A Real-Robot Dataset for Assessing Transferability of Learned Dynamics Models

Diego Agudelo-España[1], Andrii Zadaianchuk[1], Philippe Wenk[1,2], Aditya Garg[1], Joel Akpo[1],
Felix Grimminger[1], Julian Viereck[1], Maximilien Naveau[1], Ludovic Righetti[3], Georg Martius[1],
Andreas Krause[2], Bernhard Schölkopf[1], Stefan Bauer[1] and Manuel Wüthrich[1]

Abstract— **In the context of model-based reinforcement learning and control, a large number of methods for learning system dynamics have been proposed in recent years. The purpose of these learned models is to synthesize new control policies. An important open question is how robust current dynamics-learning methods are to shifts in the data distribution due to changes in the control policy. We present a real-robot dataset which allows to systematically investigate this question. This dataset contains trajectories of a 3 degrees-of-freedom (DOF) robot being controlled by a diverse set of policies. For comparison, we also provide a simulated version of the dataset. Finally, we benchmark a few widely-used dynamics-learning methods using the proposed dataset. Our results show that the iid test error of a learned model is not necessarily a good indicator of its accuracy under control policies different from the one which generated the training data. This suggests that it may be important to evaluate dynamics-learning methods in terms of their transfer performance, rather than only their iid error.**

I. Introduction

Model-based reinforcement learning and optimal control methods synthesize controllers using dynamics models. There is a broad spectrum of methods for acquiring dynamics models, ranging from physics-based modeling combined with parameter identification to black-box methods based on e.g. neural networks. Naturally, such models would be of little use if they were only valid for the controller under which the training data was collected. Hence, the model learned under a certain source distribution must be valid for a different target distribution. This is an instance of a problem known as transfer learning in the machine learning community [1], we will also refer to it as extrapolation throughout the paper. Theoretical analysis is very difficult in our case for two main reasons: i) It is hard to quantify the shift in the data distribution implied by a shift in the control policy, and ii) many current methods are very complex. Therefore, it is necessary to empirically evaluate algorithms in terms of their extrapolation performance. The main contribution of this article is to propose a dataset for this purpose.

The benefit of benchmarks in the research community has lately been demonstrated in image classification [2], [3] and autonomous driving [4], [5]. In the context of robotic learning, a significant push has been made in this direction through simulation-based benchmarking suits for continuous control [6], [7]. Since it is unclear how well simulation results generalize to real-world scenarios, these simulators must be complemented by real-world datasets. Such datasets have been released for drone racing [8], [9], grasping and manipulation [10], [11] and identifying the dynamics of robotic arms [12], [13], which is the problem we consider here. In contrast to the aforementioned datasets, we design the learning problem such that the main difficulty lies in extrapolation. Specifically,

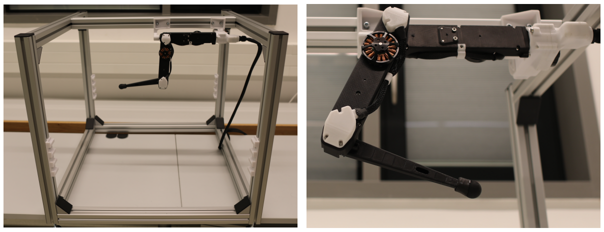

Fig. 1: The data was recorded on a 3 DOF real-robotic system using a number of widely different controllers. Left: view of the entire setup. Right: close-up of the 3 DOF arm.

1) the system (3 DOF arm, see Figure 1) is simple, such that achieving low iid test error is relatively easy,
2) yet transfer is challenging because we record data from a number of widely different controllers.

We provide both real and simulated data (including simulation code) of the same setup[1].

We evaluate a few widely-used dynamics-learning methods on this dataset and release the code of all implementations and evaluations[1]. The main insights we gained can be summarized as follows: The iid test error of a model (i.e. its accuracy under the controller from which the training data was generated) can be a poor indicator of its transfer error (i.e. the accuracy under a different controller). This effect appears to be more pronounced for very flexible models, e.g. neural networks (NNs) and Gaussian processes (GPs), than for more constrained models, e.g. linear. This suggests that it may be important to evaluate dynamics-learning methods in terms of their transfer performance, rather than only their iid error.

[1] Max Planck Institute for Intelligent Systems, Tubingen, Germany. `dagudelo@tue.mpg.de`
[2] ETH Zürich, Zürich, Switzerland.
[3] New York University, New York City, USA.

