

Flexible-Joint Manipulator Trajectory Tracking with Learned Two-Stage Model employing One-Step Future Prediction

Dmytro Pavlichenko

Autonomous Intelligent Systems
University of Bonn, Germany

Email: pavlichenko@ais.uni-bonn.de

Sven Behnke

Autonomous Intelligent Systems
University of Bonn, Germany

Email: behnke@cs.uni-bonn.de

Abstract—Flexible-joint manipulators are frequently used for increased safety during human-robot collaboration and shared workspace tasks. However, joint flexibility significantly reduces the accuracy of motion, especially at high velocities and with inexpensive actuators. In this paper, we present a learning-based approach to identify the unknown dynamics of a flexible-joint manipulator and improve the trajectory tracking at high velocities. We propose a two-stage model which is composed of a one-step forward dynamics future predictor and an inverse dynamics estimator. The second part is based on linear time-invariant dynamical operators to approximate the feed-forward joint position and velocity commands. We train the model end-to-end on real-world data and evaluate it on the Baxter robot. Our experiments indicate that augmenting the input with one-step future state prediction improves the performance, compared to the same model without prediction. We compare joint position, joint velocity and end-effector position tracking accuracy against the classical baseline controller and several simpler models.

I. INTRODUCTION

Robot manipulators have been used for decades, and trajectory tracking control has been extensively researched to achieve fast and accurate motion. In the case of classical industrial manipulators, approaches like iterative learning control (ILC) [1] can efficiently achieve this goal. ILC assumes that the exact same trajectories are repeated in a well-structured environment. In recent years, the role of robotic manipulators is extending beyond such scenarios: direct human-robot collaboration in shared workspaces induces significantly increased safety requirements. Frequently, their satisfaction starts at the hardware level by the use of compliant series-elastic actuators. However, flexible manipulators produce less accurate motions and the complex underlying dynamics models are often unknown. The simplest way to address this issue is operation at low velocities. Unfortunately, this limits the efficiency of the system. The objective of this paper is to achieve accurate trajectory execution at high velocities for inexpensive flexible-joint manipulators.

Neural networks (NNs) are known for their ability to generalize and model complex non-linear relations. We present

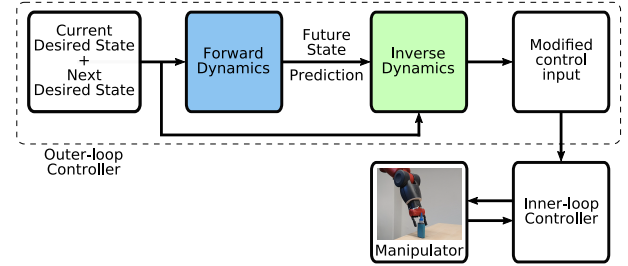


Fig. 1: Two-stage model, employing input augmentation with one-step future prediction for inverse dynamics approximation.

a methodology for neural-learned feed-forward outer-loop control based on linear time-invariant (LTI) dynamical operators. In particular, as a part of our model we use the novel dynoNet [2], which resembles the features of RNN [3] and 1-D convolution [4]. The LTI layers are specifically designed for sequence modeling and system identification, successfully approximating complex non-linear causal dynamics, while being differentiable and suitable for backpropagation. Thus, we utilize them for approximating the inverse dynamics of flexible-joint manipulators. We propose a two-stage model (Fig. 1). The motivation for such model architecture is to hardwire the “*Infer what will happen in the future, then think what would be the best action now to prevent the foreseen inaccuracies*” structure within the network. We elaborate that such an architecture moves away from the pure reactive policy towards the more intelligent planning-ahead behavior. This can be viewed as a simplistic model-predictive framework. The first part approximates the forward dynamics of the manipulator. The output of this model is used to augment the input to the LTI-based model, which approximates the inverse dynamics, producing the feed-forward joint position and velocity commands, which are then being fed to the inner-loop feedback controller of the robot manipulator. Explicitly utilizing both forward and inverse dynamics helps to maximize the extraction of information from the scarce real-world data. The models are trained end-to-end on the real-robot data in a supervised manner using plain backpropagation.

We represent each point of the manipulator trajectory as a tuple $\langle \theta, \dot{\theta}, \ddot{\theta} \rangle$, where θ is a joint positions vector. Explicit inclusion of velocity and acceleration provides information about the dynamics to the NN directly, as opposed to forcing the NN to infer it from the series of joint positions. This state formulation should allow learning the dynamics of the manipulator from a smaller dataset. It is especially important in case of flexible-joint manipulators, which often do not have an accurate dynamics model which could be used to pre-train the model in simulation. Thus, limited real-world data should be used for training. We evaluate the performance of our method on the real Baxter robot against the baseline controller, a multi-layer fully-connected NN, RNN, and plain dynoNet without future prediction step. The training is done on a small real-world dataset. Our approach significantly improves trajectory tracking accuracy, compared to the baseline controller, and outperforms other models. The method allows executing fast trajectories with higher accuracy.

The main contributions of this work are:

- Two-stage model architecture with one-step future prediction for feed-forward trajectory tracking, trained end-to-end with backpropagation,
- investigation of effectiveness of LTI-based models for robotic manipulator inverse dynamics approximation.

II. RELATED WORK

Trajectory tracking has been thoroughly studied for decades. One of the classical methods is iterative learning control [1] (ILC): it iteratively updates the control input with tracking error and after a small number of iterations is able to perfectly track a desired trajectory. ILCs main limitation is non-transferability: the optimization has to be performed from scratch for each new trajectory. Differential dynamic programming [5] (DDP) is another approach, which utilizes a linear quadratic regulator (LQR) in order to iteratively update the control inputs. This approach requires much computational power and is impractical to apply online.

