Proceedings of Machine Learning Research vol 120:1–10, 2020 2nd Annual Conference on Learning for Dynamics and Control

Encoding Physical Constraints in Differentiable Newton-Euler Algorithm

Giovanni Sutanto* GSUTANTO@USC.EDU
Department of Computer Science, University of Southern California

Austin S. Wang WANGAUSTIN@FB.COM
Yixin Lin YIXINLIN@FB.COM
Mustafa Mukadam MUKADAM@FB.COM
Facebook Artificial Intelligence Research

Gaurav S. Sukhatme GAURAV@USC.EDU
Department of Computer Science, University of Southern California

Akshara Rai AKSHARARAI@FB.COM
Franziska Meier FMEIER@FB.COM
Facebook Artificial Intelligence Research

Editors: A. Bayen, A. Jadbabaie, G. J. Pappas, P. Parrilo, B. Recht, C. Tomlin, M. Zeilinger

Abstract

The recursive Newton-Euler Algorithm (RNEA) is a popular technique for computing the dynamics of robots. RNEA can be framed as a differentiable computational graph, enabling the dynamics parameters of the robot to be learned from data via modern auto-differentiation toolboxes. However, the dynamics parameters learned in this manner can be physically implausible. In this work, we incorporate physical constraints in the learning by adding structure to the learned parameters. This results in a framework that can learn physically plausible dynamics via gradient descent, improving the training speed as well as generalization of the learned dynamics models. We evaluate our method on real-time inverse dynamics control tasks on a 7 degree of freedom robot arm, both in simulation and on the real robot. Our experiments study a spectrum of structure added to the parameters of the differentiable RNEA algorithm, and compare their performance and generalization.

Keywords: learning, structure, rigid body parameters, differentiable, recursive Newton-Euler algorithm, inverse dynamics

1. Introduction

An accurate dynamics model is key to compliant force control of robots, and there is a rich history of learning of such models for robotics (An et al., 1988; Murray et al., 1994; Atkeson et al., 1986). With an accurate dynamics model, inverse dynamics can be used as a policy to predict the torques required to achieve a desired joint acceleration, given the state of the robot (Murray et al., 1994).

Due to their widespread utility, robot dynamics have been learned in many ways. One way is to use a purely data-driven approach with parametric models (Hitzler et al., 2019), non-parametric models (Nguyen-Tuong et al., 2008), and learning error-models (Kappler et al., 2017), in a supervised or self-supervised fashion. However, these purely data-driven approaches typically suffer

* This work was done when Giovanni Sutanto was an intern at Facebook Artificial Intelligence Research.

© 2020 G. Sutanto, A.S. Wang, Y. Lin, M. Mukadam, G.S. Sukhatme, A. Rai & F. Meier.

from a lack of generalization to previously unexplored parts of the state space. Alternatively, Atkeson et al. (1986) recast the dynamics equations such that inertial parameters are a linear function of state-dependent quantities, given the joint torques. While inferior to unstructured approaches in terms of flexibility to fit data, this approach typically provides superior generalization capabilities. Recently, Lutter et al. (2019); Gupta et al. (2019) learn the parameters of Lagrangian dynamics, incorporating benefits of flexible function approximation into structured models. However, these approaches ignore some of the physical relationships and constraints on the learned parameters, which can lead to physically implausible dynamics.

In this work, we combine the modern approach of parameter learning with the structured approach to inverse dynamics learning, similar to Ledezma and Haddadin (2017). We implement the Recursive Newton-Euler algorithm in PyTorch (Paszke et al., 2017), allowing the inertial parameters to be learned using gradient descent and automatic-differentiation for gradient computation. The benefit, over traditional least-squares implementations, is that we can easily explore various re-parameterizations of the inertial parameters and additional constraints that encode physical consistency, such as Wensing et al. (2018), without the need for linearity.

In that context, the contribution of this paper is threefold: first, we present several re-parameterizations of the inertial parameters, which allow us to learn physically plausible parameters using a differentiable recursive Newton-Euler algorithm. Second, we show that these re-parameterizations help in improving training speed as well as generalization capability of the model to unseen situations. Third, we evaluate a spectrum of structured dynamics learning approaches on a simulated and real 7 degree-of-freedom robot manipulator.