[1] `https://github.com/rr-learning/transferable_dynamics_dataset`

II. Related Work

Identifying dynamical systems is a fundamental problem encountered in many areas of research and engineering, which is reflected by an accordingly large body of literature. Most of the early work on this problem was published in the control community under the name of system identification. Work on system identification started in the 1960s and was initially mainly concerned with linear systems, see [14], [15], [16] for an overview. Greater interest in nonlinear system identification started in the 1990s (e.g. [17], [18]), with a particular focus on neural networks (e.g. [19]). Nonlinear dynamics-learning methods were also proposed in the robotics community (e.g. locally linear models [20]) and the machine learning community (e.g. using radial basis functions [21]), see [22] for an overview.

Currently, the most widely used families of methods for fitting nonlinear dynamical systems are Gaussian processes (GP) and neural networks (NN). Another promising direction is taken by methods attempting to discover the underlying physical equations of the system. In our experiments we compare methods from each of these three families, and in the following we give a brief overview of the literature.

A. Gaussian Processes

GP-based methods are the most widely-used approaches for Bayesian non-parametric regression [23]. They have been successfully applied to dynamics-learning problems, leading to data-efficient reinforcement learning algorithms such as PILCO [24]. However, the computational complexity (cubic in the number of training points) has hampered application to big-data settings, which are ubiquitous in robotics applications. To address this computational constraint, approaches based on local models [13] and sparse approximations [25], [26] have been proposed. More recently, sparse methods have been extended to incorporate stochastic variational inference, which enables mini-batch learning [27]. This algorithm is called the sparse variational GP (SVGP), and we evaluate it our experiments.

B. Neural Networks

Early work on NN-based dynamics used feedforward NNs to fit discrete-time dynamics [19]. More recently, NN dynamics models using high-dimensional observations, such as images, have been proposed (e.g., [28], [29], [30], [31]). These methods find a low-dimensional latent state representation, typically using variational auto encoders [32]. The dynamics in the latent representation are then usually represented by a feedforward NN. Alternative formulations based on recursive NNs [34] and Bayesian NNs [35] have been recently used to model dynamics as well.

C. Discovering Physical Equations

In [36], the authors propose a method based on evolutionary search for discovering physical equations from data. In contrast, the authors of [37] propose to use sparse regression for identifying the governing equation from a dictionary of possible terms. Yet another approach is taken in [38], [39], where the authors endow a neural network with a set of nonlinearities typically occurring in mechanics. Appropriate regularization encourages solutions with few terms. This method is called the equation learner (EQL), and we evaluate it in the experimental section.

III. Problem Formulation

Our goal here is to find an autoregressive model f which takes as input a history of observations Y and controls U, and predicts some future observation. Formally, we have

$$\hat{Y}_{t+p} = f(Y_{t-h+1:t}, U_{t-h+1:t}, U_{t+1:t+p-1}) \qquad (1)$$

where $p \geq 1$ denotes the prediction horizon and $h \geq 1$ the history length.

The inputs in (1) can be quite high-dimensional, since they consist of time series (history and horizon). This can be problematic for some methods (e.g. GPs). Therefore, we consider a second setting, where we average the inputs over time: Instead of (1) we predict according to $\hat{Y}_{t+p} = f(\bar{Y}_t, \bar{U}_t, \bar{U}_t^{\text{future}})$, with $\bar{Y}_t = \frac{1}{h}\sum_{i=1}^{h} Y_{t-i+1}$, $\bar{U}_t = \frac{1}{h}\sum_{i=1}^{h} U_{t-i+1}$ and $\bar{U}_t^{\text{future}} = \frac{1}{p}\sum_{i=1}^{p} U_{t+i-1}$.

A. Loss function $\mathcal{L}$

Given an observed trajectory $(Y_{1:T}, U_{1:T})$, we compute the loss of a dynamics model f as the absolute prediction error averaged over dimensions and time steps:

$$\mathcal{L}(Y_{1:T}, U_{1:T}; f) = \frac{1}{(T-p-h+1)N_y} \cdot \sum_{t=h}^{T-p} \|Y_{t+p} - f(Y_{t-h+1:t}, U_{t-h+1:t}, U_{t+1:t+p-1})\|_1, \qquad (2)$$

where N_y is the dimension of the observation.

B. Transferability

In model-based reinforcement learning, we typically collect training trajectories $(Y_{1:T}, U_{1:T})$ using some initial policy π (or a sequence of policies). Based on this data, we learn a dynamics model $\hat{f}$. This model is then used to synthesize a new policy π'. However, π' induces a different distribution over $(Y_{1:T}, U_{1:T})$ than π. Hence, it is not clear that the model learned under π will also be accurate under π'. This motivates our experimental setup, in which we learn dynamics models using trajectories sampled from a policy π and test performance (as given by (2)) on trajectories sampled from an unseen policy π'.