In recent years, trajectory tracking was often addressed by means of NNs [6] [7]. Such increased attention is explained by their comprehensive ability to generalize and model complex nonlinear dynamics. Radial basis function (RBF) NNs are a popular model choice [8] [9] [10]. Xia et al. [11] used RBF-NN to mitigate the effects of friction in swing-up control of a two-joint manipulator. However, such models contain a large number of parameters, and it is a challenging task to tune the hyper-parameters, such as number of Gaussian kernels, their centers and shapes. A wavelet fuzzy neural network is proposed for predictive control by Lu [12]. The model is based on a set of fuzzy rules, where each rule is linked to the wavelet function from the consequent rules. The model is trained with backpropagation. The drawback of this architecture is its complexity and computational expensiveness. Gaussian process regression (GPR) is used by Rueckert et al. [13] to perform kinematic control of a surgical cable-driven manipulator. Saveriano et al. [14] modeled the residual dynamics using GP-based model together with reinforcement learning.

The main drawback of GP-based models, in comparison to neural networks, is that they grow together with the data, thus requiring a lot of resources for evaluation and execution. Mahler et al. [15] demonstrated that LSTM networks can outperform the GP-based method for modeling inverse dynamics. While many works focus on the model architecture, Morse et al. [16] apply meta-learning to obtain state-dependent loss functions, which demonstrates another viable approach.

Much research is built around ILC [17] [18]. For instance, in work by Schwarz et al. [19] the compliant position control is achieved through learning the parameters of DC motors and friction models with a help of ILC. The method was tested in real world with humanoid robot gait. This approach avoids performing a separate run for each parameter, but instead identifies all parameters at once. A combination of \mathcal{L}_1 adaptive control and ILC is used for transfer learning between systems with different dynamics by Pereida et al. [20]. An extended \mathcal{L}_1 controller runs in the inner closed-loop control level and achieves robust and repeatable behavior. ILC is used as an outer-loop control, where the transfer of learned experience is realized. The system is tested on two quadrotors with different dynamics and outperforms systems composed of ILC and proportional-derivative (PD) or proportional-integral-derivative (PID) controllers. Another use of ILC involves production of the ground truth input trajectories to train NNs which approximate the inverse dynamics of the system by Chen et al. [21]. The authors apply this approach to an industrial manipulator, training the model in simulation. The model is then applied to the real robot with transfer learning. A separate NN is trained for each joint using backpropagation. The approach demonstrates a significant improvement of the trajectory tracking accuracy both in simulation and in the real world. However, this method cannot be applied to systems which do not have good dynamics approximation for simulation, which is frequently the case for compliant systems. In addition, the assumption of decoupled joints does not hold when dealing with flexible joints.

Chen et al. [22] present two approaches for performing feed-forward trajectory tracking with series-elastic actuators. The first approach uses RNN which approximates forward dynamics in combination with ILC, which then utilizes this model to find an optimal command sequence. The main drawback of this approach is a significant runtime required for ILC to converge, which makes this method impractical to be applied online. The second approach utilizes bi-directional RNN (BRNN) [23] to approximate the inverse dynamics directly, such as in [24] [25]. This allows to directly obtain required control inputs in a short time. Both approaches are trained with backpropagation on three hours of sinusoidal and random trajectories recorded on the real Baxter robot. Resulting models improved the trajectory tracking over the baseline PD controller. In contrast, we propose a model which utilizes causal LTI dynamical operators. In addition, we also encapsulate a pre-trained model for forward dynamics approximation to augment the input to the inverse dynamics model. This two-stage approach, inspired by Zeng et al. [26],

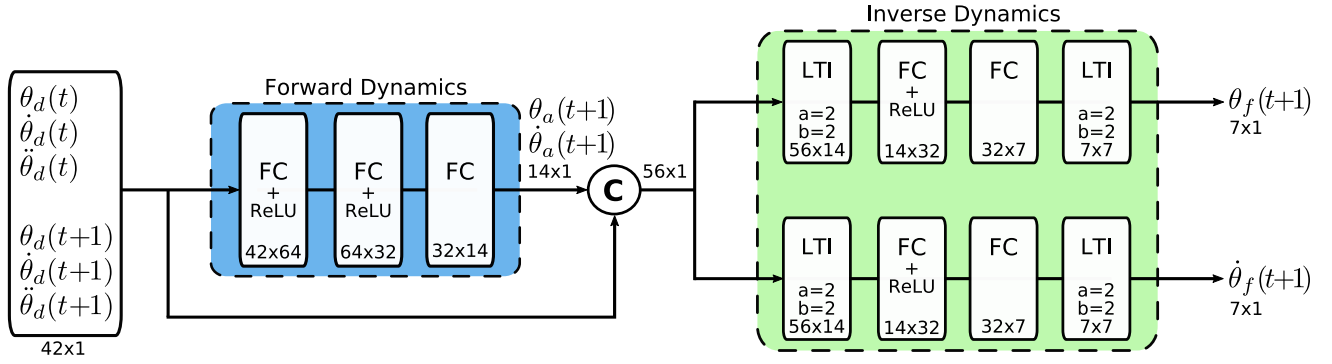


Fig. 2: Proposed model. Input is the current desired state of the manipulator (joint position, velocity, acceleration) plus the next desired state. Blue: Forward Inference Network (FIN), which approximates the forward dynamics. Green: dynoNet Wiener-Hammerstein model (DWH), which takes the same input, augmented with the future state estimation from FIN. It outputs the feed-forward joint position and velocity commands. (c): Concatenation. FC: fully-connected layer. LTI: linear time-invariant dynamical operator-based block.

allows the model to take into account the future prediction in order to estimate the feed-forward command which mitigates future inaccuracies. Combined with more abstract input representation (two timesteps of joint positions, velocities and accelerations instead of multiple timesteps of joint positions), the proposed model is able to learn and generalize from a smaller amount of real-world data. In addition, our model produces not only feed-forward joint positions commands, but velocities as well, which allows to further improve the trajectory tracking accuracy. Finally, we train the model on a smaller dataset of functional trajectories, emulating learning from a regular operation, as opposed to learning from trajectories of specific artificial shapes, such as sine waves.

