flexible manipulator, demonstrating consistent improvements in feedforward controller performance.

Keywords Feedforward control · Inverse dynamics · Data-driven · Neural networks · Online learning · Error modelling

✉ M. Pikuliński
maciej.pikulinski@pw.edu.pl

P. Malczyk
pawel.malczyk@pw.edu.pl

R. Aarts
r.g.k.m.aarts@utwente.nl

[1] Institute of Aeronautics and Applied Mechanics, Faculty of Power and Aeronautical Engineering, Warsaw University of Technology, Nowowiejska 24, Warsaw, 00-665, Poland

[2] Applied Mechanics and Data Analysis, University of Twente, P.O. Box 217, Enschede, 7500 AE, The Netherlands

Published online: 06 September 2024

1 Introduction

Effective and safe control of multibody systems, such as manipulators or other robotic appliances, requires a rigorous and systematic approach grounded in well-studied traditional control theory. However, the constant need to make control systems more efficient, accurate, and robust often leads engineers to tackle complex problems. Detailed modeling of phenomena and analyzing multiobjective optimization problems with various constraints are necessary. Additionally, optimizing the computational cost of already efficient control algorithms can be a challenging task. The question arises whether the investment of extensive effort to achieve further improvements is economically justifiable, considering the Pareto principle, which suggests that most gains have already been achieved through primary adjustments.

In this context, the recent development of data-driven techniques sheds light on new paths to solve the mentioned issues. The methods tend to automate processes, which allows moving human labor into areas where significant gains have not been achieved. Moreover, their approximation abilities might help develop more sophisticated models or surrogate ones better conditioned for specific computations. These broader perspectives highlight the potential of leveraging data-driven techniques to enhance traditional control methodologies [1].

Consequently, this work focuses on the modeling aspect, which constitutes the core of model-based control. Specifically, we explore using regression-based models in a feedforward control scheme to minimize the error of the controller based on physics-informed neural networks. An efficient online learning method is used to adapt the model. A 2-degrees-of-freedom (2 DOF) manipulator with flexure joints serves as a demonstrative platform for validating the proposed approach.

The foundation of our research lies in the use of physics-informed neural networks in a feedforward controller. This approach combines the strengths of the Lagrangian Neural Network (LNN) and the Feedforward Neural Network (FNN). The LNN is employed to model the conservative forces, while the FNN's primary role is to identify and predict discrepancies between the total forces required and those produced by the LNN – namely, non-conservative forces. However, it is important to note that neither of the networks can provide an exact model of these forces.

This limitation naturally leads us to the concept of residual modeling to learn missing physics [2]. The primary objective of this work is to compute an online-learned error model for the neural-network-based feedforward controller. Subsequently, we aim to enhance the controller's performance by adding the predicted error to controller's original output. We employ weighted recursive least-squares with regularization to create a linear model mapping system configurations to control signals or forces to achieve this. The implicit linearity of this Data-Driven Inverse Dynamics (DID) error model enables efficient updates and adaptations, outperforming neural network retraining, which takes place at a slower pace in the assumed control structure. Updates are realized based on the Sherman–Morrison formula, but usage of the Woodbury formula, QR, and Cholesky decompositions are also discussed.

As the inverse dynamics solution can significantly help in the control of a device [3], many approaches to build the required model have been proposed, starting from white-box approaches assuming the structure of the equations and estimating parameters only to solely data-driven based methods, categorized as black-box ones, approximating highly nonlinear solutions with rather uninterpretable internals [4]. While it is not feasible to review all these methods [5, 6], we can examine those closely related to the proposed approach.

One noteworthy family of methods is Locally Weighted Learning (LWL) [7, 8], which approximate nonlinear functions using piecewise linear models. These methods serve as a reference for comparing the efficiency of adaptive algorithms [9]. They have some advantages over the methods used in this work, as they retain a long memory of previously

encountered configurations and their dynamics. However, this feature may increase memory requirements and computational costs. Work proposing incorporating prior knowledge into methods from the discussed family showed rather low generalization capabilities [10], also referenced as evidence for data-driven strategies extrapolating poorly in general [9].

Two other notable categories of methods are based on Gaussian Process Regression (GPR) [11] and Support Vector Regression [12]. These methods can potentially achieve higher accuracy compared to LWL [10]. However, when applied in real-time control settings, these methods are often found to be computationally more demanding than LWL [3, 6]. It is worth noting that incorporating the locality concept from LWL can enable GPR to work in real-time online learning with rank-1 updates [13]. Alternatively, strict window time constraints, such as those used in [3], are necessary when employing these methods for online inverse dynamics error learning. Also, there were attempts to include prior physics knowledge in these methods [14].

The most recently emerging methods are based on neural networks – from presenting promising results obtained by the application of simplest Extreme Learning Machines [15] to employing classical FNNs for error learning, but with special care taken about building the training set [16]. Moreover, Physics-Informed Neural Networks are gaining popularity as they offer a framework for integrating physical principles into the learning process or networks' structure, enhancing the accuracy and interpretability of data-driven inverse dynamics models [4]. Other remarkable concepts propose introducing meta-training [17] to strengthen the adaptability of the models.

Hitzler et al. compared two approaches for improving inverse dynamics solutions in control [9]. One involved adapting the entire model based either on classical parameter estimation or neural networks, while the other focused on error reduction using online learned neural networks. Their analysis revealed that the former method was more effective at minimizing errors, but it was noted that incremental learning of neural networks was rather slow.

To the best of the authors' knowledge, it is challenging to find works that employ linear models for inverse dynamics error modeling. As demonstrated in the references mentioned above, various methods have been adapted for online learning. However, when it comes to error learning, neural networks are primarily discussed recently as the base model or as the error learning tool. On the other hand, linear mappings have been successfully applied to error modeling in the context of inverse kinematics problems [18].

Therefore, we find the investigation of the tools combined in our proposition particularly interesting. The primary novelties and importance of the paper may be summarized as follows:

1. We utilize linear regression techniques to model the error of the feedforward controller based on neural networks and apply the error prediction to enhance control performance.
2. Leveraging the simplicity of the linear model, we employ efficient rank-1 Sherman-Morrison updates for the online learning of these error models.
3. Our empirical results consistently demonstrate significant improvements in controlling a 2 DOF manipulator with flexure joints.
4. We provide practical guidelines for selecting parameters and methods for online error learning, supported by hands-on experience.

The article is organized as follows. Section 2 offers a concise overview of feedforward control foundations. This section serves as a general introduction to feedforward control, setting the stage for the subsequent discussion on novel feedforward controllers based on physics-informed neural networks, which we want to improve by introducing error learning.

Moving to the core of our study, Sect. 3 begins with a discussion on error modeling. We propose a solution using weighted recursive least-squares with regularization. Within

this section, we also provide a comprehensive derivation of the weighted recursive DID algorithm in Sect. 3.3, followed by its windowed counterpart in Sect. 3.4. These sections lay the theoretical foundation for our method.

Section 4 transits from theory to practical implementation and empirical validation. We start by introducing the real robotic device under study and proceed to discuss key implementation concepts. In this section, we show the parameter selection process and conclude with the empirical validation of the method's performance using real-world data in Sect. 4.6.

Finally, Sect. 5 comprehensively summarizes our key contributions and findings and outlines potential avenues for future research.

2 Feedforward control and neural networks

2.1 Feedforward control

Feedforward control is a powerful technique to improve tracking performance of a manipulator by predicting the required input commands to achieve a desired output trajectory. The feedforward controller is designed to account for the dynamic properties of the system and provide improved control compared to purely feedback-based controllers.

Several techniques for designing feedforward controllers include model predictive control and optimal control. These methods aim to minimize the prediction error between the desired and actual behavior of the system, sometimes achieving more general, abstract goals simultaneously. However, one might decompose these approaches into two stages, i.e., finding an optimal trajectory (motion) of the robotic system and solving the inverse dynamics problem.

Once the trajectory optimization is delegated to another module, which provides input for the feedback controller and the feedforward one, the latter might be considered an inverse dynamics solver.

2.2 Inverse dynamics

The feedforward controller can generate the appropriate input commands by computing the joint torques or forces required to achieve the desired trajectory, which is commonly known as the inverse dynamics (ID). It follows from the equation of motion for a robotic manipulator, which can typically be written like

$$\boldsymbol{F} = \mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{Q}(\boldsymbol{q}) + \boldsymbol{F}_{\mathrm{nc}}, \tag{1}$$

where $\boldsymbol{q}$ is the vector with a set of independent generalized coordinates with velocities $\dot{\boldsymbol{q}}$ and accelerations $\ddot{\boldsymbol{q}}$. The generalized forces $\boldsymbol{F}$ are applied at these coordinates. Mass and inertia are described by the (generalized) mass matrix $\mathbf{M}(\boldsymbol{q})$ and matrix $\mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ accounts for Coriolis and centrifugal contributions. The term $\boldsymbol{Q}(\boldsymbol{q})$ includes gravity and other forces that depend on the configuration $\boldsymbol{q}$ only. Finally, all other force contributions like friction and contact forces are described with $\boldsymbol{F}_{\mathrm{nc}}$. The subscript "nc" indicates these generalized forces are in general non-conservative, whereas the other terms in the right-hand side of equation (1) are conservative and denoted with $\boldsymbol{F}_{\mathrm{c}}$ in the sequel.

In general, the number of independent generalized coordinates $\boldsymbol{q}$ can be larger or smaller than the number of actuators contributing to the force $\boldsymbol{F}$, resulting in an underactuated or redundantly actuated system, respectively. In this paper, we do not consider such systems,

but only fully actuated systems. It also implies that we focus on the rigid body dynamics (RBD) of the manipulator that, e.g., excludes internal dynamic behavior arising from finite link stiffness. To implement the feedforward control, a model-based approach can be used to derive a white-box model representing the equation of motion (1) of the multibody system and estimate the relevant model parameters [19, 20]. Alternatively, in a data-driven approach, a black-box model can be identified purely from data, e.g., using machine learning techniques, as will be outlined in the next subsection.

2.3 Physics-informed neural networks

Instead of using RBD to derive the equation of motion (1), Neural Networks (NN) can estimate the required feedforward driving force $\boldsymbol{F}$ as a function of any specified configuration, velocity and acceleration $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$. A so-called Feedforward Neural Network (FNN) can be trained purely from data for this purpose [21]. Such FNN is a completely black-box model that does not assume any knowledge about the system dynamics and its parameters. Consequently, large amounts of data may be needed and care has to be taken to avoid overfitting, otherwise the model outputs are possibly incorrect for operating conditions that were not sufficiently excited in the training data.

To mitigate this risk, physics informed neural networks (PINN) are being researched, where the training is constrained to a predefined physical law. Lutter et al. [4, 22] propose a Lagrangian Neural Network (LNN), or Deep Lagrangian Network (DeLaN), to incorporate the Lagrangian dynamics into a neural network. For this purpose, the equation of motion (1) is rewritten as

$$\boldsymbol{F} = \boldsymbol{F}_{\mathrm{c}} + \boldsymbol{F}_{\mathrm{nc}}, \tag{2}$$

where $\boldsymbol{F}_{\mathrm{c}}$ represents explicitly the conservative part in the total actuator force vector $\boldsymbol{F}$. Exploiting the structure of the RBD model for the terms in the right-hand side of equation (1), the conservative contribution to the force vector $\boldsymbol{F}_{\mathrm{c}}$ can be written as

$$\boldsymbol{F}_{\mathrm{c}} = \mathbf{L}(\boldsymbol{q})\mathbf{L}^{T}(\boldsymbol{q})\,\ddot{\boldsymbol{q}} + \frac{1}{2}\dot{\boldsymbol{q}}^{T}\frac{\partial(\mathbf{L}(\boldsymbol{q})\mathbf{L}^{T}(\boldsymbol{q}))}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} + \frac{\partial V(\boldsymbol{q})}{\partial \boldsymbol{q}}, \tag{3}$$

where the position-dependent potential energy $V(\boldsymbol{q})$ is introduced to compute the conservative forces $\boldsymbol{Q}(\boldsymbol{q})$ from its derivative. The lower triangular matrix $\mathbf{L}$ is the Cholesky decomposition of the mass matrix, i.e., $\mathbf{M} = \mathbf{L}\mathbf{L}^{T}$. The diagonal terms of this matrix $\mathbf{L}$ have to be positive to guarantee that a physical meaningful mass matrix $\mathbf{M}$ is obtained which is symmetric and positive definite.