Our results show that adding such structure to the learning can improve the learning speed as well as generalization abilities of the dynamics model. Our models can generalize with much lesser data, and need much fewer training epochs to converge. With our learned dynamics, we see reduced contributions of feedback terms in control, resulting in more compliant motions.

2. Background and Related Work

The dynamics of a robot manipulator are a function of the joint torque $\boldsymbol{\tau}$, joint acceleration $\ddot{\mathbf{q}}$ and joint position and velocity $\mathbf{q}, \dot{\mathbf{q}}$:

$$\boldsymbol{\tau} = \mathbf{f}_{ID}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{1}$$

with $\mathbf{H}(\mathbf{q}), \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}), \mathbf{g}(\mathbf{q})$ are the system inertia matrix, Coriolis matrix, and gravity force, respectively. $\mathbf{f}_{ID}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is the inverse dynamics model that returns the torques that can achieve a desired joint acceleration, given the current joint positions and velocities.

The Recursive Newton-Euler Algorithm (RNEA) (Luh et al., 1980; Featherstone, 2007) is a computationally efficient method of computing the inverse dynamics, which scales linearly with respect to the number of the degrees-of-freedom of the robot.

2.1. Learning models for model-based control

Accurate inverse dynamics models are crucial for compliant force controlled robots, and hence widely studied in robotics. Previously, researchers have used unstructured multi-layer perceptron (MLP) to learn the complete inverse dynamics (Jansen, 1994; Hitzler et al., 2019) or a residual component of the inverse dynamics (Kappler et al., 2017). Nguyen-Tuong et al. (2008) compare

non-parametric methods like locally weighted projection regression (LWPR), support vector regression (SVR) and Gaussian processes regression (GPR) for learning inverse dynamics models.

Recently Lutter et al. (2019) and concurrently Gupta et al. (2019, 2020), proposed a semi-structured learning method for the Lagrangian dynamics of a manipulator, called Deep Lagrangian Networks (DeLaN). In DeLaN, some of the physical constraints of Lagrangian dynamics are obeyed. For example, the inertia matrix is parametrized to be symmetric positive definite. Moreover, the relationship between coriolis and centrifugal terms and the inertia matrix and joint velocities (Murray et al., 1994) is satisfied via automatic differentiation. Similarly, the gravity term is derived from a neural network which takes generalized coordinates as input, representing the potential energy. However, other constraints in the dynamics, such as the triangle inequality in the principal moments (Traversaro et al., 2016) of the inertia matrix are not considered. Moreover, neural networks can be sensitive to the chosen architecture and need variations in input data to generalize to new situations. Similar to DeLaN, Hamiltonian Neural Networks (HNN) (Greydanus et al., 2019) predict the Hamiltonian (instead of Lagrangian) of a dynamical system.

Many previous works in parameter identification boil down to setting up a least square problem with some (hard) constraints (Wensing et al., 2018; Mistry et al., 2009; Kozlowski, 2012), followed by solving a convex optimization problem. On the other hand, our method incorporates hard constraints as structure in learning representations of the parameters, and then performs back-propagation on the computational graph for optimization. As a result, it is not limited to learning linear parameters, and can generalize to a larger range of problems. Moreover, our approach can be applied to an online learning setup: for example when the robot carries an additional mass (an object) on one of its links, our approach can adapt the dynamics parameters online, as the robot continues to operate and the data is collected in batches. Traditionally, online learning approaches –including adaptive control (Slotine and Li, 1987)– do not guarantee physical plausibility of the learned dynamics parameters. On the other hand, more modern system identification methods which incorporate hard constraints on physical parameters (Wensing et al., 2018; Mistry et al., 2009; Kozlowski, 2012) require collecting data in the new setting before optimizing the new dynamics parameters.

Our work is closely related to the work by Ledezma and Haddadin (2017), in the sense that our work is also derived from the Newton-Euler formulation of inverse dynamics. However, in our work, we emphasize more on how incorporating structure in learning dynamics parameters helps with improving the training speed and generalization capability of the model. Moreover, we also compare our method with the state-of-the-art semi-structured DeLaN and an unstructured MLP.