III. METHOD

Given a desired trajectory θ_d which consists of T equally spaced in time points $\theta_d(t) \in \mathbb{R}^N, t \in [1 \dots T]$ in a joint space with N joints, our objective is to produce feed-forward commands θ_f which would lead the manipulator to follow θ_d . In this section, we present the details on the two-stage model for the manipulator inverse dynamics approximation.

A. Data Collection

In order to learn the inverse dynamics approximation, we generate 45 minutes of *functional* trajectories θ_d for the left arm of the Baxter robot. Random sinusoidal trajectories are often used to form the base of the training set [21] [22]. However, recording of such dataset takes the valuable robot hours away from the user. Thus, we wanted to emulate learning from the data which was collected when performing actual tasks. That is why we refer to such trajectories as *functional*. This is advantageous, because data collection can take place transparently while the robot is performing useful tasks. Subsequently, the learned model extends the range of executable tasks and reduces execution time, improving the overall capabilities of the flexible-joint manipulator.

The training set consists of pick-and-place trajectories with equal portions of them executed with 0.6, 0.8 and 1.0 rad/s

maximum velocities. We did our best to cover the major part of the workspace in front and to the side of the robot. When executing a trajectory θ_d , the actual response of the robot θ_a and $\dot{\theta}_a$ is recorded. Since Baxter has no way to measure the actual acceleration $\ddot{\theta}_a$, we approximate it by a cubic spline interpolation of the θ_a . Approximately 60% of the trajectories contain 1-2 additional waypoints between the start and the goal. This increases the variety of the movements and simulates maneuvers such as avoiding an obstacle.

B. Two-stage Model

We propose a two-stage model which consists of two parts (Fig. 2). The first part is the Forward-Inference Network (FIN): a fully connected multi-layer NN which approximates manipulator forward dynamics. The second part is based on LTI dynamical operators, as implemented in the novel dynoNet¹. LTI operators were shown to be efficient [2] when learning the complex causal non-linear dynamics, while being suitable for an end-to-end backpropagation. These properties are advantageous for learning the dynamics of the flexible-joint manipulator. That is why we chose LTI-based blocks to be the core of our model. The model resembles a MIMO Wiener-Hammerstein structure, according to the block-oriented modeling framework [27]. Thus, it is referred to as dynoNet Wiener-Hammerstein model (DWH). DWH uses the original input augmented with the FIN prediction to approximate the inverse dynamics. We refer to the whole model as FIN-DWH. The two-stage architecture with one step future prediction aims to push the model from purely reactive policy behavior towards more intelligent planning ahead, which would allow achieving a higher accuracy of the trajectory tracking.

The Baxter arm has $N = 7$ joints, thus $\theta(t) \in \mathbb{R}^7$. Since the joints of Baxter are coupled [22], it is not feasible to train a separate model for each joint. Instead, we approximate the

¹<https://github.com/forgi86/dynonet>

underlying dynamics by considering all joints simultaneously. So, the input to the FIN-DWH model is a 42-element vector:

$$[\theta_d(t), \dot{\theta}_d(t), \ddot{\theta}_d(t), \theta_d(t+1), \dot{\theta}_d(t+1), \ddot{\theta}_d(t+1)], \quad (1)$$

where $\theta_d(t)$ is the current desired state of the manipulator and $\theta_d(t+1)$ is the next desired state. The output of the network is then a 14-element vector $[\theta_f(t+1), \dot{\theta}_f(t+1)]$ where $\theta_f(t+1)$ is the feed-forward command which should lead the manipulator to the state $\theta_d(t+1)$. The proposed method is open-loop and does not include live feedback from the robot. There is an assumption that the consequent execution of the corrective feed-forward commands should result in the manipulator following the desired trajectory perfectly. This allows to train offline with the collected data as it is, without the need to introduce an additional dynamics model to produce the ground-truth control inputs, as described in the next subsection. We analyze the practical shortcomings of the aforementioned assumption by performing the experiments with a previously unseen payload. Since the complete feed-forward command sequence from our model is available before it is executed, we apply a zero-phase Savitzky-Golay filter to each individual joint position and joint velocity trajectory with window 21 and polynomial order 2 to alleviate any potential non-smooth fragments in the control signal.

The best-performing Baxter baseline controller is "Inverse Dynamics Feed Forward Position control"². This controller calculates the necessary torque from the supplied positions, velocities and accelerations using the internal dynamics model. Thus, we train our model to produce velocities $\dot{\theta}_f(t+1)$ as well. We do not train the model to output joint accelerations $\ddot{\theta}_f(t+1)$ because they could not be measured by the robot hardware and training with approximated spline accelerations as a target would most likely lead to an inferior performance. To give an intuition about how the position baseline controller and the inverse dynamics feed-forward position baseline controller compare, the average cumulative joint position tracking error per point is 0.157 ± 0.082 rad in the first case against 0.069 ± 0.032 rad ($\mu \pm \sigma$) in the second case. The latter is more than two times accurate. Thus, we train the model to provide feed-forward velocity input and compare against this more accurate baseline. Note, that it is straightforward to use only the position or velocity vector from the output in case when position or velocity control is used. The existing error in trajectory tracking accuracy shows that the internal dynamics model does not represent the complex coupled-joints real-robot dynamics accurately enough. The proposed data-driven method does not replace the classical baseline controller, but forms an outer-loop, complementing the existing dynamics model and learning to compensate for the observed inaccuracies.