In general, the non-conservative forces $\boldsymbol{F}_{\mathrm{nc}}$ cannot be ignored to optimize the performance of the feedforward control. The control structure in Fig. 1 accounts for both terms in equation (2) by combining the physics-informed LNN with a black-box FNN. More specifically, the outputs of the LNN are estimates of the potential energy $\widetilde{V}$ and matrix $\widetilde{\mathbf{L}}$, where the tilde symbol is to designate estimated parameters and values delivered by the discussed models. The lower triangular matrix $\mathbf{L}$ is split into the off-diagonal $\widetilde{\boldsymbol{l}}_o$ and positive diagonal $\widetilde{\boldsymbol{l}}_d$ terms. This is accomplished by implementing the output layer for $\widetilde{\boldsymbol{l}}_d$ with, e.g., a ReLu or rectifier activation, which means that the neuron's output equals its input for non-negative values and zero otherwise. All estimates of the LNN are functions of the positions $\boldsymbol{q}$ only, as can be understood from equation (3). Automatic differentiation [23] is used to compute the derivatives of $\widetilde{V}$ and $\widetilde{\mathbf{L}}$ with respect to $\boldsymbol{q}$ that appear in equation (3).

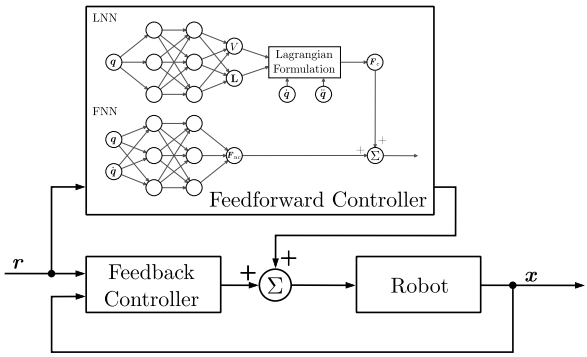

Fig. 1 Control structure of the system with a feedforward controller combining an LNN with an FNN

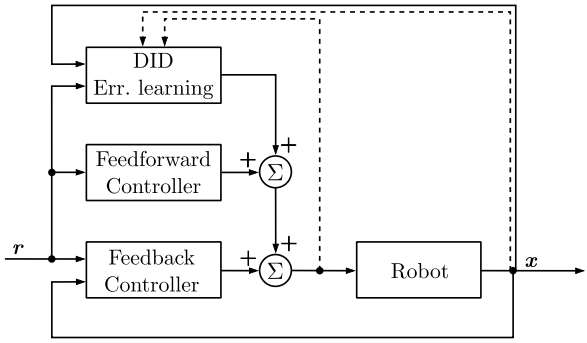

Fig. 2 Control system structure with the DID error learning

This LNN is trained first with experimental data to minimize the mean squared error between the estimated forces $\widetilde{\boldsymbol{F}}_{\mathrm{c}}$ and measured forces [24]. It is expected that in this way, the LNN accounts for the major part of the force according to equation (2). Next, the FNN is trained with the residual representing the mismatch between the LNN output $\widetilde{\boldsymbol{F}}_{\mathrm{c}}$ and measured forces, where it is assumed the forces $\boldsymbol{F}_{\mathrm{nc}}$ only depend on positions $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$. We prefer this successive training of the NNs to avoid the dependency on a penalization factor that is used in simultaneous training [25]. To compute the total feedforward control input, the outputs of both NNs are added as indicated in Fig. 1.

3 Online learning of the error model

3.1 Error modeling

Even though two neural networks, trained offline, are employed to predict the control forces for a manipulator with flexure joints considered herein, there is still a place for control quality improvements. We devise a recursive strategy to minimize the discrepancy between the real control needed and estimates from the LNN and FNN methods. The procedure, called DID error discovery, works online and updates a model that predicts the mismatch as new data becomes available.

This error-modeling element is integrated into the control structure, as illustrated in Fig. 2, in a manner akin to the direct learning architecture described in [26]. In this illustration, its predictive action is represented by a solid line, while a dashed line indicates

the information flow used in the model learning phase. We predict a residual error $\boldsymbol{e} \in \mathcal{R}^n$ (n – number of degrees of freedom), defined in Eq. (4), between real control signals $\boldsymbol{F}$ delivered to the fully-actuated robotic system and the control forces $\widetilde{\boldsymbol{F}}$, in the form of Eq. (2), predicted by simultaneous action of LNN and FNN.

$$\boldsymbol{e}_j = \boldsymbol{F}_j - \widetilde{\boldsymbol{F}}_j = \boldsymbol{g}(\boldsymbol{x}_j, \boldsymbol{r}_j, \boldsymbol{r}_{j+1}), \quad j = 1, \ldots, k. \tag{4}$$

The term $\boldsymbol{e}$ is defined in discrete-time and captures errors in feedforward control – inaccuracies in the dynamic parameters predicted by LNN and FNN, e.g., vibrations, couplings, friction, and sensor noise, among the others. It is described, in general, by a nonlinear function $\boldsymbol{g}$ that maps from system state $\boldsymbol{x}_j = [\boldsymbol{q}_j^T \quad \dot{\boldsymbol{q}}_j^T]^T \in \mathcal{R}^{2n}$, the desired system state $\boldsymbol{r}_j = [\bar{\boldsymbol{q}}_j^T \quad \dot{\bar{\boldsymbol{q}}}_j^T]^T \in \mathcal{R}^{2n}$ and the next desired system state $\mathbf{r}_{j+1}$ (indeed, the feedforward does not depend on the current state, i.e., $\widetilde{\boldsymbol{F}}_j = \widetilde{\boldsymbol{F}}_j(\boldsymbol{r}_j, \boldsymbol{r}_{j+1})$) to the errors in the system control input $\boldsymbol{e}_j$.

For the sake of simplicity and computational efficiency, we propose to approximate $\boldsymbol{g}$ in a linear form using a set of arguments reduced to $\mathbf{x}_j$ and $\mathbf{r}_{j+1}$

$$\boldsymbol{e}_j = \mathbf{D}\boldsymbol{z}_j, \quad \text{where} \quad \boldsymbol{z}_j = \left[\boldsymbol{x}_j^T \quad \boldsymbol{r}_{j+1}^T\right]^T \in \mathcal{R}^{4n}, \tag{5}$$

where matrix $\mathbf{D} \in \mathcal{R}^{m \times 4n}$ represents linearized error model. Omitting the $\boldsymbol{r}_j$ argument does not compromise the generality of our approach. From a predictive perspective, our primary concern lies in the outcome $\boldsymbol{e}_j$, rather than in the decomposition into $\boldsymbol{F}$ and $\widetilde{\boldsymbol{F}}$. Furthermore, in an ideal setting when the prediction of the residue is exact, the proposed additional controller brings the feedback controller action to zero, which also means that $\boldsymbol{x}_j \to \boldsymbol{r}_j$.

Another point concerning the choice of arguments is that we want the predictions to be based on the actual state of the system $\boldsymbol{x}_j$. This option increases the likelihood of successfully minimizing the feedback controller output. Indeed, the success does not only depend on adjusting the feedforward model with the error model, but also on responding to unpredictable disturbances. Ultimately, such a design enables the controller to introduce corrections related to tracking errors.

The mentioned ideal setting is crucial when building and updating the model. We assume direct access to $\boldsymbol{x}_{j-1}$, $\boldsymbol{x}_j$, $\boldsymbol{F}_j$, and $\widetilde{\boldsymbol{F}}$, where $\widetilde{\boldsymbol{F}} = \widetilde{\boldsymbol{F}}(\cdot, \cdot)$ is considered a function of two consecutive states. When constructing the training dataset as follows

$$\mathcal{D}_k = \left\{ \left(\boldsymbol{e}_j, \boldsymbol{z}_j\right) : \boldsymbol{e}_j = \boldsymbol{F}_{j-1} - \widetilde{\boldsymbol{F}}\left(\boldsymbol{x}_{j-1}, \boldsymbol{x}_j\right),\ \boldsymbol{z}_j = \left[\boldsymbol{x}_{j-1}^T \quad \boldsymbol{x}_j^T\right]^T,\ j = 1, \ldots, k \right\}, \tag{6}$$

we mimic ideal tracking that aligns with our overall objective, specifically by constructing the training dataset from the measurements as if the desired state $\boldsymbol{r}_{j+1} \leftarrow \boldsymbol{x}_j$ is exactly reached from the actual state $\boldsymbol{x}_j \leftarrow \boldsymbol{x}_{j-1}$. Consequently, error learning does not relay on feedback control accuracy.

In reference to [16], apart from this indirect approach of forming the training data, there exists an alternative, direct approach that builds the dataset using the desired trajectory $\boldsymbol{r}_j$, $\boldsymbol{r}_{j+1}$, and the feedback controller outcome. However, this alternative dataset did not demonstrate superior results in our preliminary tests.

The following subsections present a detailed derivation of the proposed DID approach. The formulas used here are adapted from [27], but the underlying computations differ. The DID algorithm heavily exploits the weighted, regularized, recursive least-squares approach with a forgetting factor and can employ a sliding-window approach to reduce memory requirements.

3.2 Weighted least-squares with regularization

A quantity $\left(\boldsymbol{e}_j, \boldsymbol{z}_j\right)$ is called a snapshot pair. Given the set $\mathcal{D}_k$ of k snapshot pairs collected in the experiment as the robot state evolves in time, we reshape the measurements to form data matrices $\mathbf{E}_k$ and $\mathbf{Z}_k$.

$$\mathbf{E}_k = \begin{bmatrix} | & | & & | \\ \boldsymbol{e}_1 & \boldsymbol{e}_2 & \dots & \boldsymbol{e}_k \\ | & | & & | \end{bmatrix}, \quad \mathbf{Z}_k = \begin{bmatrix} | & | & & | \\ \boldsymbol{z}_1 & \boldsymbol{z}_2 & \dots & \boldsymbol{z}_k \\ | & | & & | \end{bmatrix}, \tag{7}$$

which have different dimensions, i.e., $n \times k$ and $4n \times k$, respectively. The proposed DID algorithm seeks the best-fit operator $\mathbf{D}_k$ that relates two snapshot matrices by penalizing the values of the entries in matrix $\mathbf{D}_k$. The matrix $\mathbf{D}_k$ is found by minimizing the following performance index:

$$J_k(\mathbf{D}_k) = \sum_{i=1}^{k} \rho^{k-i} ||\boldsymbol{e}_i - \mathbf{D}_k \boldsymbol{z}_i||^2 + \alpha ||\mathbf{D}_k||_F^2, \quad \text{where} \quad 0 < \rho \leq 1 \quad \text{and} \quad \alpha \geq 0. \tag{8}$$

The scalar ρ is the weighting factor that focuses more on recent snapshots and gradually forgets the older snapshots. The second component in equation (8) is a penalty term to shrink the elements of $\mathbf{D}_k$ towards zero (the symbol $||.||_F$ is the Frobenius norm). The quantity α is a small, fixed penalty factor that improves the numerical robustness of the formulation. One reason for using a penalty method in equation (8) is that we always get a unique least-squares solution, even when the data matrices are not full rank. When the data quantities are of full rank, the extra term can improve the numerical conditioning appearing in the normal equations, and resulting in better numerical behavior.