3. Encoding Physical Consistency in Newton-Euler Equations

The Newton-Euler equations can be implemented as a differentiable computational graph, e.g. with PyTorch (Paszke et al., 2017), which we call differentiable NEA (DiffNEA). The parameters of DiffNEA, e.g. the inertial parameters $\boldsymbol{\theta}$, can now be optimized via gradient descent utilizing automatic differentiation to compute the gradients. Although both kinematics and dynamics parameters are involved in Newton-Euler algorithm, in this paper we study the optimization of only the dynamics parameters, assuming the kinematics specification of the robot is correct and fixed.

Specifically, we aim to optimize parameters $\boldsymbol{\theta}$ of the Newton-Euler equations, such that the inverse dynamics loss is minimized

$$L_{ID} = \sum_{t=1}^{T} \|\boldsymbol{\tau}_t - \mathbf{f}_{NE}(\mathbf{q}_t, \dot{\mathbf{q}}_t, \ddot{\mathbf{q}}_t; \boldsymbol{\theta})\|_2^2 \,. \tag{2}$$

Typically, $\boldsymbol{\theta}$ is a collection of inertial parameters $\boldsymbol{\theta}_i = [m, \boldsymbol{\theta_h}, I_{xx}, I_{xy}, I_{xz}, I_{yy}, I_{yz}, I_{zz}]^T \in \mathbb{R}^{10}$ per link, where m is the link mass, $\boldsymbol{\theta_h} = [h_x, h_y, h_z] = m\boldsymbol{c}$ with $\boldsymbol{c}$ being the CoM, and the last 6 parameters representing the rotational inertia matrix $\boldsymbol{I}_C$ (Atkeson et al., 1986). When optimizing $\boldsymbol{\theta}$ physical consistency of the estimated parameters is not guaranteed. Enforcing physical constraints on the parameters can be done through explicit constraints (Traversaro et al., 2016; Wensing et al., 2018), which requires constrained optimization algorithms to find a solution. In the following, we discuss and propose several possible parameter representations $\boldsymbol{\theta}$, which encode increasingly more physical consistency implicitly and allows us to perform unconstrained gradient descent.

3.1. Unstructured Mass and Rotational Inertia Matrix (DiffNEA No Str)

We start out with the simplest representation, with an *unconstrained* mass value θ_m and 9 *unconstrained* parameters for the rotational inertia matrix:

$$\boldsymbol{\theta}_{\text{No Str}} = \begin{bmatrix}\theta_m & \boldsymbol{\theta_h} & \theta_{\boldsymbol{I}_C 1} & \theta_{\boldsymbol{I}_C 2} & \theta_{\boldsymbol{I}_C 3} & \theta_{\boldsymbol{I}_C 4} & \theta_{\boldsymbol{I}_C 5} & \theta_{\boldsymbol{I}_C 6} & \theta_{\boldsymbol{I}_C 7} & \theta_{\boldsymbol{I}_C 8} & \theta_{\boldsymbol{I}_C 9}\end{bmatrix} \tag{3}$$

This parametrization does not encode any physical constraints and only serves as baseline.

3.2. Symmetric Rotational Inertia Matrix (DiffNEA Symm)

In this parametrization, we explicitly construct the rotational matrix as a symmetric matrix, with only 6 learnable parameters. Furthermore, we represent the link mass as: $m = (\theta_{\sqrt{m}})^2 + b$, where $b > 0$ is a (non-learnable) small positive constant to ensure $m > 0$. Thus the learnable parameters of this representation are

$$\boldsymbol{\theta}_{\text{Symm}} = \begin{bmatrix}\theta_{\sqrt{m}} & \boldsymbol{\theta_h} & \theta_{\boldsymbol{I}_C 1} & \theta_{\boldsymbol{I}_C 2} & \theta_{\boldsymbol{I}_C 3} & \theta_{\boldsymbol{I}_C 4} & \theta_{\boldsymbol{I}_C 5} & \theta_{\boldsymbol{I}_C 6}\end{bmatrix} \tag{4}$$

This parameter representation enforces positive mass estimates and symmetric –but not necessarily positive definite– rotational inertia matrices.