By explicitly including velocity and acceleration into the input, we provide enriched information about the manipulator state without forcing the network to infer derivatives from the time series of joint positions. In addition, since a tuple of

$\langle \theta, \dot{\theta}, \ddot{\theta} \rangle$ contains certain information about the dynamic state of the manipulator, we can significantly reduce the number of time steps needed as an input. In this work, we use only two time steps, as described above. Moreover, velocity and acceleration represent certain patterns in robot dynamics in a more general way than sequences of joint positions alone, which values depend on the specific manipulator position in the workspace. This should allow the model to learn from fewer data points, which is critical when the learning is performed directly on the real-manipulator data. The trajectories used for training contain points separated by $\Delta t = \frac{1}{F}$ s where $F = 20$ Hz. In case of any encoder inaccuracies of consistent magnitude, a larger time span between sample points allows decreasing their influence. In addition, larger Δt also reduces the influence of latency. The inner-loop feedback controller of the Baxter joints operates at a much higher frequency.

For approximating the forward dynamics of the manipulator, we define the FIN model: a simple NN consisting of three fully connected layers. It takes a 42-element vector as an input and produces a 14-element vector $[\theta_a(t+1), \dot{\theta}_a(t+1)]$, where $\theta_a(t+1)$ is a predicted state of the manipulator after executing the command $\theta_d(t+1)$ as it is. We use the ReLU activation function as non-linearity. FIN model has 5,294 weights in total. The following DWH model approximates the inverse dynamics and consists of two identical independent branches. Each branch takes in the original 42-element input concatenated with the 14-element output of FIN model, resulting in a 56-element input. Each branch then outputs a 7-element vector. One branch produces positions, the other – velocities. The architecture of a branch is as follows: first the LTI block with $a = 2$ and $b = 2$, which outputs 14 features (motivated to represent a rough position + velocity approximation). a and b define the polynomial order for the denominator and nominator of a rational transfer function (see [2] for details). The LTI block is followed by two fully connected layers. The result of these layers is then fed to the last LTI block which as well has $a = 2$ and $b = 2$ and produces the final 7-element vector. The overall architecture of the model is shown in Fig. 2. Each branch has 4,043 parameters, which results in total of $4,043 \times 2 = 8,086$ parameters for the DWH model. The LTI layers are parameterized in terms of rational transfer functions, and thus apply infinite impulse response (IIR) filtering to the input. Stacking this hardwired linear structure in multiple layers together with fully connected layers was shown [2] to approximate the complex non-linear dynamics. By providing the forward dynamics estimation obtained from the FIN, we allow the model to take into account the predicted future in which we would execute the next command as it is. This can be interpreted as a simplistic version of model-predictive control with only one step of looking ahead. As we show in our evaluation, this additional input improves the performance of the model.

C. Training

Given the set of desired trajectories θ_d , $\dot{\theta}_d$ and $\ddot{\theta}_d$, as well as the set of the actual robot responses θ_a , $\dot{\theta}_a$ and $\ddot{\theta}_a$, we train

²https://sdk.rethinkrobotics.com/wiki/Joint_Trajectory_Action_Server

the two-stage model in two steps.

First, we train the FIN model. Since we have the desired trajectories and the actual responses, the composition of the training input-output tuples is straightforward. We train the network using stochastic gradient descent (SGD) with a mini-batch of 32 data points in a fully supervised manner. We use the Adam optimizer with a learning rate of 10^{-4} to minimize the mean square error (MSE) loss and employ L2 regularization.

The training of the full FIN-DWH model is not as straightforward, because the ground-truth commands θ_f are unknown. It would be possible to employ ILC to obtain them, but since we do not have a good model for simulation, this would have to be done on the real robot, significantly increasing the number of robot-hours needed to produce such a dataset. Instead, we use the same data as for FIN training, and apply the Hindsight Experience Replay (HER) technique [28]. This method can be described in short as pretending that what we achieved was what we actually wanted and is commonly used in reinforcement learning to mitigate the negative effects of sparse delayed rewards. We use θ_a as the goal trajectory and then θ_d becomes the ground-truth input. After inverting the training examples, we perform the same training procedure as above: 32 data points per minibatch, Adam optimizer with learning rate of 10^{-4} with the MSE loss and L2 regularization. Note, that we keep the weights of the FIN frozen during the whole training of the FIN-DWH model. This ensures that the DWH model is supplied with forward dynamics prediction. Moreover, we have observed that allowing the network to update the FIN weights during the training led to inferior performance. This supports our idea that when training on a limited dataset, careful hard-wiring of dataflow structure is of high importance.

Although we train the model directly with trajectories with different velocity profiles, it is possible to train the model by gradually increasing the difficulty. Such training scenario would fit transparently into the real-world application scenario, when allocating robot hours only for training is infeasible and learning from the real tasks is required. In this setting, the tasks would be first executed with low velocities, to minimize the trajectory tracking inaccuracies and allow to successfully complete the tasks. Then, as the model improves the tracking accuracy, the velocity would be gradually increased.

IV. EVALUATION AND EXPERIMENTS

To evaluate our approach, we conduct experiments on the real Baxter robot. We compare the performance to the baseline Baxter controller (Inverse Dynamics Feed Forward Position control). In addition, we also compare our method against the three other models. The first model consists of three fully connected layers: $42 \times 64 \rightarrow 64 \times 32 \rightarrow 32 \times 14$. It has 5,294 parameters and is further referred to as FC. Note, that we also experimented with a larger FC model with four layers and 16,302 parameters, however, it did not demonstrate a superior performance. The second model is a three-layer RNN, which

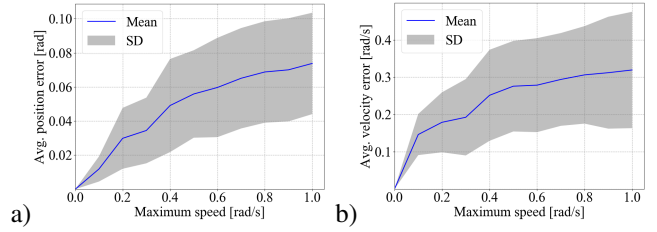


Fig. 3: Maximum joint speed in a trajectory vs: a) average position error, b) average velocity error, per trajectory point.