Let us define $\rho = \sigma^2$, where $0 < \sigma \leq 1$ and rewrite the cost function in (8) as

$$J_k(\mathbf{D}_k) = \sum_{i=1}^{k} ||\sigma^{k-i} \boldsymbol{e}_i - \mathbf{D}_k \sigma^{k-i} \boldsymbol{z}_i||^2 + \alpha ||\mathbf{D}_k||_F^2. \tag{9}$$

If we introduce the scaled versions of the matrices defined in Eq. (7), as

$$\hat{\mathbf{E}}_k = \begin{bmatrix} | & | & & | \\ \hat{\boldsymbol{e}}_1 & \hat{\boldsymbol{e}}_2 & \dots & \hat{\boldsymbol{e}}_k \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \sigma^{k-1}\boldsymbol{e}_1 & \sigma^{k-2}\boldsymbol{e}_2 & \dots & \boldsymbol{e}_k \\ | & | & & | \end{bmatrix}, \tag{10}$$

$$\hat{\mathbf{Z}}_k = \begin{bmatrix} | & | & & | \\ \hat{\boldsymbol{z}}_1 & \hat{\boldsymbol{z}}_2 & \dots & \hat{\boldsymbol{z}}_k \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \sigma^{k-1}\boldsymbol{z}_1 & \sigma^{k-2}\boldsymbol{z}_2 & \dots & \boldsymbol{z}_k \\ | & | & & | \end{bmatrix}, \tag{11}$$

then the cost function (8) can be rewritten as

$$J_k(\mathbf{D}_k) = ||\hat{\mathbf{E}}_k - \mathbf{D}_k \hat{\mathbf{Z}}_k||_F^2 + \alpha ||\mathbf{D}_k||_F^2. \tag{12}$$

The minimum-norm solution to the problem (12) is given as

$$\mathbf{D}_k = \hat{\mathbf{E}}_k \hat{\mathbf{Z}}_k^T (\hat{\mathbf{Z}}_k \hat{\mathbf{Z}}_k^T + \alpha \mathbf{I})^{-1}, \tag{13}$$

where $\mathbf{I}$ is the identity matrix. When $\alpha = 0$, then $\mathbf{D}_k = \hat{\mathbf{E}}_k \hat{\mathbf{Z}}_k^+$, where $\hat{\mathbf{Z}}_k^+ = \hat{\mathbf{Z}}_k^T (\hat{\mathbf{Z}}_k \hat{\mathbf{Z}}_k^T)^{-1}$ denotes the Moore–Penrose pseudoinverse of $\hat{\mathbf{Z}}_k$. By using (13), one finds a matrix $\mathbf{D}_k$, which

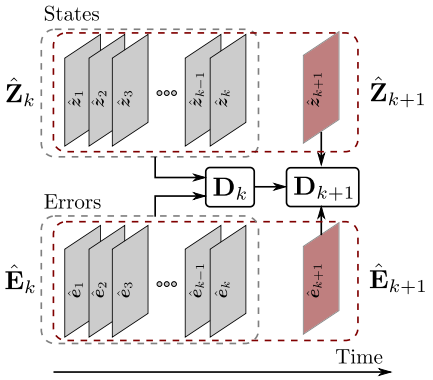

Fig. 3 Recursive DID error discovery

can be employed to predict the mismatch as defined in equation (5). All snapshots are processed in one batch. The process may be compute- and data-inefficient, because the method requires storing all past data measurements to find the inverses of large-size matrices. Therefore, in the following subsection, we devise recursive techniques similar to those used in the recursive least-squares estimation [28] that demonstrate much smaller compute and memory requirements.

3.3 Weighted recursive DID

Let us assume that we have already calculated the characteristic matrix $\mathbf{D}_k$ for a given data organized in the form of the matrices $\hat{\mathbf{E}}_k$ and $\hat{\mathbf{Z}}_k$ (cf. equation (10)), as depicted in Fig. 3. As time runs, a new pair of snapshots $(\hat{z}_{k+1}, \hat{e}_{k+1})$ becomes available. At time $k+1$, the matrix $\mathbf{D}_k$ may be updated to find $\mathbf{D}_{k+1}$, using the information from time k, and new snapshot pair $(\hat{z}_{k+1}, \hat{e}_{k+1})$ at time $k+1$. Therefore, $\mathbf{D}_{k+1}$ will map the dataset $\hat{\mathbf{Z}}_{k+1} = \begin{bmatrix} \sigma\hat{\mathbf{Z}}_k & \hat{z}_{k+1} \end{bmatrix}$ into $\hat{\mathbf{E}}_{k+1} = \begin{bmatrix} \sigma\hat{\mathbf{E}}_k & \hat{e}_{k+1} \end{bmatrix}$.

The derivation is started by observing that the result in (13) can be expressed as

$$\mathbf{D}_k = \hat{\mathbf{E}}_k \hat{\mathbf{Z}}_k^T (\hat{\mathbf{Z}}_k \hat{\mathbf{Z}}_k^T + \alpha\mathbf{I})^{-1} = \mathbf{Q}_k \mathbf{P}_k, \tag{14}$$

where $\mathbf{Q}_k$ is a $m \times 4n$ matrix and $\mathbf{P}_k$ is a $4n \times 4n$ quantity

$$\mathbf{Q}_k = \hat{\mathbf{E}}_k \hat{\mathbf{Z}}_k^T, \tag{15}$$

$$\mathbf{P}_k = (\hat{\mathbf{Z}}_k \hat{\mathbf{Z}}_k^T + \alpha\mathbf{I})^{-1}. \tag{16}$$

The matrix $\mathbf{P}_k$ is well defined. The term $\alpha\mathbf{I}$ ensures that the matrix $\hat{\mathbf{Z}}_k \hat{\mathbf{Z}}_k^T + \alpha\mathbf{I}$ is non-singular, even when the $\hat{\mathbf{Z}}_k \hat{\mathbf{Z}}_k^T$ matrix is rank deficient, which makes the least-squares procedure presented herein numerically robust. The quantity $\mathbf{P}_k$ is also symmetric and strictly positive definite.

We start the derivation of the recursive algorithm by expressing the factors $\mathbf{Q}_{k+1}$, $\mathbf{P}_{k+1}$ involved in the matrix $\mathbf{D}_{k+1} = \mathbf{Q}_{k+1}\mathbf{P}_{k+1}$ in relation to the quantities $\mathbf{Q}_k$, $\mathbf{P}_k$:

$$\begin{aligned} \mathbf{Q}_{k+1} &= \hat{\mathbf{E}}_{k+1} \hat{\mathbf{Z}}_{k+1}^T = \begin{bmatrix} \sigma\hat{\mathbf{E}}_k & \hat{e}_{k+1} \end{bmatrix} \cdot \begin{bmatrix} \sigma\hat{\mathbf{Z}}_k & \hat{z}_{k+1} \end{bmatrix}^T \\ &= \sigma^2 \hat{\mathbf{E}}_k \hat{\mathbf{Z}}_k^T + \hat{e}_{k+1} \hat{z}_{k+1}^T = \rho\mathbf{Q}_k + \hat{e}_{k+1} \hat{z}_{k+1}^T. \end{aligned} \tag{17}$$

Similarly, we came up with the following formula:

$$\mathbf{P}_{k+1}^{-1} = \hat{\mathbf{Z}}_{k+1}\hat{\mathbf{Z}}_{k+1}^T + \alpha\mathbf{I} = \begin{bmatrix} \sigma\hat{\mathbf{Z}}_k & \hat{z}_{k+1} \end{bmatrix} \cdot \begin{bmatrix} \sigma\hat{\mathbf{Z}}_k & \hat{z}_{k+1} \end{bmatrix}^T + \alpha\mathbf{I} = \rho\mathbf{P}_k^{-1} + \hat{z}_{k+1}\hat{z}_{k+1}^T. \quad (18)$$

The formulas in (17) and (18) show that given $\mathbf{Q}_k$ and $\mathbf{P}_k^{-1}$, one can calculate the quantities at time $k+1$ by using simple rank-1 updates. The matrix $\mathbf{D}_{k+1}$ at time $k+1$ is generated then in the form:

$$\mathbf{D}_{k+1} = \mathbf{Q}_{k+1}\mathbf{P}_{k+1} = (\rho\mathbf{Q}_k + \hat{\boldsymbol{e}}_{k+1}\hat{z}_{k+1}^T) \cdot (\rho\mathbf{P}_k^{-1} + \hat{z}_{k+1}\hat{z}_{k+1}^T)^{-1}. \quad (19)$$

Now, the recursive formula that evaluates $\mathbf{Q}_{k+1}$ from $\mathbf{Q}_k$ is available in an explicit form. Finding $\mathbf{P}_{k+1}$ from $\mathbf{P}_k$ requires computing the inverse directly, which is a numerically expensive operation. We employ the Sherman–Morrison matrix inversion formula [29] to make the calculations efficient.

The Sherman–Morrison formula computes the inverse of the sum of an invertible matrix $\mathbf{A}$ and the outer product of two column vectors $\boldsymbol{u}$ and $\boldsymbol{v}$. The matrix $\mathbf{A} + \boldsymbol{u}\boldsymbol{v}^T$ is invertible if and only if $1 + \boldsymbol{v}^T\mathbf{A}\boldsymbol{u} \neq 0$. In this case, the inverse is given as

$$(\mathbf{A} + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\boldsymbol{u}\boldsymbol{v}^T\mathbf{A}^{-1}}{1 + \boldsymbol{v}^T\mathbf{A}^{-1}\boldsymbol{u}}. \quad (20)$$

We can apply the Sherman–Morrison lemma (20) to equation (18) to get

$$\mathbf{P}_{k+1} = (\rho\mathbf{P}_k^{-1} + \hat{z}_{k+1}\hat{z}_{k+1}^T)^{-1} = \frac{\mathbf{P}_k}{\rho} - \gamma_{k+1}\frac{\mathbf{P}_k}{\rho}\hat{z}_{k+1}\hat{z}_{k+1}^T\frac{\mathbf{P}_k}{\rho}, \quad (21)$$

where

$$\gamma_{k+1} = \frac{1}{1 + \hat{z}_{k+1}^T\frac{\mathbf{P}_k}{\rho}\hat{z}_{k+1}}. \quad (22)$$

By going back with (21) and (22) to (19), we can develop a formula for the update in the following form:

$$\mathbf{D}_{k+1} = \mathbf{D}_k + \gamma_{k+1}(\hat{\boldsymbol{e}}_{k+1} - \mathbf{D}_k\hat{z}_{k+1})\hat{z}_{k+1}^T\frac{\mathbf{P}_k}{\rho}. \quad (23)$$

The formula (23) simply shows that the updated model $\mathbf{D}_{k+1}$ is a sum of the of the model $\mathbf{D}_k$ and a correction term proportional to the prediction error $\hat{\boldsymbol{e}}_{k+1} - \mathbf{D}_k\hat{z}_{k+1}$. This quantity represents the error of fitting new data using the previous model $\mathbf{D}_k$. It is also clear that we have eliminated the necessity of making explicit matrix inversions and improved the computational efficiency.

The recursive algorithm described above requires initial values. In particular, one needs the matrices $\mathbf{P}_k$ and $\mathbf{D}_k$ to implement the recursive formulas shown in (21) and (23). We initialize the algorithm by collecting enough snapshots so that $\hat{\mathbf{Z}}_k$ has full row rank, and then find $\mathbf{P}_k$ and $\mathbf{D}_k$ directly from (16), (13).

The DID method with fixed regularization and forgetting factor is summarized below.