IV. Dataset

The proposed dataset contains data recorded from our robotic platform (see Figure 1) using a diverse set of controllers, which allows to test the non-iid generalization case in addition to the iid case. In this dynamical system

- the control and observation rate is 1000 Hz, meaning that the time elapsed between t and $t+1$ is 0.001 seconds,
- we run the system for multiple rollouts, each of which lasts 14 seconds, hence $T = 14000$,

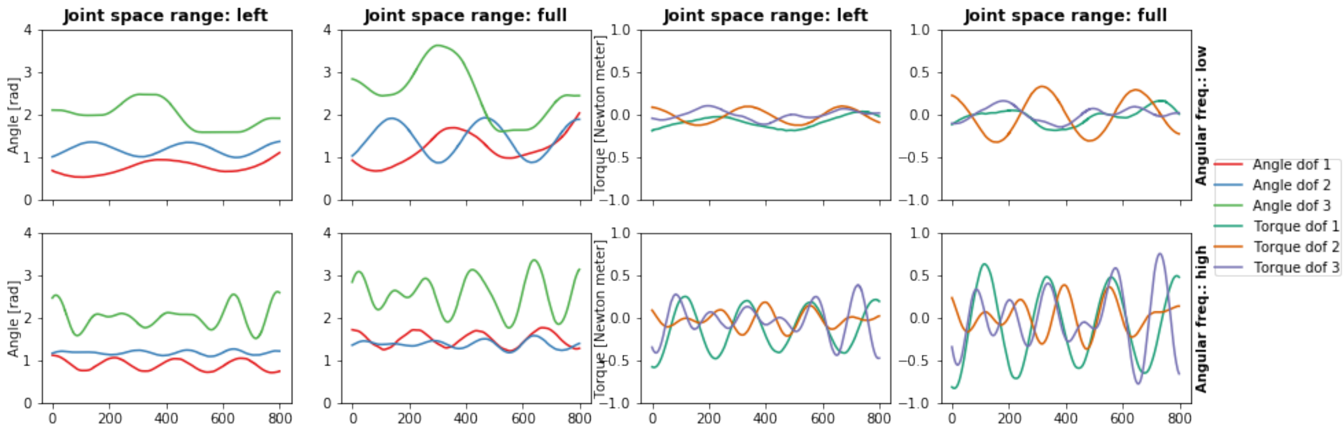

Fig. 2: Recorded angle and desired-torque trajectories for the 3 DOF in the closed-loop dataset (Section IV-B). In the left half angle trajectories for 4 different controllers are shown (each controller uses a particular configuration of angular frequency $\{\text{low}, \text{high}\}$ and reachable joint space $\{\text{left}, \text{full}\}$). The corresponding trajectories of desired torques are shown in the right half of the figure. Note that the inputs (torques) and outputs (angles) look significantly different for different controller settings.

- the control input U_t corresponds to the three ($N_u = 3$) desired torques [in Newton meters] sent to the motors of each joint at time index t,
- the observation Y_t is nine-dimensional ($N_y = 9$) and contains measured angles [in radians], measured velocities [in radians / seconds] and measured torques [in Newton meters] at each joint.

We generate two real-world datasets, one using open-loop controllers and one using closed-loop controllers. In the non-feedback case (Section IV-A) we sample trajectories of desired torques from Gaussian processes (GPs). In contrast, for feedback policies (Section IV-B), we use PD control to track trajectories in joint space which are generated as a random superposition of sine waves.

For each of the two datasets, we sample controllers from four different distributions (e.g. one distribution generating slow controllers, one generating aggressive controllers etc.). This will then allow us to evaluate whether a model learned under one distribution of controllers generalizes to a different controller distribution.

We generated 50 rollouts for each controller distribution, each consisting of a control trajectory U along with an observation trajectory Y, both of length $T = 14000$. These rollouts are grouped into different categories: *training* (38 rollouts), *validation* (3 rollouts) and *testing* (9 rollouts). In Section V we use these datasets to test dynamics learning algorithms in both the iid and non-iid setting.

Moreover, we record simulated versions of the described real-world datasets to investigate to what extent our findings change in simulation scenarios.