has 7,772 parameters and is referred to as RNN. It has analogous to FC structure: $(42 + h) \times 64 \rightarrow 64 \times 32 \rightarrow 32 \times 14$, with two additional layers to produce a 14-element hidden state h : $(42 + h) \times 32 \rightarrow 32 \times h$. The third model is dynoNet Wiener-Hammerstein (DWH) with 6,518 parameters. It has the same structure as the network described in Section III, except that it does not have a FIN model to estimate the future outcome of feed-forward control. All three models take 42-element vectors as input and produce a 14-element vector of feed-forward joint positions and velocities. All models use ReLU non-linearity and were trained on the same dataset until convergence, minimizing MSE loss with Adam optimizer. We did our best to find the best set of hyper-parameters for each model using grid search. For the FC model they were: learning rate of $2.0 \cdot 10^{-4}$ and minibatch size of 24 data points. For the RNN model they were: learning rate of $1, 5 \cdot 10^{-4}$ and minibatch size of 48 data points. For the DWH and FIN-DWH we used the same hyper-parameters (learning rate of 10^{-4} and minibatch size of 32 data points), since the core of both models is the same. This allows to better observe how the proposed two-stage architecture influences the performance of the model, compared to the same model without the future forward dynamics prediction step.

A. Quantitative Evaluation

To quantitatively evaluate the proposed approach, we generated 100 unseen trajectories for the left arm of Baxter. Although our model can be used for trajectories with arbitrary speed profile, we find it especially interesting to evaluate methods on the fast trajectories because in this case the compliance of the manipulator causes the most inaccuracies due to hard to model inertia effects on flexible joints. In Fig. 3 one can see the plot of maximum allowed speed in a trajectory vs. average cumulative error of joint position and velocity per point in a trajectory executed with a baseline controller. Clearly, the higher the maximum allowed speed is, the less accurate are the movements of the manipulator. The error grows slower with maximum speed increase because not all joints are able to reach this maximum speed during a trajectory. Thus, in our experiments, the maximum joint speed was set to 1.0 rad/s. We measure the average cumulative error per point in a trajectory for the joint position, joint velocity and end-effector position. We also measure the extra time needed to converge to the final point in a trajectory, as well as the

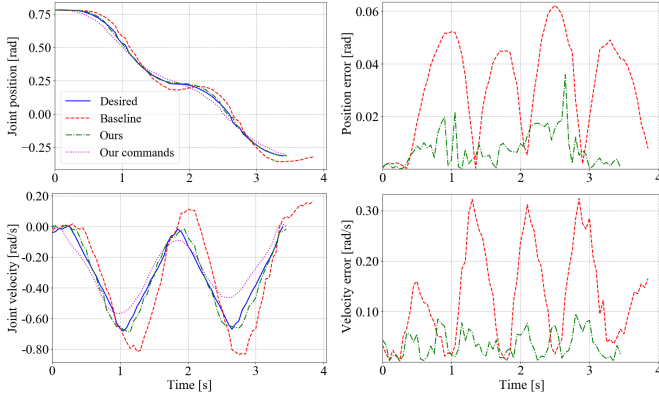


Fig. 4: Shoulder yaw trajectory. Top-left: position vs time. Bottom-left: velocity vs time. Top-right: position error vs time. Bottom-right: velocity error vs time. Blue (solid line): desired trajectory. Red (dashed line): execution with baseline controller. Green (dash-dotted line): execution with our method. Magenta (dotted line): feed-forward command of our method. The baseline controller alone leads to several major deviations, while our method allows to compensate for the inertia affecting the elastic joints and follows the trajectory with higher accuracy.

runtime of each model. Given an actual trajectory θ_a and a desired trajectory θ_d with T points and N joints, we calculate the average cumulative joint position error e_n per point as follows:

$$e_n = \frac{1}{T} \sum_{t=1}^T \sum_{n=1}^N |\theta_d(t, n) - \theta_a(t, n)|. \quad (2)$$

The procedure is analogous for the average joint velocity error and average end-effector position error.

In Table I, the average cumulative joint position and velocity errors are shown. In all tables we provide 95% confidence intervals (CI) of the mean. One can see that all the models made an improvement over the sole baseline controller. However, the FC model clearly shows the worst performance. This is the case because raw fully-connected layers do not have an underlying structure to capture the unknown dynamics. RNN and DWH models show quite similar performance, although DWH has slightly better results. Finally, the FIN-DWH improves over the plain DWH, demonstrating that the one-step prediction of the future allows producing a more accurate control input. In Table II, the average end-effector position error per point as well as the extra time to converge to the final point are shown. The extra time is defined as a difference between the actual trajectory execution time and the desired execution time. This difference arises when the state of the manipulator in the final trajectory point is significantly deviated from the desired state and extra time is needed to reach it. A similar tendency on the performance of the models can be observed. On average, the baseline controller has a 3 cm deviation from the desired path, while our method achieves an improvement of three times, reducing it to 1 cm on average.

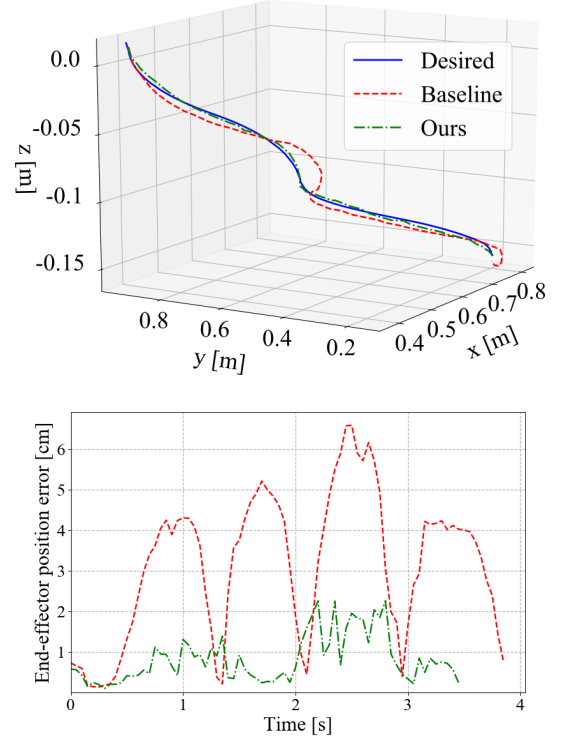


Fig. 5: An example trajectory. Top: path of the end-effector in 3D. Bottom: end-effector position error. Blue (solid line): desired trajectory. Red (dashed line): execution with baseline controller. Green (dash-dotted line): execution with our method. Execution with the baseline controller leads to several major deviations, while our approach allows to follow the trajectory with higher accuracy.