3.3. Symmetric Positive Definite Rotational Inertia Matrix (DiffNEA SPD)

Next, we introduce a change of variables, to enforce positive definiteness of the rotational inertia matrix. We construct the lower triangular matrix:

$$\mathbf{L} = \begin{bmatrix}\theta_{\text{LI}_1} & 0 & 0 \\ \theta_{\text{LI}_4} & \theta_{\text{LI}_2} & 0 \\ \theta_{\text{LI}_5} & \theta_{\text{LI}_6} & \theta_{\text{LI}_3}\end{bmatrix} \tag{5}$$

and construct the rotational inertia matrix $\boldsymbol{I}_C$ via Cholesky decomposition plus a small positive bias on the diagonal: $\boldsymbol{I}_C = \mathbf{L}\mathbf{L}^{\mathrm{T}} + b\mathbf{I}_{3\times3}$. $\mathbf{I}_{3\times3}$ is a 3×3 identity matrix and $b > 0$ is a (non-learnable) small positive constant to ensure positive definiteness of $\boldsymbol{I}_C$. The learnable parameters are:

$$\boldsymbol{\theta}_{\text{SPD}} = \begin{bmatrix}\theta_{\sqrt{m}} & \boldsymbol{\theta_h} & \theta_{\text{LI}_1} & \theta_{\text{LI}_2} & \theta_{\text{LI}_3} & \theta_{\text{LI}_4} & \theta_{\text{LI}_5} & \theta_{\text{LI}_6}\end{bmatrix} \tag{6}$$

While this representation enforces positive mass and positive definite inertia matrices, it could still lead to inertia estimates that are not physically plausible, as discussed in Traversaro et al. (2016). To achieve full consistency, the estimated inertia matrix also needs to fulfill the triangular inequality of the principal moments of inertia of the 3D inertia matrices (Traversaro et al., 2016).

3.4. Triangular Parameterized Rotational Inertia Matrix (DiffNEA Tri)

To encode the triangular inequality constraints, we first decompose the rotational inertia matrix as:

$$\boldsymbol{I}_C = \boldsymbol{R}\boldsymbol{J}\boldsymbol{R}^{\mathrm{T}} \tag{7}$$

where $\boldsymbol{R} \in SO(3)$ is a rotation matrix, and $\boldsymbol{J}$ is a diagonal matrix containing the principal moments of inertia J_1, J_2, J_3. The principal moments of inertia are all positive ($J_1 > 0$, $J_2 > 0$, $J_3 > 0$) such that $\boldsymbol{I}_C$ is positive definite. In addition to the positiveness of the principal moments of inertia, a physically realizable rotational inertia matrix $\boldsymbol{I}_C$ needs to have $\boldsymbol{J}$ that satisfies the triangular inequalities (Wensing et al., 2018; Traversaro et al., 2016):

$$J_1 + J_2 \geq J_3, \quad J_2 + J_3 \geq J_1, \quad J_1 + J_3 \geq J_2 \tag{8}$$

In Wensing et al. (2018); Traversaro et al. (2016) the triangular inequality and $\boldsymbol{R} \in SO(3)$ constraints were encoded explicitly, here we propose a change of variables such that these constraints are encoded implicitly allowing us to utilize the standard gradient based optimizers of toolboxes such as PyTorch (Paszke et al., 2017).

We start out by introducing a set of unconstrained parameters $\boldsymbol{\theta}_{\text{RAA}} = \begin{bmatrix}\theta_{\text{RAA1}} & \theta_{\text{RAA2}} & \theta_{\text{RAA3}}\end{bmatrix}^{\mathrm{T}}$ that represent an axis-angle orientation from which the rotation matrix $\boldsymbol{R}$ can be recovered by applying the exponential map to the skew-symmetric matrix recovered from $\boldsymbol{\theta}_{\text{RAA}}$.

$$\boldsymbol{R} = exp\left(\begin{bmatrix} 0 & -\theta_{\text{RAA3}} & \theta_{\text{RAA2}} \\ \theta_{\text{RAA3}} & 0 & -\theta_{\text{RAA1}} \\ -\theta_{\text{RAA2}} & \theta_{\text{RAA1}} & 0 \end{bmatrix}\right) \tag{9}$$

where $exp(.)$ is the exponential mapping that maps $\boldsymbol{\theta}_{\text{RAA}}$, a member of $so(3)$ group, to $\boldsymbol{R}$, a member of the $SO(3)$ group (Murray et al., 1994).
Second, to satisfy triangular inequality constraints in Eq. 8 above, we can parameterize J_1, J_2, and J_3 as the length of the sides of a triangle. The length of the first 2 sides of the triangle are encoded by J_1 and J_2, and the length of the 3rd side is computed as

$$J_3 = \sqrt{J_1^2 + J_2^2 - 2J_1J_2\cos\alpha} \tag{10}$$

with $0 < \alpha < \pi$. To encode that $J_1, J_2 > 0$, and $0 < \alpha < \pi$ we choose the following parametrization:

$$J_1 = (\theta_{\sqrt{J_1}})^2 + b, \quad J_2 = (\theta_{\sqrt{J_2}})^2 + b, \quad \alpha = \pi\text{sigmoid}(\theta_a) \tag{11}$$