1. Collect k snapshot pairs $(z_j, \boldsymbol{e}_j)$, $j = 1, \ldots, k$, where $k > 4n$ is large enough so that $\hat{\mathbf{Z}}_k$ has full row rank.
2. Find $\mathbf{P}_k$ and $\mathbf{D}_k$ directly from (16), (13).
3. Calculate $\mathbf{D}_{k+1}$ from (23) and update $\mathbf{P}_{k+1}$ according to (21) and (22).

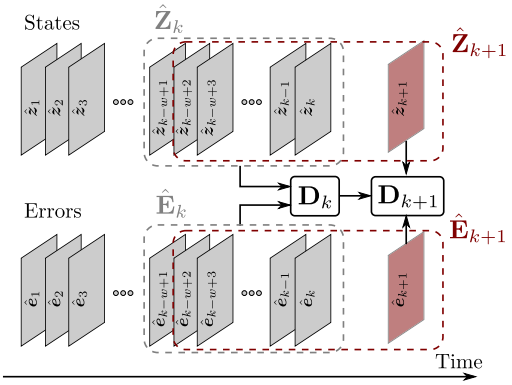

Fig. 4 Recursive DID error discovery — windowed setup

3.4 Windowed DID

In this subsection, we present a recursive algorithm that use a sliding window containing the most recent snapshots to identify the errors in ID. The weighting factor is also incorporated into the method to gradually discount the older snapshots within the window. Figure 4 presents the approach. Each iteration of the algorithm consists of two steps, namely *updating* and *downdating*. The former takes into account the current snapshot $(\hat{\boldsymbol{z}}_{k+1}, \hat{\boldsymbol{e}}_{k+1})$, and the latter removes the effect of the old data samples $(\hat{\boldsymbol{z}}_{k-w+1}, \hat{\boldsymbol{e}}_{k-w+1})$, which run out of the sliding window of size w.

Let us assume that at time k we have access to past snapshot pairs $\{(\boldsymbol{z}_j, \boldsymbol{e}_j)\}_{j=k-w+1}^{k}$ and let us organize the data in the following way:

$$\hat{\mathbf{E}}_k = \begin{bmatrix} | & | & & | \\ \hat{\boldsymbol{e}}_{k-w+1} & \hat{\boldsymbol{e}}_{k-w+2} & \dots & \hat{\boldsymbol{e}}_k \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \sigma^{w-1}\boldsymbol{e}_{k-w+1} & \sigma^{w-2}\boldsymbol{e}_{k-w+2} & \dots & \boldsymbol{e}_k \\ | & | & & | \end{bmatrix}, \tag{24}$$

$$\hat{\mathbf{Z}}_k = \begin{bmatrix} | & | & & | \\ \hat{\boldsymbol{z}}_{k-w+1} & \hat{\boldsymbol{z}}_{k-w+2} & \dots & \hat{\boldsymbol{z}}_k \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \sigma^{w-1}\boldsymbol{z}_{k-w+1} & \sigma^{w-2}\boldsymbol{z}_{k-w+2} & \dots & \boldsymbol{z}_k \\ | & | & & | \end{bmatrix}, \tag{25}$$

We consider minimizing the following cost function

$$J_k(\mathbf{D}_k) = \sum_{i=k-w+1}^{k} \rho^{k-i} ||\boldsymbol{e}_i - \mathbf{D}_k \boldsymbol{z}_i||^2 + \alpha ||\mathbf{D}_k||_F^2 = ||\hat{\mathbf{E}}_k - \mathbf{D}_k \hat{\mathbf{Z}}_k||_F^2 + \alpha ||\mathbf{D}_k||_F^2, \tag{26}$$

where $0 < \rho \leq 1$ and $\alpha \geq 0$. The process of updating takes new information at time $k+1$ to generate a matrix $\mathbf{D}'_{k+1}$ that maps the dataset $\hat{\mathbf{Z}}'_{k+1} = \begin{bmatrix} \sigma\hat{\mathbf{Z}}_k & \hat{\boldsymbol{z}}_{k+1} \end{bmatrix}$ into $\hat{\mathbf{E}}'_{k+1} = \begin{bmatrix} \sigma\hat{\mathbf{E}}_k & \hat{\boldsymbol{e}}_{k+1} \end{bmatrix}$. The strategy is very similar to that presented in Sect. 3.3. Specifically, the matrices $\mathbf{Q}_k$ and $\mathbf{P}_k$ are given by

$$\mathbf{Q}_k = \hat{\mathbf{E}}_k \hat{\mathbf{Z}}_k^T, \tag{27}$$

$$\mathbf{P}_k = (\hat{\mathbf{Z}}_k \hat{\mathbf{Z}}_k^T + \alpha \mathbf{I})^{-1}, \tag{28}$$

and we wish to compute $\mathbf{D}'_{k+1} = \mathbf{Q}'_{k+1}\mathbf{P}'_{k+1}$. The quantities $\mathbf{Q}'_{k+1}$, $\mathbf{P}'_{k+1}$ correspond to $\mathbf{Q}_k$ and $\mathbf{P}_k$ (cf. (17), (21), and (22)):

$$\mathbf{Q}'_{k+1} = \hat{\mathbf{E}}'_{k+1}\hat{\mathbf{Z}}'^{T}_{k+1} = \begin{bmatrix} \sigma\hat{\mathbf{E}}_k & \hat{\boldsymbol{e}}_{k+1} \end{bmatrix} \cdot \begin{bmatrix} \sigma\hat{\mathbf{Z}}_k & \hat{z}_{k+1} \end{bmatrix}^T = \rho\mathbf{Q}_k + \hat{\boldsymbol{e}}_{k+1}\hat{\boldsymbol{z}}^T_{k+1}. \tag{29}$$

$$\mathbf{P}'_{k+1} = (\hat{\mathbf{Z}}'_k\hat{\mathbf{Z}}'^{T}_k + \alpha\mathbf{I})^{-1} = (\rho\mathbf{P}_k^{-1} + \hat{\boldsymbol{z}}_{k+1}\hat{\boldsymbol{z}}^T_{k+1})^{-1} = \frac{\mathbf{P}_k}{\rho} - \gamma'_{k+1}\frac{\mathbf{P}_k}{\rho}\hat{\boldsymbol{z}}_{k+1}\hat{\boldsymbol{z}}^T_{k+1}\frac{\mathbf{P}_k}{\rho}, \tag{30}$$

where

$$\gamma'_{k+1} = \frac{1}{1 + \hat{\boldsymbol{z}}^T_{k+1}\frac{\mathbf{P}_k}{\rho}\hat{\boldsymbol{z}}_{k+1}}. \tag{31}$$

The update formula can be viewed as (cf. (23)):

$$\mathbf{D}'_{k+1} = \mathbf{D}_k + \gamma'_{k+1}(\hat{\boldsymbol{e}}_{k+1} - \mathbf{D}_k\hat{\boldsymbol{z}}_{k+1})\hat{\boldsymbol{z}}^T_{k+1}\frac{\mathbf{P}_k}{\rho}. \tag{32}$$

Now, we start the downdating process, which removes the oldest snapshot pair $\left(\hat{\boldsymbol{e}}_{k-w+1}, \hat{\boldsymbol{z}}_{k-w+1}\right)$ from the sliding window. At step $k+1$, we need to compute a matrix $\mathbf{D}_{k+1} = \mathbf{Q}_{k+1}\mathbf{P}_{k+1}$ that corresponds to the matrices $\mathbf{Q}'_{k+1}$ and $\mathbf{P}'_{k+1}$ evaluated in the updating step. Let us define the matrices based on scaled snapshots that exclude the oldest snapshot pair:

$$\hat{\mathbf{E}}_{k+1} = \begin{bmatrix} | & | & & | \\ \hat{\boldsymbol{e}}_{k-w+2} & \hat{\boldsymbol{e}}_{k-w+3} & \dots & \hat{\boldsymbol{e}}_{k+1} \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \sigma^{w-1}\boldsymbol{e}_{k-w+2} & \sigma^{w-2}\boldsymbol{e}_{k-w+3} & \dots & \boldsymbol{e}_{k+1} \\ | & | & & | \end{bmatrix}, \tag{33}$$

$$\hat{\mathbf{Z}}_{k+1} = \begin{bmatrix} | & | & & | \\ \hat{\boldsymbol{z}}_{k-w+2} & \hat{\boldsymbol{z}}_{k-w+2} & \dots & \hat{\boldsymbol{z}}_{k+1} \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \sigma^{w-1}\boldsymbol{z}_{k-w+2} & \sigma^{w-2}\boldsymbol{z}_{k-w+3} & \dots & \boldsymbol{z}_{k+1} \\ | & | & & | \end{bmatrix}. \tag{34}$$

Then, we should write the following relation for $\mathbf{Q}_{k+1}$:

$$\mathbf{Q}_{k+1} = \hat{\mathbf{E}}_{k+1}\hat{\mathbf{Z}}^T_{k+1} = \mathbf{Q}'_{k+1} - \hat{\boldsymbol{e}}_{k-w+1}\hat{\boldsymbol{z}}^T_{k-w+1}. \tag{35}$$

The matrix $\mathbf{P}_{k+1}$ can be evaluated by using Sherman–Morrison formula (20) to get:

$$\begin{aligned} \mathbf{P}_{k+1} &= (\hat{\mathbf{Z}}_{k+1}\hat{\mathbf{Z}}^T_{k+1} + \alpha\mathbf{I})^{-1} = ((\mathbf{P}'_{k+1})^{-1} - \hat{\boldsymbol{z}}_{k-w+1}\hat{\boldsymbol{z}}^T_{k-w+1})^{-1} \\ &= \mathbf{P}'_{k+1} - \gamma_{k+1}\mathbf{P}'_{k+1}\hat{\boldsymbol{z}}_{k-w+1}\hat{\boldsymbol{z}}^T_{k-w+1}\mathbf{P}'_{k+1}, \end{aligned} \tag{36}$$

where $\hat{\boldsymbol{e}}_{k-w+1} = \rho^w\boldsymbol{e}_{k-w+1}$ and $\hat{\boldsymbol{z}}_{k-w+1} = \rho^w\boldsymbol{z}_{k-w+1}$ and

$$\gamma_{k+1} = \frac{-1}{1 - \hat{\boldsymbol{z}}^T_{k-w+1}\mathbf{P}'_{k+1}\hat{\boldsymbol{z}}_{k-w+1}}. \tag{37}$$

Finally, the derivation leads to (cf. (23)):

$$\mathbf{D}_{k+1} = \mathbf{D}'_{k+1} + \gamma_{k+1}(\hat{\boldsymbol{e}}_{k+1} - \mathbf{D}'_{k+1}\hat{\boldsymbol{z}}_{k+1})\hat{\boldsymbol{z}}^T_{k+1}\mathbf{P}'_{k+1}. \tag{38}$$

The summary of the windowed version of the algorithm that includes forgetting factor is presented below.

1. Collect w snapshot pairs $(\boldsymbol{z}_j, \boldsymbol{e}_j)$, $j = 1, \ldots, w$, where $w \geq 4n$ is large enough and $\mathbf{Z}_k$, defined in (24), is full row rank.
2. Initialize $\mathbf{P}_k$ and $\mathbf{D}_k$ in a batch processing mode.
3. When a new snapshot pair $(\boldsymbol{z}_{k+1}, \boldsymbol{e}_{k+1})$ becomes available execute the following steps:
   (a) Perform the updating step and use (29), (30) to calculate $\mathbf{Q}'_{k+1}$ and $\mathbf{P}'_{k+1}$.
   (b) Calculate the update $\mathbf{D}'_{k+1}$ by taking (32).
   (c) Perform the downdating step and use (35), (36) to calculate $\mathbf{Q}_{k+1}$ and $\mathbf{P}_{k+1}$.
   (d) Calculate the update $\mathbf{D}_{k+1}$ by taking (38).