A. Open-loop dataset

We use random GP samples directly as torque inputs to explore the platform dynamics in the open-loop setting. More precisely, we have $\left(U_1^i, .., U_T^i\right) \sim \mathcal{N}(\mathbf{0}, K)$ with

$$K_{mn} = s^2 \exp\left(-\frac{(m-n)^2}{2l^2}\right), \tag{3}$$

where l is the trajectory lengthscale and s^2 its output variance. We draw torque trajectories from 4 different distributions, one for each combination of length scales $\mathcal{B} = \{\text{low}, \text{high}\}$ and standard deviations $\mathcal{S} = \{\text{low}, \text{high}\}$. Low length-scales are sampled uniformly from $[16, 64)$ and high ones from $[64, 255)$. Similarly, standard deviations are sampled from either $[0.015, 0.06)$ (low) or $[0.06, 0.24)$ (high). Hence our open-loop dataset D^{open} is a family of smaller datasets indexed by the length scale and the standard deviation regimes $D^{open} = \left(D^{open}_{b,s}\right)_{b \in \mathcal{B}, s \in \mathcal{S}}$.

a) Safe Execution of Controls: Without any safety measure, execution of such torque trajectories would lead the robot to enter into self-collision or to hit its joint limits at high velocities. To avoid this, a safety feature is always running which overrides the desired torques when necessary, such as to maintain the robot always inside a safe zone.

B. Closed-loop dataset

This dataset is recorded while the platform tracks sine trajectories X_t in joint space (refer to Figure 2 for a set of recorded trajectories) which are generated according to

$$X_t = C + A \odot \left(\sum_{i=1}^{k} B_i \odot \sin(\omega_i t + \varphi_i)\right), \tag{4}$$

where $C, A, B_i, \omega_i, \varphi_i$ are all three-dimensional vectors, k denotes the number of sine components that are superimposed (we chose k=3 for the released dataset) and $\odot$ denotes the element-wise product. In order to have a diverse set of trajectories, B_i, ω_i and φ_i are randomly chosen. In particular, the angular frequencies ω_i are sampled from one of two uniform distributions in $\mathcal{F} = \{\text{low}, \text{high}\}$, which correspond to a low $[0, 7.5\pi)$ and a high $[7.5\pi, 15\pi)$ frequency setting. The amplitudes $[B_1, B_2, \ldots, B_k] \in \mathbb{R}^{3\times k}$ are jointly sampled as a stochastic matrix (i.e., each row is a probability mass vector that sums up to one). C and A determine the joint-range reachable by the manipulator, which allows us to consider

$$\begin{tikzcd}
D^{\text{closed}}_{\text{left,low}} \arrow[r, "2"] \arrow[d, "1"'] \arrow[dr, "3"] & D^{\text{closed}}_{\text{full,low}} \\
D^{\text{closed}}_{\text{left,high}} & D^{\text{closed}}_{\text{full,high}}
\end{tikzcd}$$

Fig. 3: Transfer settings: Each arrow goes from the training data to the test data and denotes a particular transfer setting.

two scenarios $\mathcal{R} = \{\text{left}, \text{full}\}$ — *left* means that the joint angles are constrained in such a way that the finger only moves in the left part of the task space, whereas *full* denotes the entire range of safe movements. The closed-loop data set is split accordingly as $D^{closed} = \left(D^{closed}_{r,f}\right)_{r\in\mathcal{R}, f\in\mathcal{F}}$.

C. Simulated datasets

We also record a simulated version of the datasets described in Sections IV-A and IV-B in order to analyze the difference in transfer performance with respect to the real world setting. We use the Articulated-Body algorithm implementation available in the Pinocchio library [40] to forward simulate the manipulator dynamics using its CAD specification. We use the measured torque trajectories in the real world datasets as control inputs for the simulation. In addition, we make the simulation code publicly available.

V. Experiments

We implemented and evaluated a number of dynamics learning algorithms. Each of these algorithms was trained on data recorded using a particular distribution over controllers and then tested on iid data and three different transfer settings in order to evaluate the generalization ability for the iid (on policy) and the transfer (off policy) scenario respectively.

A. Setup

As training set D_{train} we use the *training* set (38 rollouts) of the data set $D^{\text{closed}}_{\text{left,low}}$ (i.e. the data recorded under a closed-loop controller tracking sine waves of low angular frequency, in the left part of the task space, see Section IV-B).

We use the *testing* set (9 rollouts) of $D^{\text{closed}}_{\text{left,low}}$ to evaluate performance under the same controller distribution (iid setting). To evaluate transfer under inputs shift we use the *testing* sets of $D^{\text{closed}}_{\text{left,high}}, D^{\text{closed}}_{\text{full,low}}$ and $D^{\text{closed}}_{\text{full,high}}$ (the transfer settings are depicted graphically in Figure 3).

The *validation* set of $D^{\text{closed}}_{\text{left,low}}$ is used to perform hyperparameter optimization if required by the learning algorithm.