Thus, the learnable parameters of this parametrization are:

$$\boldsymbol{\theta}_{\text{TRI}} = \begin{bmatrix}\theta_{\sqrt{m}} & \boldsymbol{\theta_h} & \theta_{\text{RAA1}} & \theta_{\text{RAA2}} & \theta_{\text{RAA3}} & \theta_{\sqrt{J_1}} & \theta_{\sqrt{J_2}} & \theta_a\end{bmatrix} \tag{12}$$

Note, even though the underlying learnable parameter vector $\boldsymbol{\theta}_{\text{TRI}}$ in Eq. 12 is unconstrained during the parameter optimization via gradient descent, the intermediate parameters $J_1, J_2, J_3, \boldsymbol{R}$ always satisfy the hard constraints for physical consistency, i.e. that J_1, J_2, and J_3 are all positive and satisfy the triangle inequality constraints in Eq. 8, as well as $\boldsymbol{R} \in SO(3)$. In other words, it *always* lies within the constraint manifold during optimization.

3.5. Covariance Parameterized Rotational Inertia Matrix (DiffNEA Cov)

Alternatively, the triangular inequality constraint in Eq. 8 can be rewritten as (Wensing et al., 2018):

$$\mathbf{\Sigma}_C = \frac{1}{2}Tr(\boldsymbol{I}_C)\mathbf{I}_{3\times3} - \boldsymbol{I}_C \succ \mathbf{0} \tag{13}$$

which provides a somewhat easier and more intuitive representation. Again, in Wensing et al. (2018) this constraint was imposed explicitly, through linear matrix inequalities. Here, we encode the constraint $\mathbf{\Sigma}_C \succ \mathbf{0}$ implicitly by enforcing a Cholesky decomposition plus a small positive bias b on the diagonal: $\mathbf{\Sigma}_C = \mathbf{L}\mathbf{L}^{\mathrm{T}} + b\mathbf{I}_{3\times3}$. We parametrize this lower triangular matrix $\mathbf{L}$:

$$\mathbf{L} = \begin{bmatrix} \theta_{\mathrm{L}\Sigma_1} & 0 & 0 \\ \theta_{\mathrm{L}\Sigma_4} & \theta_{\mathrm{L}\Sigma_2} & 0 \\ \theta_{\mathrm{L}\Sigma_5} & \theta_{\mathrm{L}\Sigma_6} & \theta_{\mathrm{L}\Sigma_3} \end{bmatrix} \tag{14}$$

and recover the rotational inertia matrix as:

$$\boldsymbol{I}_C = Tr(\mathbf{\Sigma}_C)\mathbf{I}_{3\times3} - \mathbf{\Sigma}_C \tag{15}$$

with $Tr()$ is the matrix trace operation, and $\mathbf{I}_{3\times3}$ is a 3×3 identity matrix. The learnable parameters of this parametrization are:

$$\boldsymbol{\theta}_{\mathrm{COV}} = \begin{bmatrix} \theta_{\sqrt{m}} & \boldsymbol{\theta_h} & \theta_{\mathrm{L}\Sigma_1} & \theta_{\mathrm{L}\Sigma_2} & \theta_{\mathrm{L}\Sigma_3} & \theta_{\mathrm{L}\Sigma_4} & \theta_{\mathrm{L}\Sigma_5} & \theta_{\mathrm{L}\Sigma_6} \end{bmatrix} \tag{16}$$

This parametrization also generates fully consistent inertial parameter estimates, like the previous parametrization, however it is less complex to implement.