4 Simulation studies

4.1 Experimental data

All the results presented further are based on data obtained from a real robotic device. Specifically, the device is a fully actuated manipulator with 2 DOF, shown in Fig. 5 [30]. The top view drawing of Fig. 6 illustrates that two actuators drive the rotation of two arms, resulting in a translational end-effector motion in a horizontal plane. All joints are flexure joints, allowing relatively smooth operation with low friction and hysteresis.

Consequently, contributions from the link mass and joint stiffness dominate in the nonlinear equation of motion written in independent generalized coordinates $\boldsymbol{q}$ as in equation (2), where $\boldsymbol{F}_c$ represents the conservative part in the total actuator force vector $\boldsymbol{F}$. It is expressed in the symmetric and positive definite mass matrix $\mathbf{M}$ and potential energy V,

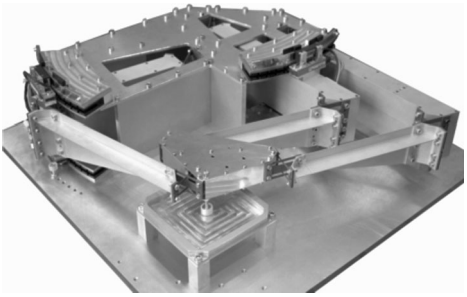

Fig. 5 Photo of the 2 DOF manipulator with flexure joints (photo by Ger Folkersma)

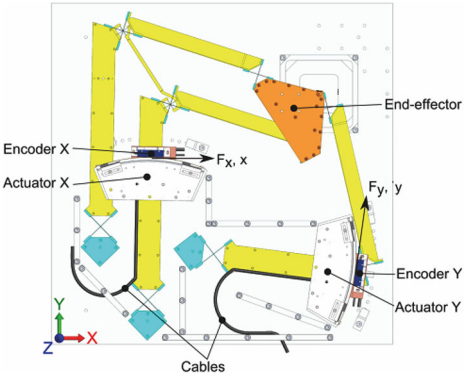

Fig. 6 Top view of the 2 DOF manipulator with flexure joints [30]. The view is rotated relative to the photo from Fig. 5

the latter due to the stiffness in all joints. The non-conservative forces $\boldsymbol{F}_{\mathrm{nc}}$ describe all remaining, possibly non-linear effects like cogging or friction caused by the cables connected to the moving parts of the actuators and sensors.

Both rotation angles of the actuated arms are chosen as independent coordinates $\boldsymbol{q}$. These are computed from the displacements x and y measured with "Encoder X" and "Encoder Y" in Fig. 6. Their derivatives $\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ are further found by means of numerical differentiation and filtering. Both actuator forces F_x and F_y shown in this figure are calculated from the applied motor currents, assuming a constant and known ratio between force and current for each motor. They form the vector $\boldsymbol{F}$, introduced in Sect. 3.1. All the measurements are originally sampled at 10 kHz. For the training of the NNs, 9 datasets with a duration of 120 s each are used. To evaluate the proposed DID method, these data are subsequently downsampled to $f_{\mathrm{sampling}} = 200$ Hz for further processing.

The LNN and FNN, as outlined in Sect. 2.3, have been trained offline with these data, as has been described before [24], and is only summarized briefly in this paper. It has been confirmed that the conservative forces $\boldsymbol{F}_{\mathrm{c}}$ estimated by the LNN account for the major contribution to the total force vector and hence this network can be trained first. The hyper-parameters that define the network topology and the settings for training are determined with a hyper-parameters search. Good performance was obtained with 8 hidden layers of 32 neurons each and using a hyperbolic tangent activation function. Confining the network output to physical meaningful estimates proved to be very beneficial, as only $20,000$ data samples were needed to train the network successfully, where these data were split in 70% training and 30% validation data. Next, the FNN was trained to account for the non-conservative contribution $\boldsymbol{F}_{\mathrm{nc}}$ in the measured forces and hence using the residuals that remain when comparing the output of the trained LNN with the original data. To avoid overfitting of this black-box model, a relatively simple topology of only 2 hidden layers of 8 neurons each was used, trained with $200,000$ data samples. For more details about the training of both NNs, the reader is referred to the original paper [24]. For the experiments described in that paper as well as used in this paper, the feedforward force generated by the trained networks has been computed offline for an a priori known reference trajectory, but this computation is fast enough for an online evaluation in real-time.

The quality of results and the overall performance of the proposed DID method depend on the choice of the learning strategy and the corresponding parameters. One parameter common to both approaches, first introduced in equation (8), is the regularization factor, set $\alpha = 10^{-7}$ for all further examples. We excluded the regularization factor from the hyper-parameters search, as we observed that the prediction quality shows limited sensitivity to changes in the regularization factor, provided it remained within a reasonable range. Specifically, excessively low values ($\alpha < 10^{-10}$) practically eliminated regularization from the optimization function, while high values ($\alpha > 10^{-3}$) altered the primary optimization goal to finding the smallest-norm model matrix.

Moreover, the parameter ρ must be set for the weighted version of the algorithm. In addition to parameter ρ, the windowed version also requires the specification of the window horizon time or its size w. We relate these parameters in Sect. 4.5, where we also look at the hyper-parameters tuning for the discussed DID controller.

The results presented in this section come from implementing the proposed method in Matlab. It is important to note that this may not be an optimal implementation of the discussed algorithms, and the timings provided were executed on a standard desktop computer. Consequently, these timings are intended only for internal comparison among the various settings outlined in the case study. Both the dataset used and the basic script demonstrating the work of the method are publicly available at https://github.com/mpiku/research-didffm

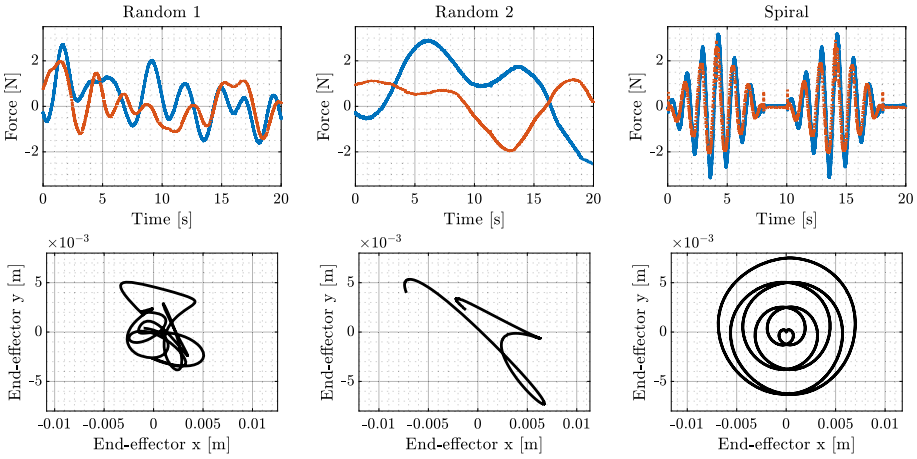

Fig. 7 Three trajectories used for experiments are shown: "Random 1", "Random 2", and "Spiral". In the top row, the total forces delivered to the system are represented by blue and red for F_x and F_y, respectively. The bottom row depicts the trajectories of the end-effector produced by these forces (color figure online)

As shown in Fig. 7, three predefined trajectories were employed in our experiments: "Random 1", "Random 2", and "Spiral". The two random trajectories, "Random 1" and "Random 2", are non-periodic, smooth random functions defined by Fourier series with random coefficients initially generated as the end-effector position history. They differ in spectral content, with "Random 1" containing higher frequencies than "Random 2".

The "Spiral" trajectory consists of two phases: in the first phase, the end-effector follows a spiral path from the center outward, and in the second phase, it follows a spiral path in the opposite direction. This trajectory includes a 2-second pause between the phases, which poses a significant challenge for the learning algorithm. The difficulty comes from the fact that as the manipulator is ordered to remain in a static configuration, its control system stabilizes the position, effectively canceling disturbances. Consequently, this leads to a dataset in which different forces map to the same or similar position and their derivative values. Therefore, learning an inverse relation becomes a cumbersome task.

The issue is also visible when the actuators must overcome static friction to start motion, as described in [16, 31]. Even though the problem of friction is reduced in flexible joints, ill-conditioning data entries might still result from other factors, such as cogging in the actuators.

4.2 Accumulated error analysis

The challenges do not solely stem from interfacing with a real device. Some of the implementation difficulties can also be observed through simulation alone. One such challenge is the accumulation of numerical errors over time. As the model undergoes continuous updates, small errors can accumulate, leading to a decrease in the model's performance.

To mitigate the effects of numerical errors, recomputing the model from scratch periodically is often employed. This restart serves as error correction, ensuring the model remains accurate and reliable. However, it is important to note that this process can be computationally expensive and especially roadblocking in implementations when dealing with long measurement history. Indeed, this computational cost is the primary motivation behind the

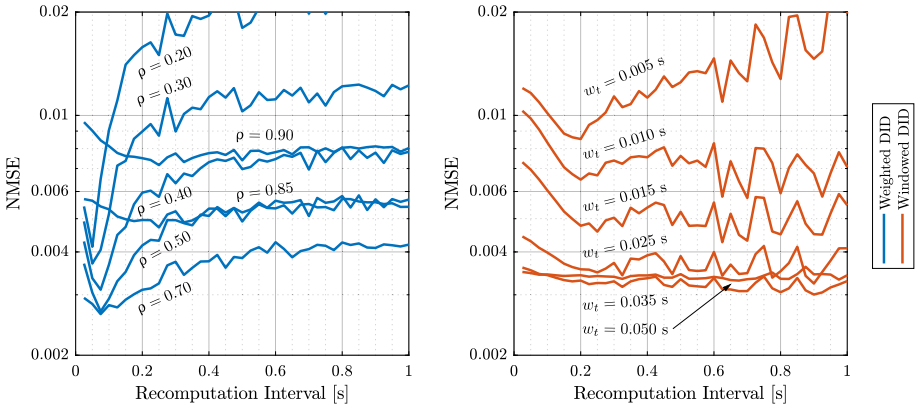

Fig. 8 NMSE as a function of recomputation interval for different weighting factors ρ in weighted recursive DID – left, and different window sizes $w_t = w/f_{\text{sampling}}$ in windowed DID ($\rho = 0.98$) – right. Both approaches are tested with prediction horizon $p_t = 0.025$ s. The results come from the "Random 1" trajectory, on which the NMSE $= 2.74 \times 10^1$ without DID implemented

introduction of updating in the proposed method. Generally, updating the model has a complexity of $\mathcal{O}\left(n^2\right)$, whereas recomputing from scratch involves inverting a matrix, resulting in an $\mathcal{O}\left(w^3\right)$ complexity.

Therefore, it is essential to find the frequency of restarts and point the recomputation interval, which will lead to a compromise between accuracy and efficiency. In Fig. 8, normalized mean squared error (NMSE) is shown as a function of the interval for different weighting factors ρ in weighted recursive DID (blue, on the left) and different window sizes w_t (expressed as periods in seconds) in windowed DID (red, on the right). The interval means that at each step coinciding with the beginning of the interval, the model is built from scratch instead of just updated.

Before further interpretation, let us establish a definition of NMSE as follows

$$\text{NMSE} = \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{l} \frac{\epsilon_{i,j}^2}{\sigma_i^2}, \tag{39}$$

where k is the number of control signals (for the discussed robot, $k = 2$), l is the number of prediction/measurement pairs used in the analysis, $\epsilon_{i,j} = \widetilde{e}_{i,j} - e_{i,j}$ is the error between predicted feedforward error $\widetilde{e}_{i,j}$ and measured one $e_{i,j}$ for the i-th control signal at j-th step, and σ_i is the standard deviation of the i-th control signal's measured feedforward error – the error the DID controller learns online.