We normalize each input dimension to have zero mean and unit variance, according to common practice.

B. Algorithms

For experimental evaluation, we selected a diverse set of algorithms, each representing a different family of methods. All the algorithms are implemented in the open-source code repository introduced in Section I.

a) Neural Network: We use fully-connected NNs with ReLU activations. For training we use l_2-regularization and Adam as optimizer with batch-size 512. The hyperpararmeters are obtained using grid search with search space: learning rate $\alpha \in [0.01, 0.001, 0.0001]$, regularization coefficient $\lambda \in [0, 0.01, 0.0001]$, number of hidden layers $h \in [2, 3, 4]$ and number of units per hidden layer $m \in [64, 128, 256, 512]$.

b) Gaussian Process: We use the SVGP [27] implementation available in GPflow [42] with a squared exponential kernel and automatic relevance determination (ARD). We assume independent GP models for each output dimension with a shared set of 1000 pseudo-inputs. The kernel hyperparameters are learned by optimizing the evidence lower bound (ELBO) using Adam with batch-size 1000.

c) Discovering Physical Equations: The EQL baseline used in our experiments implements the architecture described in [41]. The learning procedure consists of two phases: the unregularized phase (i.e., 20 epochs of MSE loss minimization) and sparsity phase (i.e., 20 epochs of minimization of the sum of MSE loss and l_0-regularization loss). The hyperparameters are optimized using grid search as follows: learning rate $\alpha \in [0.01, 0.001, 0.0001]$, number of hidden layers $h \in [2, 3]$, number of units groups in hidden layer $m \in [10, 20]$ and for the l_0-regularization coefficient λ we sample 20 values in log-scale from the interval $[10^{-4}, 10^{-1}]$.

d) Linear Model: We also consider a linear baseline. For computational reasons we optimize the parameters using stochastic gradient descent (SGD), instead of solving the resulting least-squares problem in closed form.

e) System Identification: The system identification methods rely on a kinematic model of the robot and estimate the inertial and friction parameters. We considered three variations of this method:

- (cad) we set all the parameters according to the manipulator's CAD model,
- (ls) we optimize the parameters (inertia and friction) to fit the data using least squares,
- (ls-lmi) we use the least squares objective subject to linear matrix inequality constraints, which ensure the physical consistency of the identified parameters [43].

C. Experimental Results

In Figure 4 we report the experimental results. Each of the methods is evaluated for the following settings:

- Prediction horizons $p \in \{1, 10, 100, 1000\}$. System identification is not evaluated for $p = 1000$ given the computational cost of forward integration.
- History lengths $h \in \{1, 10\}$. $h = 10$ is not applicable to system identification, as it takes only one measurement as input.
- With/without averaging, see Section III for details (EQL only with averaging).
- On real data (left column) and simulated data (right column), EQL only on real data.

In the following, we briefly discuss the experimental results.