The extra time to arrive at the destination point is reduced by significant 92%, which happens because the motion is much smoother overall, making the immediate accurate arrival at the final point possible. The average runtimes per trajectory (usually consisting of 60-90 time steps) are as follows. FC: 0.031 s, RNN: 0.062 s, DWH: 0.067 s, FIN-DWH: 0.085 s. All computations were executed on an Intel i7-6700HQ 2.6 GHz CPU. Further improvements of the runtimes are possible. By listing them here, we give an intuition about the relative computational load of the compared models.

In Fig. 4, we show an example test trajectory of the shoulder yaw. One can see that the trajectory achieved solely by the baseline controller deviates from the target much more than the FIN-DWH generated trajectory. There are four big peaks of the position error and three big peaks of the velocity error. It can be noticed that the minima of the position error correspond to the maxima of the velocity error and vice versa. This can be explained by the controller trying to compensate for the position error by increasing velocity to "catch up", however, this consequently leads to the overshoot of the velocity, which is repeated. We attribute this effect to the compliance of the manipulator. It can be seen that our model learned to compensate for this, modifying the joint

TABLE I: Comparison of average cumulative joint position and velocity error per point.

Method	Joint position error, rad	Joint velocity error, rad/s
Baseline	0.069 ± 0.0062 (—)	0.325 ± 0.028 (—)
FC	0.047 ± 0.0061 (32%)	0.243 ± 0.021 (25%)
RNN	0.042 ± 0.0042 (39%)	0.227 ± 0.017 (30%)
DWH	0.040 ± 0.0037 (42%)	0.218 ± 0.016 (32%)
FIN-DWH	0.036 ± 0.0033 (47%)	0.213 ± 0.015 (35%)

95% confidence interval is provided after "±". Improvement over the baseline is in brackets.

commands when needed (see the dotted yellow line), achieving a much more precise trajectory tracking of both joint position and velocity. In Fig. 5, we show the path and the error of the end-effector position. One can see that with the baseline controller, the end-effector makes several swings, deviating significantly from the desired trajectory. The peaks of the deviations correspond to the peaks of shoulder yaw deviations, because shoulder joints are the ones most affected by inertia, bearing the weight of the whole arm. It is also worth noticing that the baseline trajectory takes around half a second of extra time to reach the final position, while our method requires almost no extra time. Overall, our approach showed better performance than the other three models, and significantly improved over the baseline controller, producing smooth high-velocity trajectories. Achieving higher accuracy over the raw DWH model, the addition of the FIN module was shown to be beneficial.

In addition, we execute 20 random trajectories from the previous experiment, while holding a payload of 0.25, 0.5 and 1.1 kg. Note, that the maximum payload for Baxter is 2.3 kg. In Table III, the average cumulative joint position and velocity errors are shown. It is possible to see that while carrying the smallest payload of 0.25 kg, both the baseline and the proposed FIN-DWH model perform similarly to the run without the payload (Table I). Increasing the payload to 0.5 kg causes a more noticeable decrease of the trajectory tracking accuracy. Finally, with a significant load of 1.1 kg, the trajectory tracking accuracy declines substantially, influenced by the inertia forces of larger magnitude. The largest deviations were observed in parts of the trajectories with relatively fast changes in acceleration. The proposed model was trained on payload-free trajectories and does not incorporate live feedback from the robot. Nevertheless, this experiment demonstrates that the learned dynamics model of the manipulator helps to reduce the negative impact of the unaccounted payload and achieve more accurate trajectory execution.

B. Practical Example

In Fig. 5, one can see that under the control of the baseline controller, the end-effector arrives to the goal pose in a curve, overshooting the desired path. Such behavior makes it very difficult to perform picking tasks with high speeds, because the end-effector often collides with objects at the pre-grasp

TABLE II: Comparison of average end-effector position error per point and average extra time to reach the endpoint.

Method	EEF position error, cm	Extra time, s
Baseline	2.981 ± 0.355 (—)	0.75 ± 0.091 (—)
FC	1.397 ± 0.243 (53%)	0.28 ± 0.029 (62%)
RNN	1.151 ± 0.142 (61%)	0.12 ± 0.013 (84%)
DWH	1.109 ± 0.131 (62%)	0.09 ± 0.010 (88%)
FIN-DWH	1.015 ± 0.122 (66%)	0.06 ± 0.008 (92%)

95% confidence interval is provided after "±". Improvement over the baseline is in brackets.

pose. This often results in failed grasping attempts and can potentially damage the robot and its surroundings. A typical mitigation approach would be to define another pre-grasp pose and, upon arrival there, continue at a very low speed to the actual pre-grasp pose. This slows down the task execution, however. In this example, we demonstrate the effectiveness of our approach by reaching a pre-grasp pose at high speed. We execute the same trajectory first with the baseline controller and then with the proposed model as an outer loop controller. Two sequences of pictures in Fig. 6 show the execution of these trajectories. In Frame c) the baseline controller deviates from the path, leading to the collision with the object in Frame d). This results in the object being tipped over in Frame e). In contrast, our method tracks the trajectory more accurate and has a smoother velocity profile, which avoids the typical overshooting. This results in the successfully reached pre-grasp pose without colliding with the object. A video of the experiment is available on our website³.