Another parameter to define is the prediction horizon, denoted as p_t, which in this case is set to $p_t = 0.025$ s. This means that even though the model is updated at every time step (at the assumed sampling rate, it is every 0.005 s), the error prediction is performed every 5 steps for the next 5 steps, using the most up-to-date model available at the time of prediction. The first prediction within this horizon is based on both actual state measurements and the planned trajectory, while subsequent predictions rely solely on the latter.

In the context of weighted DID, it is observed that the NMSE generally increases as we extend the recomputation interval. Notably, more substantial differences exist when dealing with relatively small weighting factors $\rho \in [0.2\ 0.4]$. Conversely, as we approach the upper

limit of the weighting factor $\rho \in [0.85\ 1]$, the error becomes nearly insensitive to changes in the interval length, resulting in almost flat lines.

Regarding windowed DID, a smaller sensitivity is seen across the entire range of window sizes tested. The effect of increasing the error by extending the interval is noticeable for window $w_t = 0.005$ s. It is essential to emphasize that shorter windows, while not entirely, often correspond to smaller weighting factors, showing the high sensitivity of the error w.r.t. the recomputation interval. Similar to the weighted approach, as the model incorporates more data (longer windows $w_t \in [0.035, +\text{inf})$), it becomes less sensitive to interval changes.

Interestingly, too frequent recomputations lead to an increase in error for both approaches. It might be the error introduced by operations involved in building the model from scratch, e.g., pseudoinverse of a large, sometimes poorly-conditioned, data matrix. Updating the algorithm further attenuates this initial error by applying weights to the old data. Furthermore, the Sherman–Morrison formula might exhibit self-correcting features [32].

While weighted DID requires a shorter recomputation interval to reach the minimum (min (NMSE) at ≈ 0.015 s) compared to the windowed approach (min (NMSE) at ≈ 0.2 s), it generally achieves a smaller NMSE. As the interval increases, exceeding 0.02 s, the plots become less smooth, potentially indicating the influence of other factors, independent of the recomputation interval, on the error.

It is valuable to consider the range of NMSE values and how the recomputation affects the error concerning the reference NMSE when DID is not implemented. For the "Random 1" trajectory used in generating the discussed results, the reference NMSE is 2.74×10^{1}. From this perspective, the influence of the recomputation interval appears relatively minor. However, when dealing with more demanding trajectories, such as the "Spiral" trajectory, there are differences of several orders of magnitude between intervals of 0.2 s and 1.0 s. Therefore, in our subsequent examples, selecting a recomputation interval of 0.2 s seemed reasonable (recomputing the model every 40 steps).

4.3 Computational efficiency of weighted algorithm

The recomputation procedure introduced in Sect. 4.2 significantly affects the computational time of the weighted DID. In this method, the lack of downdating implies that the model is built using all the measurements gathered up to the last update. Consequently, rebuilding the model requires storing all these past measurements, which is memory-inefficient and inverting such a large data matrix – an operation also having considerable computational costs.

In contrast, the computational time of windowed DID does not change notably when recomputation is implemented because the model is always constructed from a predefined number of measurements. Empirically, the difference between these approaches can be observed in the computational times presented in Table 1. The original time for weighted DID with recomputing implemented (shown in parentheses) is consistently 10 times longer than that for the windowed counterpart across all updating methods tested (additional methods are explained in Sect. 4.4).

Figure 9 presents a scheme of weighted DID working on measurement pairs $(\boldsymbol{e}_i, \boldsymbol{z}_i)$ with a recomputation interval that corresponds to rebuilding the model every 3 steps. The dashed rectangles enclose data used for each of the recent 3 steps (it is analogical for all prior steps). While steps k and $k-2$ utilize only a single measurement pair, the recomputation step at $k-1$ involves rebuilding the model using data from $(\boldsymbol{e}_1, \boldsymbol{z}_1)$ up to $(\boldsymbol{e}_{k-1}, \boldsymbol{z}_{k-1})$.

The data is multiplied by a weighting factor ρ, creating an exponential weighting function. Then, the influence of the oldest measurements on the model can be minimal, e.g., if

Table 1 Average computation time for the updating part of the algorithm, measured for different updating approaches (described in Sect. 4.4) and learning strategies. Results come from averaging the time of 1,000 runs of each combination. Timings in parentheses, included in the weighted strategy column, relate to the original weighted algorithm without the adjustment described in Sect. 4.3

Update approach	Learning strategy	
	Weighted	Windowed
Sherman–Morrison	0.046 s (0.545 s)	0.053 s
Cholesky decomp.	0.042 s (0.562 s)	0.048 s
QR decomp.	0.049 s (0.573 s)	–
Woodbury	–	0.054 s

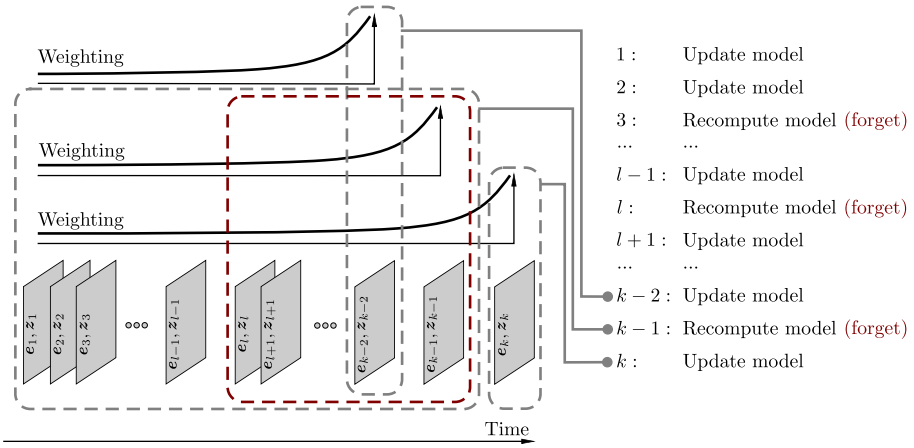

Fig. 9 Scheme illustrating the introduction of forgetting old and low-impact measurements in the weighting approach. The change (presented in dark red) allows for much faster computation when the model is recomputed. The example assumes the recomputation interval is 3 steps

the current step $k = 100$, the $(\boldsymbol{e}_1, \boldsymbol{z}_1)$ pair is weighted with a factor of $0.9^{99} = 2.95 \times 10^{-5}$ if $\rho = 0.9$. Hence, we propose removing data samples weighted by a factor smaller than a predefined threshold $\rho_{th} < \rho$. The scheme illustrates this suggested modification using dark red colors, assuming the cut in the measurement pairs used at step $k-1$ is made at pair l concerning the threshold.

Since this threshold is solely related to the forgetting factor ρ, it is possible to predetermine a specific window size w_{th} used during recomputations, i.e.,

$$w_{th} = \lfloor \frac{\ln \rho_{th}}{\ln \rho} \rfloor. \tag{40}$$

This adjustment makes the weighted algorithm similar to the windowed DID, with the key difference being that in windowed DID, the window moves at each step, including update-downdate steps. In contrast, here it is applied only during the recomputation step. Additionally, the proposed modification smoothly attenuates the influence of the oldest elements toward the threshold, whereas in windowed DID, removing elements from the model that

had a substantial impact in the previous step is possible. These changes result in lower computational times and make it comparable to the windowed version (see Table 1, modified weighted algorithm timings are presented outside the parentheses). In further analysis, we utilize a threshold value $\rho_{th} = 10^{-3}$.

4.4 Performance of windowed algorithm

Using the Sherman–Morrison formula is one of the available options for updating and downdating. In the derivation and implementation of the windowed version of the algorithm, a formula more general to Sherman–Morrison appears — the Woodbury matrix identity — that allows consolidating the updating and downdating of the model into a single operation, eliminating the need for splitting it into two separate steps. Following the approach in [27], utilizing the Woodbury formula

$$(\mathbf{D} + \mathbf{UCV})^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{VD}^{-1}\mathbf{U}\right)^{-1}\mathbf{VD}^{-1}, \tag{41}$$

where for a step k

$$\mathbf{U}_k = \begin{bmatrix} \sigma^w \boldsymbol{z}_{k-w+1} & \boldsymbol{z}_{k+1} \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{V}_k = \begin{bmatrix} \sigma^w \boldsymbol{e}_{k-w+1} & \boldsymbol{e}_{k+1} \end{bmatrix}^T \tag{42}$$

would replace the two-steps update originating from (20) and described in steps 3.(a) to 3.(d) of the windowed version of the algorithm in Sect. 3.4. Finally, the updated model $\mathbf{D}_{k+1}$ could be found using the following equations

$$\mathbf{D}_{k+1} = \mathbf{D}_k + (\mathbf{V}_k - \mathbf{D}_k\mathbf{U}_k)\,\boldsymbol{\Gamma}_{k+1}\mathbf{U}_k^T\frac{\mathbf{P}_k}{\rho}, \tag{43}$$

where

$$\boldsymbol{\Gamma}_{k+1} = \left(\mathbf{C}^{-1} + \mathbf{U}_k^T\frac{\mathbf{P}_k}{\rho}\mathbf{U}_k\right)^{-1}, \quad \mathbf{P}_{k+1} = \frac{\mathbf{P}_k}{\rho} - \frac{\mathbf{P}_k}{\rho}\mathbf{U}_k\boldsymbol{\Gamma}_{k+1}\mathbf{U}_k^T\frac{\mathbf{P}_k}{\rho}. \tag{44}$$

Additionally, we have implemented Cholesky decomposition updates in both weighted and windowed DID methods and QR decomposition updates in the weighted approach. We accomplished this by utilizing the respective functions available in Matlab, whose documentation refers [29] for the algorithmic details.

When decomposition techniques are employed, the algorithms described at the end of Sect. 3.3 and 3.4 change the initialization of $\mathbf{P}_k$ – instead of matrix inversion, this step involves the respective matrix decomposition (step 2 in the algorithms). Then, decomposition updates are utilized to determine the new $\mathbf{P}_{k+1}$ as new measurements are streamed.

It is worth noting that the decompositions are of $\mathcal{O}\left(n^3\right)$ complexity, and their updates have a complexity of $\mathcal{O}\left(n^2\right)$. Together with the additional Woodbury-based approach, this provides 4 alternative methods that share the same complexity regarding initialization, recomputation, and updating/downdating. The timing results presented in Table 1 reflect this fact, where minor differences between the methods may stem from implementation specifics or variations in constants within the complexities.

The computation times in Table 1 were measured during the generation of data for the plots in Fig. 10. These results represent the total time spent on building models over 15-second stream of measurements and are averaged over approximately 1,000 runs of the

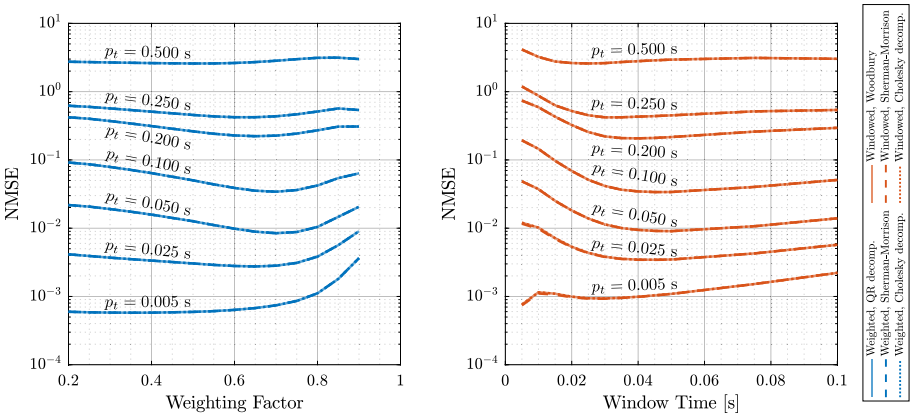

Fig. 10 NMSE as a function of weighting factor and window time $w_t = w/f_{\text{sampling}}$ for weighted recursive DID and windowed DID ($\rho = 0.98$), respectively. Reference NMSE, without DID implemented, equals 2.74×10^1. Results are shown for different prediction horizon lengths p_t and various updating methods - Sherman Morisson formula, Woodbury identity, QR decomposition-based update, and Cholesky decomposition-based update

DID algorithm with different parameters (weighting factor, window time; updating does not depend on prediction horizon) on the "Random 1" trajectory. Thus, the purpose of timing is only to compare these times relatively to each other.