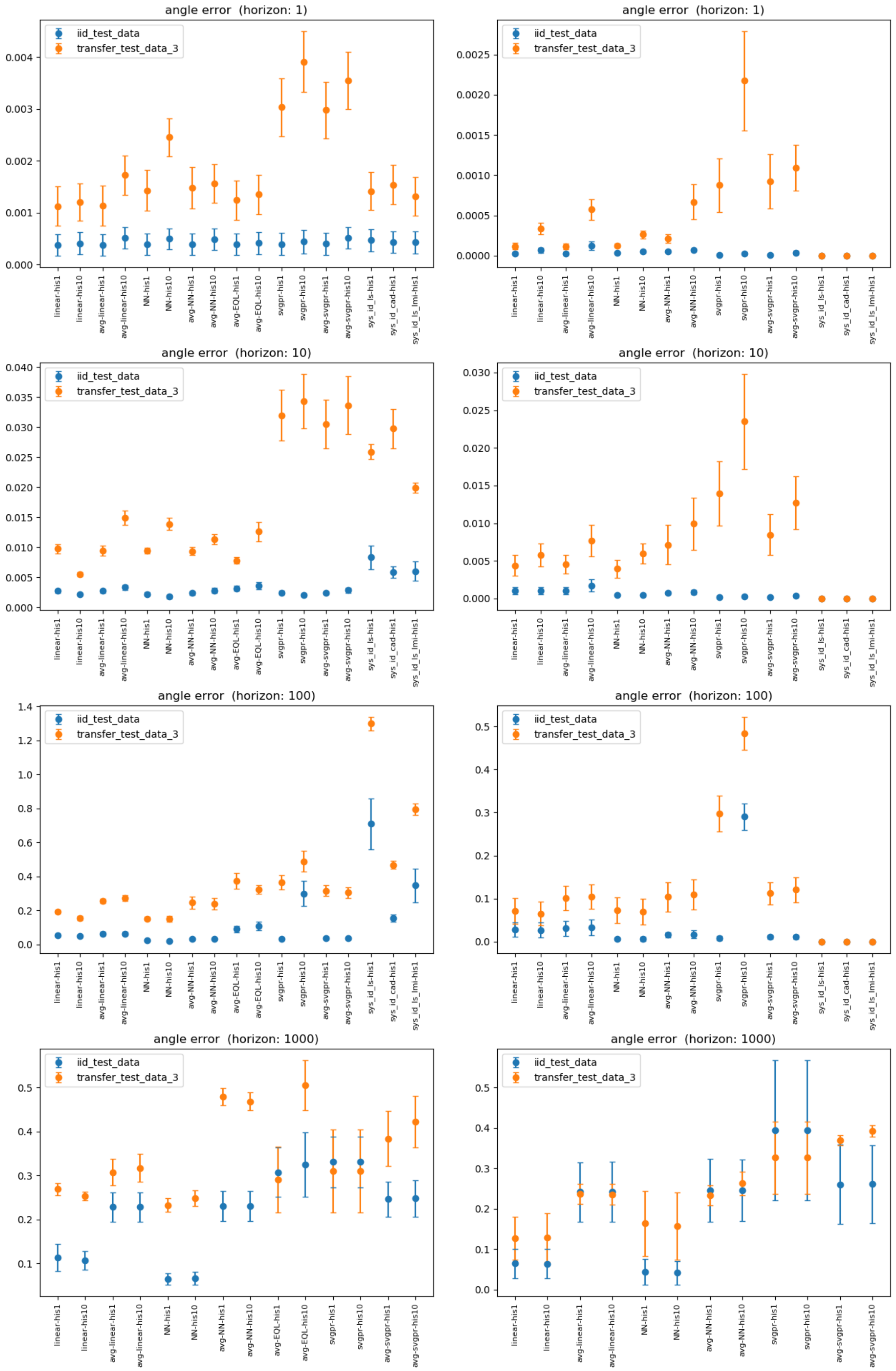

Fig. 4: Plots on the left correspond to real-world settings and their simulation equivalents are shown in the right column. From top to bottom we vary the prediction horizon. The prefix *avg-* indicates that we averaged the inputs, and the suffix (*-his1* or *-his10*) stands for the history length, see Section III for details. We compute an error for each test trajectory according to (2), except that here we only show the error in the angle (i.e. for easier interpretation, we do not include the velocity and torque prediction errors in the average).

a) Real vs Simulated Data: We can see in Figure 4 that the relative performance of the benchmarked methods is similar on simulated and real-world data, except for the system identification approaches. Not surprisingly, these approaches achieve perfect performance in simulation, which indicates that they identify the parameters correctly (up to an equivalence class). However, on the real data, the system identification methods perform very poorly. This performance gap is most likely due to physical effects which are not captured by the rigid-body dynamics model, such as nonlinear actuator dynamics. By including such effects in our model, we could probably achieve much higher accuracy for the system identification methods, but doing so is nontrivial and beyond the scope of this paper.

b) Prediction Horizon: Not surprisingly, with increasing prediction horizon the mean and variance of the errors generally increase. It seems that the system identification methods degrade the most with increasing prediction horizon. This is likely due to the fact that they need to forward integrate, while all other methods directly predict the observation at the desired horizon. The plot for prediction horizon 1000 indicates that averaging the action inputs over time degrades accuracy, in comparison to taking the entire action horizon into account. Hence, methods which can accommodate high-dimensional inputs, such as neural networks and linear models, achieve the best performance.

c) Aggregate Performance of Different Methods: In Figure 5, we aggregate the performance of the different prediction horizons shown in the left column of Figure 4 (and also velocity prediction performance, not shown here). This allows us to obtain an overall ranking of the methods. At the top of Figure 5, we order the methods by their ranking on the iid test set. Among the tested methods, neural networks clearly achieve the best iid test error. At the bottom, we show the same rankings, but now ordered according to the transfer test set. Interestingly, the best performance on the transfer task is achieved by a different method, the linear prediction model with a history of 10. This raises the question whether we can expect models with a good iid test error to generalize well to the transfer distribution.

d) Transfer Error vs iid Error: In Figure 4, we observe that the error on the transfer test data is always substantially higher than the error on the iid test data. This is due to (1) the methods being trained on a different data distribution and (2) the robot moving more aggressively in the transfer data.