C. Discussion

The conducted experiments demonstrated that the proposed two-stage model architecture with one-step future prediction achieves a substantial improvement of the trajectory tracking accuracy, compared to the sole baseline controller, and outperforms the trivial models. The raw DWH model represents a reactive policy, which approximates the inverse dynamics

³https://www.ais.uni-bonn.de/videos/IRC_2021_Pavlichenko

TABLE III: Comparison of average cumulative joint position and velocity error per point while carrying a payload.

Payload, kg	Method	Joint position error, rad	Joint velocity error, rad/s
0.25	Baseline	0.074 ± 0.014	0.341 ± 0.064
	FIN-DWH	0.039 ± 0.008 (47%)	0.217 ± 0.035 (36%)
0.5	Baseline	0.080 ± 0.016	0.347 ± 0.067
	FIN-DWH	0.042 ± 0.009 (47%)	0.226 ± 0.038 (34%)
1.1	Baseline	0.108 ± 0.019	0.378 ± 0.083
	FIN-DWH	0.071 ± 0.013 (34%)	0.256 ± 0.046 (32%)

95% confidence interval is provided after "±". Improvement over the baseline is in brackets.

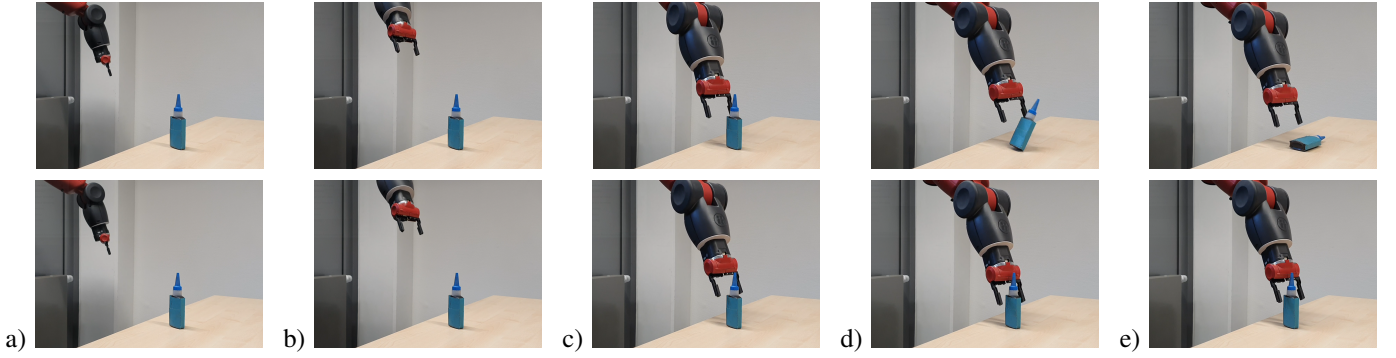


Fig. 6: Pre-grasp trajectory execution. Top row: using only the baseline controller. Bottom row: using our model as an outer-loop controller. a) Start. b) Approach. c) Close proximity to the goal. In case of the baseline controller, inaccurate motion leads to a collision with the object. d-e) Pre-grasp pose reached. However, in case of the baseline controller, the collision led to tipping over the object. Our approach allowed to reach the goal position smoothly.

of the manipulator. In contrast, the future prediction step of the FIN-DWH model allows to utilize the learned forward dynamics of the flexible-joint manipulator in order to find a corrective action, taking into account the predicted future inaccuracies. By making use of both forward and inverse dynamics, such an architecture allows extracting more useful knowledge from a significantly limited real-world dataset. The manipulators with flexible joints often do not have accurate dynamics models to perform training in simulation and/or have instance-specific dynamics properties due to wear and tear. Thus, the proposed method is useful for improving trajectory tracking of such robots. The achieved improvements do not come at a cost of increased stiffness and are based solely on learned dynamics of coupled joints.

One could argue that it is possible to develop a classical controller with higher accuracy of trajectory tracking than the Baxter baseline. However, this would require tedious tuning performed by an experienced professional on a time-span of several days. Moreover, such procedure is typically instance-specific and may need to be repeated after wear and tear accumulate during the use of the robot. At the same time, the proposed framework does not rely on any robot-specific parameters and requires only 45 minutes of data for the models to be trained. In addition, the learned models can be easily retrained to adjust for wear and tear, using the most recent recorded trajectories at any time, since the training procedure takes several hours on a regular computer. Finally, developing and tuning a classical controller which takes into account coupled joint dynamics is an extremely challenging task, which we resolve by following the data-driven approach.

The main limitation of the presented approach is its open-loop nature. Thus, any unexpected disturbances are unaccounted for, and remain for the inner-loop controller to be dealt with. As it was shown in the experiment with payloads, an increase of such unaccounted disturbance degrades the effectiveness of the method. Nevertheless, the model, learned by our method still achieves more accurate trajectory tracking, compared to the sole baseline controller, even with

unaccounted disturbances. The underlying low-level classical controller provides guarantees on system stability. As the proposed method does not use feedback, it can be easily integrated into a software pipeline as a top-level addition over the existing low-level controller. Our methodology is agnostic of the type of the underlying classical controller and does not have robot-specific parameters, making it possible to apply it to most robotic manipulators without major changes.

V. CONCLUSION

We presented a two-stage model based on linear time-invariant (LTI) dynamical operators for feed-forward outer loop control of a manipulator with flexible joints and unknown complex dynamics. The first part of the model estimates the future state of the system one step ahead with an unchanged control command. This estimation is used to augment the input for the second part of the model, which produces feed-forward joint position and velocity commands. The aim of this two-stage architecture is to push the model from reactive policy behavior towards more intelligent planning. The model was trained with backpropagation on a small 45 min real-robot dataset. The approach was evaluated on the Baxter robot. Ablation study showed that one-step future prediction improved the performance. Our approach improved the trajectory tracking accuracy over the baseline controller: by 47% and 35% for the joint position and velocity tracking respectively, which resulted in 66% improvement of the end-effector position tracking. This contributed to fast and smooth trajectory executions which required 92% less extra time to reach the endpoint, allowing to perform tasks faster.