As Fig. 10 demonstrates, both algorithms (the weighted in blue on the left and the windowed in red on the right) produced similar NMSE values within the tested range of parameters. Therefore, based on Table 1, it is reasonable to conclude that the weighted algorithm consistently outperformed the windowed one in terms of computational time. It is worth pointing out that the windowed algorithm needs an additional step or rank-2 modification to downdate the model, which might be the cause for longer execution times. In both approaches, the Cholesky-decomposition-based method yielded the best time performance. However, even in the most demanding scenario, where the prediction horizon is 0.005 s, and the slowest-performing algorithm takes 0.054 s for $3,000$ data snapshots (resulting in 1.8×10^{-5} s per snapshot), there is still time available for context switching and other computational-related overhead to ensure that the DID controller can operate in real-time on the machine used for tests.

Figure 10 illustrates that all introduced updating methods yield the same NMSE, as evidenced by the overlapping lines for both weighted and windowed DID. Despite the previously mentioned similarity in NMSE achieved by both approaches, there is a distinct difference when considering small prediction horizons. Specifically, weighted DID can achieve smaller NMSE, and it appears easier to find better-performing settings in this approach. The NMSE lines for $p_t \leq 0.025$ s are nearly flat up to $\rho < 0.75$ (also containing the minimum), a significant part of the available values.

Interestingly, in the case of windowed DID with $p_t = 0.005$ s, the best results are obtained with a window size of $w_t = 0.005$ s. This reads that it might be reasonable to utilize only the most recent measurement to predict one step without considering the overall dynamics of the error. As the prediction horizon increases, the plots become flatter for both approaches, indicating that the error in such long-term predictions is almost independent of the controller's parameters. However, even for a prediction horizon of $p_t = 0.5$ s, the

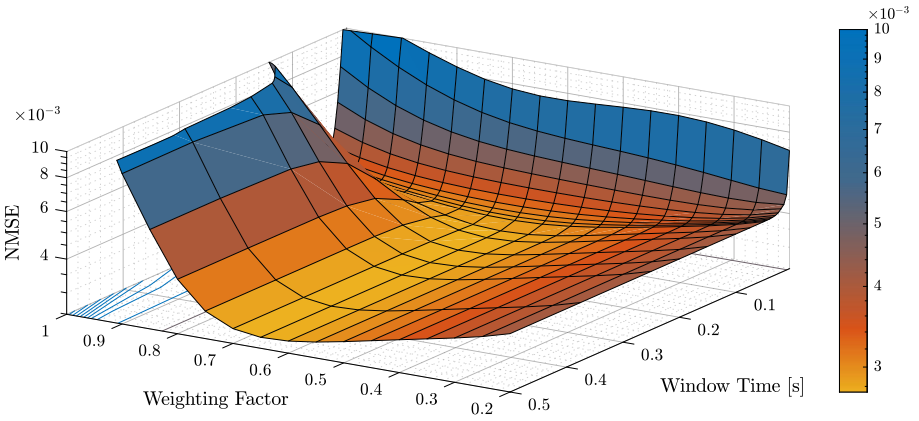

Fig. 11 Hyper-parameters search for the windowed algorithm (Sherman-Morrison) — NMSE as a function of weighting factor and window time $w_t = w/f_{\text{sampling}}$. The surface is rendered for prediction horizon $p_t = 0.025$ s on "Random 1" trajectory, on which the reference (without DID) NMSE $= 2.74 \times 10^1$. The minimum NMSE is found in the neighborhood of a weighting factor of $\rho = 0.6$ ($\rho = 0.65$, $w_t = 0.15$ s), resulting in NMSE equals 2.74×10^{-3}

NMSE, which oscillates around 3, remains smaller than the reference NMSE $= 2.74 \times 10^1$ (without error learning).

Apart from the boundary values of the prediction horizon analyzed, the parameters that achieve the minimum NMSE are in the neighborhood of $\rho = 0.7$ for weighted DID, and $w_t = 0.03$ s for windowed DID. Nevertheless, it is essential to note that these optimal settings may vary significantly for different devices or distinct trajectories.

4.5 Choosing learning strategy for the 2 DOF manipulator

As observed in Sect. 4.3, after implementing periodic recomputations, the weighted approach shares many similarities with windowed DID. In particular, one can set the weighting factor and window size in windowed DID such that the oldest measurements in the window are effectively multiplied by the weighting threshold defined for weighted DID. Given that both approaches can achieve similar NMSE results, as shown in Sect. 4.4, it becomes interesting to look at hyperparameter tuning, where we seek to minimize the error as a function of both the weighting factor and window size.

The resulting surface is presented in Fig. 11, indicating the optimal settings for $\rho^* = 0.65$ and $w_t^* = 0.15$ s. Given the intersection along the weighting factor axis, we recognize a similar pattern to the plot in Fig. 10, provided that the window is long enough. A similar pattern emerges also at the intersection for a specific weighting factor. However, it is worth noting that the range of window times presented in Fig. 11 is significantly broader relative to Fig. 10.

The most noteworthy observation is that for a long enough window, its exact size does not impact the NMSE, as long as the weighting factor is set around 0.6. When considering windowed DID with settings that minimize NMSE, it becomes apparent that the oldest measurement in the window is multiplied by a weighting-related factor of $(\rho^*)^{w_t^* f_{\text{sampling}}} = 6.59 \times 10^{-4}$, less than the threshold assumed in Sect. 4.2. Nonetheless, if the window is set to correspond to the threshold window defined in (40) for a weighting factor $\rho = 0.65$, the NMSE changes by only 0.5% relative to the minimal NMSE.

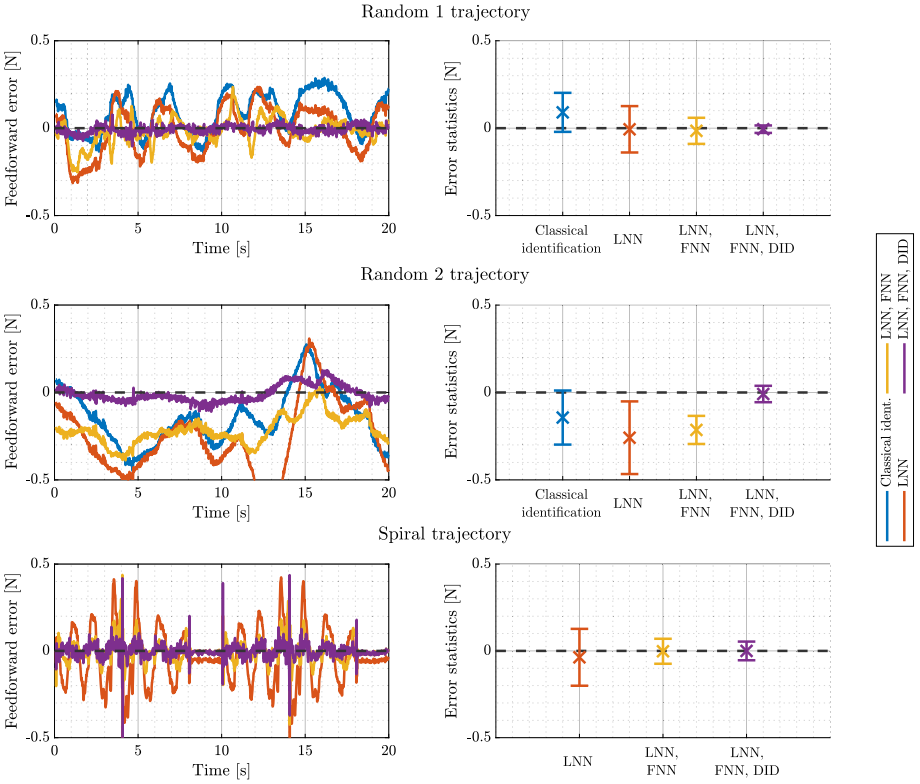

Fig. 12 Time history of error and its statistics (mean and standard deviation) for various combinations of feedforward (FF) components, i.e., FF based solely on Classical identification, FF based solely on LNN, FF based on LNN and FNN working together, and FF based on LNN and FNN with DID implemented (results for "Actuator X"). Error is tested on three trajectories: "Random 1", "Random 2", and "Spiral" (color figure online)

Therefore, using weighted NMSE is more convenient because, as long as a reasonable weighting threshold is set, we only need to tune the weighting factor. In relation to the results presented, the NMSE as a function of the weighting factor remains relatively flat near the minimum, which is advantageous when a detailed analysis is not possible. Moreover, since $\mathbf{P}_k$ is a symmetric positive definite matrix, the denominator in (22) is always larger than 1. In contrast, for the windowed approach, the denominator in (37) might converge to 0, amplifying errors and introducing further numerical challenges.

4.6 Validation on different reference trajectories

All the previous considerations used "Random 1" trajectory to generate data. It is common in system identification to find and use a specific system-exciting signal to determine or fine-tune model parameters. However, introducing this DID error-learning controller aims to enhance feedforward prediction capabilities for arbitrary trajectories. Therefore, Fig. 12 presents error analysis for feedforward controllers including different components (ultimately including DID), tested on three trajectories introduced in Sect. 4.1. Results and

Table 2 Error statistics (μ - mean, σ - standard deviation) for various combinations of components included in the feedforward controller (results for "Actuator X"). Error is tested on three trajectories: "Random 1", "Random 2", and "Spiral"

Feedforward	Trajectory		
	Random 1	Random 2	Spiral
Classical identification	$\mu = 0.090$	$\mu = -0.143$	–
	$\sigma = 0.112$	$\sigma = 0.155$	–
LNN	$\mu = -0.006$	$\mu = -0.259$	$\mu = -0.037$
	$\sigma = 0.133$	$\sigma = 0.208$	$\sigma = 0.164$
LNN, FNN	$\mu = -0.015$	$\mu = -0.214$	$\mu = -0.002$
	$\sigma = 0.075$	$\sigma = 0.080$	$\sigma = 0.072$
LNN, FNN, DID	$\mu = -0.006$	$\mu = -0.009$	$\mu = 0.000$
	$\sigma = 0.021$	$\sigma = 0.047$	$\sigma = 0.054$

statistics are presented for "Actuator X" (see Fig. 6). "Actuator Y" has the same trends with minor divergence in the "Spiral" trajectory, which is discussed further.

To see whether the relaying on the NMSE index led us in a good direction, Fig. 12 provides the time history of feedforward error on the left side and the respective error statistics (mean and standard deviation) on the right. We employ weighted DID with $\rho = 0.65$, as found in Sect. 4.5, and weighting threshold $\rho_{th} = 10^{-3}$, as assumed in Sect. 4.2. Ideally, the error should remain as close as possible to the dashed horizontal line at 0.

The results for the "Random 1" trajectory, to which the parameters were tuned, are expectedly quite promising. Compared to a feedforward controller based on classical identification, the mean of the error is reduced by a factor of 10, and its standard deviation is 5 times smaller (exact values included in Table 2). Additionally, significant improvements are found when comparing the final structure of the control system (LNN, FNN, DID) with pure LNN-based control or LNN with FNN.