However, it is interesting to note that the gap between iid and transfer error varies across methods. In Figure 5, we observe that flexible methods (NNs and GPs) tend to have a better ranking in terms of iid error than in terms of transfer error (i.e. the blue dot is below the orange dot). The converse holds for simple methods (linear) and algorithms which encourage the model to have few terms (EQL). Hence, the ordering of the algorithms with respect to the iid error is significantly different from their ordering with respect to the transfer error (compare top and bottom in Figure 5). This indicates that regularization may be even more important for transfer to different distributions (controllers) than it is for standard iid generalization. However, for the transfer case it is not clear how to perform validation, because at training time we do not know which control policy will be used at test time.

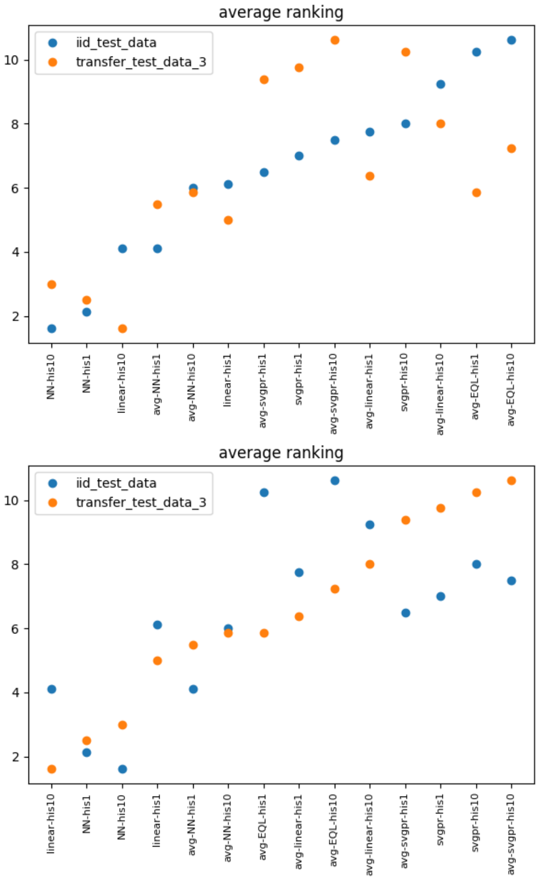

Fig. 5: These two figures show the average ranking of each method (averaged over error type [angle, velocity] and prediction horizons, only real data), lower is better. Both figures show the same information, with the only difference that the upper one is ordered according to the average ranking on the iid test set and the lower one according to the average ranking on the transfer test set.

VI. Discussion

We generated a real-world dataset to systematically evaluate dynamics-learning algorithms in terms of transferability to unseen controllers. As a first step into this direction, we benchmarked a number of methods, and we found that the iid test error is not necessarily a good proxy for the extrapolation error. This indicates that it is important to benchmark dynamics-learning methods in terms of their transfer/extrapolation error, in addition to the iid test error. Therefore, we believe that the proposed dataset is a suitable complement to existing benchmarks.

References

[1] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[3] Y. LeCun, “The mnist database of handwritten digits,” *http://yann. lecun. com/exdb/mnist/*, 1998.

[4] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[5] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, “Bdd100k: A diverse driving video database with scalable annotation tooling,” *arXiv preprint arXiv:1805.04687*, 2018.

[6] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control,” in *International Conference on Machine Learning (ICML)*, 2016, pp. 1329–1338.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” 2016.

[8] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, “Are we ready for autonomous drone racing? the uzhfpv drone racing dataset,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[9] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman, “The blackbird dataset: A large-scale dataset for uav perception in aggressive flight,” *arXiv preprint arXiv:1810.01987*, 2018.

[10] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection,” *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[11] Y. Huang and Y. Sun, “A dataset of daily interactive manipulation,” *The International Journal of Robotics Research*, p. 0278364919849091, 2018.

[12] S. Vijayakumar, A. D’Souza, and S. Schaal, “Incremental online learning in high dimensions,” *Neural computation*, vol. 17, no. 12, pp. 2602–2634, Dec. 2005.

[13] D. Nguyen-tuong, J. R. Peters, and M. Seeger, “Local Gaussian Process Regression for Real Time Online Model Learning,” in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2009, pp. 1193–1200.

[14] M. Gevers, “A personal view of the development of system identification: A 30-year journey through an exciting field,” *IEEE Control systems magazine*, vol. 26, no. 6, pp. 93–105, 2006.

[15] L. Ljung and T. Söderström, *Theory and practice of recursive identification*. MIT press, 1983.