4. Experiments

In this Section, we evaluate our Torch implementation of the Newton-Euler algorithm, with the parametrizations introduced in Section 3. We study how the parametrizations affect convergence speed when training the parameters, and how well the dynamics generalize to unseen scenarios. Furthermore, we compare a spectrum of structured dynamics learning approaches, starting from an unstructured MLP, and semi-structured models like DeLaN, to our highly structured approach. We start out with simulation experiments and then provide real system results on Kuka iiwa7 robot.

4.1. Simulation

In simulation, we collect training data on a simulated KUKA IIWA environment in PyBullet (Coumans and Bai, 2016–2018), by tracking sine waves in each joint with the ground-truth inverse dynamics model. The sine waves have time periods of $[23.0, 19.0, 17.0, 13.0, 11.0, 7.0, 5.0]$ seconds in each joint, and amplitudes $[0.7, 0.5, 0.5, 0.5, 0.65, 0.65, 0.7]$ times the maximum absolute movement in each joint. All dynamics models are trained with this sine wave motion dataset. All feed-forward neural networks involved in MLP and DeLaN models have $[32, 64, 32]$ nodes in the hidden layer with $tanh()$ activation functions.

We perform each experiment with 5 different random seeds — which affects the random initialization as well as the random end-effector goal position to be tracked during generalization tests — and then compute the performance statistics with mean and standard deviation across these.

Table 1: Comparison between models trained to optimize L_{ID} on the sine motion dataset from simulation, in terms of training speed, joint position ($\mathbf{q}$) and velocity ($\dot{\mathbf{q}}$) tracking, and generalization performance: end-effector position ($\mathbf{x}$) and velocity ($\dot{\mathbf{x}}$) tracking unseen end-effector reaching tasks.

Model	Sine Tracking (NSME)			End-Effector Tracking (NSME)	
	# Training Epochs	$\mathbf{q}$ Tracking	$\dot{\mathbf{q}}$ Tracking	$\mathbf{x}$ Tracking	$\dot{\mathbf{x}}$ Tracking
Ground Truth	N/A	0.000	0.000	0.005±0.006	0.008±0.010
MLP	23±3	0.000	0.001	0.256±0.405	5.542±6.980
DeLaN	58±19	0.000	0.001	0.016±0.008	0.254±0.278
DiffNEA NoStr	61±53	0.000	0.005±0.006	0.005±0.006	0.037±0.057
DiffNEA Symm	8±4	0.000	0.000	0.005±0.006	0.011±0.011
DiffNEA SPD	2±1	0.000	0.000	0.005±0.006	0.008±0.010
DiffNEA Tri	2±1	0.000	0.000	0.005±0.006	0.008±0.010
DiffNEA Cov	2±1	0.000	0.000	0.005±0.006	0.008±0.010

4.1.1. TRAINING SPEED, GENERALIZATION PERFORMANCE, AND EFFECTIVENESS OF INVERSE DYNAMICS LEARNING

We train each model until it achieves at most a normalized mean squared error (NMSE) of 0.1 for all joints. We record total epochs of training required to reach that level of accuracy and store the model once this accuracy has been achieved. Next, we evaluate model on tracking a) the sine motion itself (which the parameters were fitted for), and b) on a series of 5 operational space control tasks. For the second task, we use a velocity-based operational space controller as described in (Nakanishi et al., 2008), and use the learned inverse dynamics model within that controller. The results for convergence speed and tracking performance are averaged across the 5 random seeds and summarized in Table 1. We measure the tracking performance through NMSE, which is the mean squared tracking error normalized by the variance of the target trajectory. The better the tracking, the less the controller relies on the feedback component, and more on the inverse dynamics model prediction. The behavior becomes more compliant as the contribution of linear feedback goes down.