The "Random 2" trajectory, which originates from the same generator as "Random 1" but contains different frequencies, shows similar improvements. In this case, the neural network-based controllers produce a larger error with a higher standard deviation. However, the DID algorithm learns the error on the fly and brings the error back close to the desired 0 line.

As mentioned in Sect. 4.1, the most challenging trajectory is the "Spiral" one. In fact, the improvement is smaller than in other cases, but the error is still reduced compared to the LNN with the FNN controller. It is worth noting that the spikes in error at times 8 s, 10 s, and 18 s coincide with transitions from motion into a pause mode, and also, they appear at 4 s and 14 s when the spiral direction changes in a non-smooth manner. "Actuator Y", in this case, is even more sensitive to the discussed phenomena, and although minimizing mean error by 70%, the DID increased the error's standard deviation by 50%. It also indicates that the use of the proposed approach for systems with impacts or other discontinuities would probably require additional research.

Overall, a clear trend is visible in which adding components to the feedforward controller reduces the error. Online learning of the error produced by the feedforward controller built with LNN and FNN certainly contributes both qualitatively and quantitatively.

5 Conclusions

This paper has explored the challenges and solutions in data-driven control, specifically focusing on the proposed DID method for error modeling of feedforward controller based on PINNs. Besides providing details on the structure of such a DID controller, we derived algorithms for efficient online error learning in weighted and windowed settings. Moreover, based on measurement data from experiments with the 2 DOF flexure manipulator, our investigation revealed several insights and findings.

- We have highlighted the challenges inherent in data-driven control, emphasizing the impact of numerical errors that can accumulate over time as models are continually updated. These errors can significantly affect the performance and reliability of the proposed controller.
- To mitigate these issues, we introduced a periodic recomputation strategy. We discussed the trade-off between accuracy and efficiency in selecting recomputation intervals. We demonstrated that too frequent recomputations can lead to increased error, and we explored the potential reasons behind this phenomenon.
- We modified the weighted DID algorithm, making its computation time in our tests comparable to or even better than the windowed approach. Experiments with various updating algorithms did not reveal significant differences in the resulting models.

While the extensively investigated case study focuses on a planar manipulator with only two actuators, this does not constrain the applicability of the method or the conclusions drawn from it. Indeed, the application of DID error learning can be readily scaled to systems with larger DOF and spatial systems, which mainly involves increasing the size of the matrices introduced in Sect. 3. However, the limiting factor may be the quadratic complexity of the updating method, which, on the other hand, remains comparable to successful online learning approaches discussed in Sect. 1.

- Nevertheless, upon examining the complexity of prediction, it is evident that the proposed method's dependency is solely on the dimension of the input. In contrast, alternative methods may also consider the number of points utilized to construct the model or local models, which must be tested across various distance measures concerning the regressed input.

The primary conclusion drawn from our research is derived from the discussion on finding the right balance between the weighting factor and window size when implementing windowed DID. It indicates that when properly configured, the weighted approach can yield favorable results while simplifying the tuning process and being less exposed to numerical issues than the windowed algorithm. Error statistics from experiments in various scenarios consistently show improvement when adding the proposed controller, even for challenging trajectories, highlighting the practical benefits of the DID error learning.

A few further research directions arise from the presented material. Concerning applications of the method in restrictive time regimes, investigating updating based on Cholesky decomposition seems promising, as it outperformed other methods in the included tests. Another direction is to make the technique more versatile by researching the automatic adaptation of the weight value or window size parameters. Moreover, a few mentioned concepts need to be investigated more in-depth, e.g., the numerical stability of the updating part.

Acknowledgements Research was funded by the Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) programme.

Author contributions All authors contributed to the study conception and design. Experimental data was mainly delivered by R.A. Formal analysis was performed by M.P. and P.M. Method implementation, numerical results collection, interpretation, and visualization were made by M.P. The first draft of the manuscript was collaboratively written by M.P., P.M., and R.A. All authors read and approved the final manuscript.

Declarations

Competing interests The authors declare no competing interests.

References

1. Soudbakhsh, D., Annaswamy, A.M., Wang, Y., Brunton, S.L., Gaudio, J., Hussain, H., Vrabie, D., Drgona, J., Filev, D.: Data-driven control: theory and applications. In: 2023 American Control Conference (ACC), pp. 1922–1939 (2023). https://doi.org/10.23919/ACC55779.2023.10156081
2. Ebers, M.R., Steele, K.M., Kutz, J.N.: Discrepancy Modeling Framework: Learning missing physics, modeling systematic residuals, and disambiguating between deterministic and random effects (2022). https://doi.org/10.48550/arXiv.2203.05164
3. Meier, F., Kappler, D., Ratliff, N., Schaal, S.: Towards robust online inverse dynamics learning. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4034–4039. IEEE, Daejeon, South, Korea (2016). https://doi.org/10.1109/IROS.2016.7759594
4. Lutter, M., Peters, J.: Combining physics and deep learning to learn continuous-time dynamics models. Int. J. Robot. Res. **42**(3), 83–107 (2023). https://doi.org/10.1177/02783649231169492
5. Stulp, F., Sigaud, O.: Many regression algorithms, one unified model: a review. Neural Netw. **69**, 60–79 (2015). https://doi.org/10.1016/j.neunet.2015.05.005
6. Nguyen-Tuong, D., Peters, J., Seeger, M., Schölkopf, B.: Learning inverse dynamics: a comparison. In: Advances in Computational Intelligence and Learning: Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2008), pp. 13–18 (2008)
7. Schaal, S., Atkeson, C.G., Vijayakumar, S.: Scalable techniques from nonparametric statistics for real time robot learning. Appl. Intell. **17**(1), 49–60 (2002). https://doi.org/10.1023/A:1015727715131
8. Vijayakumar, S., Schaal, S.: Locally weighted projection regression: an O(n) algorithm for incremental real time learning in high dimensional space. In: Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), vol. 1, pp. 288–293. Morgan Kaufmann, San Mateo (2000)
9. Hitzler, K., Meier, F., Schaal, S., Asfour, T.: Learning and adaptation of inverse dynamics models: a comparison. In: 2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids), pp. 491–498. IEEE, Toronto (2019). https://doi.org/10.1109/Humanoids43949.2019.9035048
10. De La Cruz, J.S., Kulić, D., Owen, W.: Online incremental learning of inverse dynamics incorporating prior knowledge. In: Kamel, M., Karray, F., Gueaieb, W., Khamis, A. (eds.) Autonomous and Intelligent Systems, vol. 6752, pp. 167–176. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-21538-4_17
11. Rasmussen, C.E.: Gaussian Processes in Machine Learning, pp. 63–71. Springer, Berlin (2004). https://doi.org/10.1007/978-3-540-28650-9_4
12. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. The MIT Press, Cambridge (2018). https://doi.org/10.7551/mitpress/4175.001.0001
13. Nguyen-Tuong, D., Seeger, M., Peters, J., Koller, D., Schuurmans, D., Bengio, Y., Bottou, L.: Local Gaussian process regression for real time online model learning and control. In: Proceedings of the 2008 Conference. Advances in Neural Information Processing Systems, vol. 21, pp. 1193–1200 (2009)
14. Nguyen-Tuong, D., Peters, J.: Using model knowledge for learning inverse dynamics. In: 2010 IEEE International Conference on Robotics and Automation, pp. 2677–2682. IEEE, Anchorage (2010). https://doi.org/10.1109/ROBOT.2010.5509858

15. Shareef, Z., Mohammadi, P., Steil, J.: Improving the inverse dynamics model of the KUKA LWR IV+ using independent joint learning. In: 7th IFAC Symposium on Mechatronic Systems MECHATRONICS 2016, IFAC-PapersOnLine, vol. 49, pp. 507–512 (2016). https://doi.org/10.1016/j.ifacol.2016.10.653
16. Kappler, D., Meier, F., Ratliff, N.D., Schaal, S.: A new data source for inverse dynamics learning. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4723–4730 (2017). https://doi.org/10.48550/arXiv.1710.02513
17. Morse, K., Das, N., Lin, Y., Wang, A.S., Rai, A., Meier, F.: Learning state-dependent losses for inverse dynamics learning. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5261–5268 (2020). https://doi.org/10.1109/IROS45743.2020.9341701
18. Reinhart, R.F., Steil, J.J.: Hybrid mechanical and data-driven modeling improves inverse kinematic control of a soft robot. In: 3rd International Conference on System-Integrated Intelligence: New Challenges for Product and Production Engineering, Procedia Technology, vol. 26, pp. 12–19 (2016). https://doi.org/10.1016/j.protcy.2016.08.003
19. Khalil, W., Dombre, E.: Modeling, Identification and Control of Robots. Butterworth-Heinemann, Oxford (2002). https://doi.org/10.1016/B978-1-903996-66-9.X5000-3
20. Wu, J., Wang, J., You, Z.: An overview of dynamic parameter identification of robots. Robot. Comput.-Integr. Manuf. **26**(5), 414–419 (2010). https://doi.org/10.1016/j.rcim.2010.03.013
21. Abdul-hadi, O.: Machine learning applications to robot control. PhD thesis, University of California, Berkeley (2018) https://escholarship.org/uc/item/78h7x9r7
22. Lutter, M., Ritter, C., Peters, J.: Deep Lagrangian networks: Using physics as model prior for deep learning (2019). https://doi.org/10.48550/arXiv.1907.04490
23. Agrawal, A., Modi, A.N., Passos, A., Lavoie, A., Agarwal, A., Shankar, A., Ganichev, I., Levenberg, J., Hong, M., Monga, R., Cai, S.: Tensorflow eager: a multi-stage, python-embedded dsl for machine learning. In: Talwalkar, A., Smith, V., Zaharia, M. (eds.) Proceedings of the 2nd SysML Conference, vol. 1, pp. 178–189 (2019). https://mlsys.org/Conferences/2019/doc/2019/88.pdf
24. Heerze, E., Aarts, R.G.K.M., Rosic, B.: Feedforward control for a manipulator with flexure joints using a Lagrangian neural network. In: IUTAM Symposium on Optimal Design and Control of Multibody Systems 2022: Adjoint Methods, Alternatives, and Beyond, p. 1 (2022). https://www.tuhh.de/mum/iutam-symposium-2022.html
25. Liu, Z., Wang, B., Meng, Q., Chen, W., Tegmark, M., Liu, T.-Y.: Machine-learning nonconservative dynamics for new-physics detection. Phys. Rev. E **104**, 055302 (2021). https://doi.org/10.1103/PhysRevE.104.055302
26. Nguyen-Tuong, D., Peters, J.: Model learning for robot control: a survey. Cogn. Process. **12**(4), 319–340 (2011). https://doi.org/10.1007/s10339-011-0404-1
27. Zhang, H., Rowley, C.W., Deem, E.A., Cattafesta, L.N.: Online dynamic mode decomposition for time-varying systems. SIAM J. Appl. Dyn. Syst. **18**(3), 1586–1609 (2019). https://doi.org/10.1137/18M1192329
28. Kailath, T., Sayed, A.H., Hassibi, B.: Linear Estimation. Prentice Hall, New Jersey (2000)
29. Golub, G.H., Loan, C.F.: Matrix Computations, 4th edn. JHU Press, Baltimore (2013)
30. Brouwer, D., Folkersma, G., Boer, S., Aarts, R.: Exact constraint design of a two-degree of freedom flexure-based mechanism. Journal of mechanisms and robotics **5**(4) (2013). https://doi.org/10.1115/1.4025175
31. Ratliff, N., Meier, F., Kappler, D., Schaal, S.: Doomed: Direct online optimization of modeling errors in dynamics. Big Data **4** (2016). https://doi.org/10.1089/big.2016.0041
32. Allgower, E.L., Georg, K.: Update methods and their numerical stability. In: Numerical Continuation Methods: An Introduction, pp. 252–265. Springer, Berlin (1990). https://doi.org/10.1007/978-3-642-61257-2_16