[16] L. Ljung, “Perspectives on system identification,” *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[17] A. Juditsky, H. Hjalmarsson, A. Benveniste, B. Delyon, L. Ljung, J. Sjöberg, and Q. Zhang, “Nonlinear black-box models in system identification: Mathematical foundations,” *Automatica*, vol. 31, no. 12, pp. 1725–1750, 1995.

[18] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky, “Nonlinear black-box modeling in system identification: a unified overview,” *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.

[19] S. Chen, S. A. Billings, and P. M. Grant, “Non-linear system identification using neural networks,” *International journal of control*, vol. 51, no. 6, pp. 1191–1214, 1990.

[20] C. G. Atkeson, A. W. Moore, and S. Schaal, “Locally weighted learning,” in *Lazy learning*. Springer, 1997, pp. 11–73.

[21] Z. Ghahramani and S. T. Roweis, “Learning nonlinear dynamical systems using an em algorithm,” in *Advances in neural information processing systems (NeurIPS)*, 1999, pp. 431–437.

[22] D. Nguyen-Tuong and J. Peters, “Model learning for robot control: a survey,” *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[23] C. E. Rasmussen, “Gaussian processes in machine learning,” in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.

[24] M. Deisenroth and C. E. Rasmussen, “Pilco: A model-based and data-efficient approach to policy search,” in *International Conference on Machine Learning (ICML)*, 2011, pp. 465–472.

[25] J. Quiñonero-Candela and C. E. Rasmussen, “A unifying view of sparse approximate gaussian process regression,” *Journal of Machine Learning Research*, vol. 6, no. Dec, pp. 1939–1959, 2005.

[26] M. Titsias, “Variational learning of inducing variables in sparse gaussian processes,” in *Artificial Intelligence and Statistics*, 2009, pp. 567–574.

[27] J. Hensman, A. G. d. G. Matthews, and Z. Ghahramani, “Scalable variational gaussian process classification,” in *Proceedings of AISTATS*, 2015.

[28] N. Wahlström, T. B. Schön, and M. P. Deisenroth, “Learning deep dynamical models from image pixels,” *IFAC-PapersOnLine*, vol. 48, no. 28, pp. 1059–1064, 2015.

[29] J.-A. M. Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth, “Data-efficient learning of feedback policies from image pixels using deep dynamical models,” *arXiv preprint arXiv:1510.02173*, 2015.

[30] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, “Embed to control: A locally linear latent dynamics model for control from raw images,” in *Advances in neural information processing systems (NeurIPS)*, 2015, pp. 2746–2754.

[31] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, “A disentangled recognition and nonlinear dynamics model for unsupervised learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 3601–3610.

[32] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[33] A. Nagabandi, G. Yang, T. Asmar, G. Kahn, S. Levine, and R. S. Fearing, “Neural network dynamics models for control of under-actuated legged millirobots,” in *Intelligent Robots and Systems (IROS)*, 2018.

[34] D. Ha and J. Schmidhuber, “Recurrent world models facilitate policy evolution,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 2450–2462.

[35] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, “Learning and policy search in stochastic dynamical systems with bayesian neural networks,” *arXiv preprint arXiv:1605.07127*, 2016.

[36] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” *Science*, vol. 324, no. 5923, pp. 81–85, Apr. 2009.

[37] S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Discovering governing equations from data by sparse identification of nonlinear dynamical systems,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 113, no. 15, pp. 3932–3937, Apr. 2016.

[38] G. Martius and C. H. Lampert, “Extrapolation and learning equations,” Oct. 2016.

[39] S. S. Sahoo, C. H. Lampert, and G. Martius, “Learning equations for extrapolation and control,” in *International Conference on Machine Learning (ICML)*, 2018, pp. 4442–4450.

[40] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, “The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives,” in *IEEE International Symposium on System Integrations (SII)*, 2019.

[41] S. S. Sahoo, C. H. Lampert, and G. Martius, “Learning equations for extrapolation and control,” in *Proceedings of the 35th International Conference on machine learning (ICML-18)*, 2018, pp. 4442–4450.

[42] A. G. d. G. Matthews, M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman, “GPflow: A Gaussian process library using TensorFlow,” *Journal of Machine Learning Research*, vol. 18, no. 40, pp. 1–6, apr 2017. [Online]. Available: http://jmlr.org/papers/v18/16-537.html

[43] P. M. Wensing, S. Kim, and J.-J. E. Slotine, “Linear matrix inequalities for physically consistent inertial parameter identification: A statistical perspective on the mass distribution,” *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 60–67, 2017.