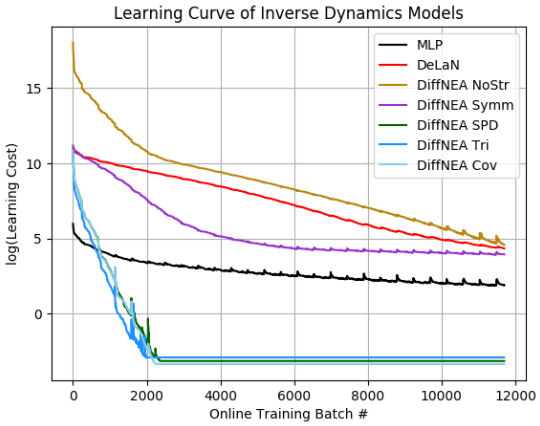
Table 1, shows that the parametrizations of $\boldsymbol{\theta}_{\text{SPD}}, \boldsymbol{\theta}_{\text{TRI}}, \boldsymbol{\theta}_{\text{COV}}$ outperform the less constrained learning, by training faster as well as low NMSE. Moreover, we see that DiffNEA performs close to ground-truth in performance, and generalizes better than the unstructured MLP as well as the DeLaN model: MLP model oscillates significantly in the end-effector velocity ($\dot{\mathbf{x}}$) tracking, while DeLaN also oscillates mildly in $\dot{\mathbf{x}}$ tracking.

4.1.2. ONLINE LEARNING SPEED

Next, we measure how fast each model can learn in an online learning setup. We train each model sequentially **without shuffling** on the sine motion data, where each batch is of size of 256. As the model trains with the sequential data, we measure its prediction performance through the NMSE on the entire dataset.

In Fig. 1 we see that the DiffNEA model with rotational inertia $\boldsymbol{I}_C$ parameterized with symmetric positive definite (SPD) matrix, triangular parameterization, and covariance parameterization learns the fastest, but also generalizes

Figure 1: Online learning, with NMSE in log scale.



the best to the yet unseen training data, outperforming other models in the online learning setup in simulation.

4.2. Real Robot Experiments

For the real KUKA IIWA robot, we collect sine wave tracking data for about 240 seconds at 250 Hz using the default URDF (Unified Robot Description Format) parameters, and the Pinocchio C++ library (Carpentier et al., 2015–2019) for dynamics and kinematics of the robot. We noticed unmodeled friction dynamics not present in simulation, and added a joint viscous friction/damping model to both DeLaN and DiffNEA models, whose parameters are also learned from data. We use one positive constant per joint damping with parameterization similar to J_1, J_2 in Eq. 11, but with $b = 0$ because each joint damping constant can be 0.

4.2.1. EVALUATION

Table 2: Comparison between models trained to optimize L_{ID} on the sine motion dataset on the real robot, in terms of the number of training epochs required to reach convergence, sine motion joint position ($\mathbf{q}$) tracking, sine motion joint velocity ($\dot{\mathbf{q}}$) tracking, and generalization performance: end-effector position ($\mathbf{x}$) and velocity ($\dot{\mathbf{x}}$) tracking NMSE on an end-effector tracking task (unseen task/situation during training).

Model	Sine Tracking (NMSE)			End-Effector Tracking (NMSE)	
	# Training Epochs	$\mathbf{q}$ Tracking	$\dot{\mathbf{q}}$ Tracking	$\mathbf{x}$ Tracking	$\dot{\mathbf{x}}$ Tracking
Default Model	N/A	0.001	0.009	0.000	0.016
MLP	2	0.000	0.011	0.003	0.513
DeLaN	4	0.001	0.013	Unstable	Unstable
DiffNEA Symm	3	0.001	0.012	0.000	0.013
DiffNEA SPD	3	0.001	0.013	0.000	0.014
DiffNEA Tri	2	0.001	0.013	0.000	0.012
DiffNEA Cov	2	0.001	0.012	0.000	0.015

Both MLP and DeLaN models converge to average training NMSE less than 0.1, while the DiffNEA models converge to average training NMSE 0.35, down from the default model with average NMSE 0.74. However, as can be seen in Table 2, the trained DeLaN model is unstable during the end-effector tracking task, while the trained MLP model has a large end-effector velocity ($\dot{\mathbf{x}}$) tracking NMSE due to oscillations. On the other hand, trained DiffNEA models still perform reasonably, showing its better generalization capability. We attribute the imperfect training of DiffNEA models to unmodelled dynamics of the real system, such as static friction.

5. Conclusion and Future Work

In this paper, we incorporate physical constraints in learned dynamics by adding structure to the learned parameters. This enables us to learn the dynamics of a robot manipulator in a computational graph with automatic differentiation, while keeping the learned dynamics physically plausible. We evaluate our approach on both simulated and real 7 degrees-of-freedom KUKA IIWA arm. Our results show that the resulting dynamics model trains faster, and generalize to new situations better than other state-of-the-art approaches.