Future work includes exploring the possibility of employing a recurrent hierarchical model which is capable of looking several steps into the future, representing a model-predictive control approach. In addition, applying such a model in a closed-loop fashion in combination with online learning would allow obtaining a very flexible universal approach which would have the potential to further improve the trajectory tracking performance.

REFERENCES

- [1] S. Arimoto. "Learning control theory for robotic motion". In: *Int. Journal of Adaptive Control and Signal Processing* 4.6 (1990), pp. 543–564.
- [2] M. Forgione and D. Piga. "dynoNet: A neural network architecture for learning dynamical systems". In: *Int. Journal of Adaptive Control and Signal Processing* 35.4 (2021), pp. 612–626.
- [3] K. Greff et al. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2017), pp. 2222–2232.
- [4] Z. Wang, W. Yan, and T. Oates. "Time series classification from scratch with deep neural networks: A strong baseline". In: *IEEE Int. Joint Conf. on Neural Networks (IJCNN)*. 2017, pp. 1578–1585.
- [5] D. Mayne. "A Second-order Gradient Method for Determining Optimal Trajectories of Non-linear Discrete-time Systems". In: *Int. Journal of Control* 3 (1966), pp. 85–95.
- [6] J. Yiming et al. "A Brief Review of Neural Networks Based Learning and Control and Their Applications for Robots". In: *Complexity* 2017 (2017), pp. 1–14.
- [7] L. Jin et al. "Robot manipulator control using neural networks: A survey". In: *Neurocomputing* 285 (2018), pp. 23–34.
- [8] C. Yang et al. "Neural-Learning-Based Telerobot Control With Guaranteed Performance". In: *IEEE Transactions on Cybernetics* 47.10 (2016), pp. 3148–3159.
- [9] Q. Chen et al. "Adaptive neural dynamic surface sliding mode control for uncertain nonlinear systems with unknown input saturation". In: *Int. Journal of Advanced Robotic Systems (IJARS)* 13.5 (2016).
- [10] H. Han et al. "Nonlinear Model Predictive Control Based on a Self-Organizing Recurrent Neural Network". In: *IEEE Transactions on Neural Networks and Learning Systems* 27 (2016), pp. 402–415.
- [11] D. Xia, L. Wang, and T. Chai. "Neural-Network-Friction Compensation-Based Energy Swing-Up Control of Pendubot". In: *IEEE Transactions on Industrial Electronics* 61 (2014), pp. 1411–1423.
- [12] C.-H. Lu. "Wavelet Fuzzy Neural Networks for Identification and Predictive Control of Dynamic Systems". In: *IEEE Transactions on Industrial Electronics* 58 (2011), pp. 3046–3058.
- [13] E. Rueckert et al. "Learning inverse dynamics models in $O(n)$ time with LSTM networks". In: *IEEE-RAS Int. Conf. on Humanoid Robotics (Humanoids)*. 2017, pp. 811–816.
- [14] M. Saveriano et al. "Data-efficient control policy search using residual dynamics learning". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2017, pp. 4709–4715.
- [15] J. Mahler et al. "Learning accurate kinematic control of cable-driven surgical robots using data cleaning and Gaussian Process Regression". In: *IEEE Int. Conf. on Automation Science and Engineering (CASE)*. 2014, pp. 532–539.
- [16] K. Morse et al. "Learning State-Dependent Losses for Inverse Dynamics Learning". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2020, pp. 5261–5268.
- [17] K. Patan, M. Patan, and D. Kowalów. "Neural networks in design of iterative learning control for nonlinear systems". In: *IFAC-PapersOnLine* 50 (2017), pp. 13402–13407.
- [18] W. Zuo and L. Cai. "A New Iterative Learning Controller Using Variable Structure Fourier Neural Network". In: *IEEE Transactions on Systems, Man, and Cybernetics* 40 (2010), pp. 458–468.
- [19] M. Schwarz and S. Behnke. "Compliant Robot Behavior Using Servo Actuator Models Identified by Iterative Learning Control". In: *RoboCup 2013: Robot World Cup XVII*. 2014, pp. 207–218.
- [20] K. Pereida et al. "Transfer learning for high-precision trajectory tracking through L1 adaptive feedback and iterative learning". In: *Int. Journal of Adaptive Control and Signal Processing* 33.2 (2018), pp. 388–409.
- [21] S. Chen and J. T. Wen. "Industrial Robot Trajectory Tracking Control Using Multi-Layer Neural Networks Trained by Iterative Learning Control". In: *Robotics* 10.50 (2021).
- [22] S. Chen and J. T. Wen. "Neural-Learning Trajectory Tracking Control of Flexible-Joint Robot Manipulators with Unknown Dynamics". In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. Vol. 2019. 2019, pp. 128–135.
- [23] M. Schuster and K. K. Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45 (1997), pp. 2673–2681.
- [24] H. A. Talebi, R. V. Patel, and K. Khorasani. "Inverse dynamics control of flexible-link manipulators using neural networks". In: *IEEE Int. Conf. on Robotics and Automation (ICRA)*. 1998, pp. 806–811.
- [25] Q. Li et al. "Deep neural networks for improved, impromptu trajectory tracking of quadrotors". In: *IEEE Int. Conf. on Robotics and Automation (ICRA)*. 2017, pp. 5183–5189.
- [26] W. Zeng et al. "DSDNet: Deep Structured Self-driving Network". In: *European Conference on Computer Vision (ECCV)*. 2020, pp. 156–172.
- [27] F. Giri and E.-W. Bai. "Block Oriented Nonlinear System Identification". In: *Lecture Notes in Control and Information Sciences*. Vol. 2010. 2010, pp. 746–758.
- [28] M. Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems 30 (NIPS)*. 2017, pp. 5048–5058.