We also observe that moving to the real robot creates new sources of discrepancies between the rigid body dynamics and the true dynamics of the system. Factors such as static friction cannot be sufficiently captured by our model, and point to interesting future directions.

References

Chae H. An, Christopher G. Atkeson, and John M. Hollerbach. *Model-based Control of a Robot Manipulator*. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01102-6.

Christopher G. Atkeson, Chae H. An, and John M. Hollerbach. Estimation of inertial parameters of manipulator loads and links. *The International Journal of Robotics Research*, 5(3):101–119, 1986. doi: 10.1177/027836498600500306.

Justin Carpentier, Florian Valenza, Nicolas Mansard, et al. Pinocchio: fast forward and inverse dynamics for poly-articulated systems. https://stack-of-tasks.github.io/pinocchio, 2015–2019.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2018.

Roy Featherstone. *Rigid Body Dynamics Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 0387743146.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *CoRR*, abs/1906.01563, 2019. URL http://arxiv.org/abs/1906.01563.

Jayesh K. Gupta, Kunal Menda, Zachary Manchester, and Mykel J. Kochenderfer. A general framework for structured learning of mechanical systems. *CoRR*, abs/1902.08705, 2019. URL http://arxiv.org/abs/1902.08705.

Jayesh K. Gupta, Kunal Menda, Zachary Manchester, and Mykel J. Kochenderfer. Structured mechanical models for robot learning and control. *CoRR*, abs/2004.10301, 2020. URL http://arxiv.org/abs/2004.10301.

K. Hitzler, F. Meier, S. Schaal, and T. Asfour. Learning and adaptation of inverse dynamics models: A comparison. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 491–498, 2019.

M. Jansen. Learning an accurate neural model of the dynamics of a typical industrial robot. In *International Conference on Artificial Neural Networks*, page 12571260, 1994.

D. Kappler, F. Meier, N. Ratliff, and S. Schaal. A new data source for inverse dynamics learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4723–4730, Sep. 2017. doi: 10.1109/IROS.2017.8206345.

Krzysztof R. Kozlowski. *Modelling and identification in robotics*. Springer Science & Business Media, 2012.

F. D. Ledezma and S. Haddadin. First-order-principles-based constructive network topologies: An application to robot inverse dynamics. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 438–445, Nov 2017. doi: 10.1109/HUMANOIDS.2017.8246910.

J. Y. S. Luh, M. W. Walker, and R. P. C. Paul. On-Line Computational Scheme for Mechanical Manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 102(2):69–76, 06 1980. ISSN 0022-0434. doi: 10.1115/1.3149599. URL https://doi.org/10.1115/1.3149599.

M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *7th International Conference on Learning Representations (ICLR)*, May 2019. URL https://openreview.net/pdf?id=BklHpjCqKm.

M. Mistry, S. Schaal, and K. Yamane. Inertial parameter estimation of floating base humanoid systems using partial force sensing. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pages 492–497, Dec 2009. doi: 10.1109/ICHR.2009.5379531.

Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994. ISBN 0849379814.

Jun Nakanishi, Rick Cory, Michael Mistry, Jan Peters, and Stefan Schaal. Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6): 737–757, 2008. doi: 10.1177/0278364908091463. URL https://doi.org/10.1177/0278364908091463.

D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schölkopf. Learning inverse dynamics: A comparison. In *Advances in Computational Intelligence and Learning: Proceedings of the European Symposium on Artificial Neural Networks*, pages 13–18, Evere, Belgium, April 2008. Max-Planck-Gesellschaft, d-side.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Jean-Jacques E. Slotine and Weiping Li. On the adaptive control of robot manipulators. *The International Journal of Robotics Research*, 6(3):49–59, 1987. doi: 10.1177/027836498700600303. URL https://doi.org/10.1177/027836498700600303.

S. Traversaro, S. Brossette, A. Escande, and F. Nori. Identification of fully physical consistent inertial parameters using optimization on manifolds. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5446–5451, Oct 2016. doi: 10.1109/IROS.2016.7759801.

P. M. Wensing, S. Kim, and J. E. Slotine. Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution. *IEEE Robotics and Automation Letters*, 3(1):60–67, Jan 2018. ISSN 2377-3774. doi: 10.1109/LRA.2017.2729659.