Action-Conditional Recurrent Kalman Networks For Forward and Inverse Dynamics Learning

Vaisakh Shaj[1,2], **Philipp Becker**[1], **Dieter Büchler**[3], **Harit Pandya**[2], **Niels van Duijkeren**[4], **C. James Taylor**[5], **Marc Hanheide**[2], and **Gerhard Neumann**[1]

[1]Autonomous Learning Robots, KIT, Germany
[2]LCAS, University Of Lincoln, UK
[3]Max Planck Institute for Intelligent Systems, Tübingen, Germany
[4]Bosch Corporate Research, Renningen, Germany
[5]Lancaster University, UK

Abstract: Estimating accurate forward and inverse dynamics models is a crucial component of model-based control for sophisticated robots such as robots driven by hydraulics, artificial muscles, or robots dealing with different contact situations. Analytic models to such processes are often unavailable or inaccurate due to complex hysteresis effects, unmodelled friction and stiction phenomena, and unknown effects during contact situations. A promising approach is to obtain spatio-temporal models in a data-driven way using recurrent neural networks, as they can overcome those issues. However, such models often do not meet accuracy demands sufficiently, degenerate in performance for the required high sampling frequencies and cannot provide uncertainty estimates.

We adopt a recent probabilistic recurrent neural network architecture, called Recurrent Kalman Networks (RKNs), to model learning by conditioning its transition dynamics on the control actions. RKNs outperform standard recurrent networks such as LSTMs on many state estimation tasks. Inspired by Kalman filters, the RKN provides an elegant way to achieve action conditioning within its recurrent cell by leveraging additive interactions between the current latent state and the action variables. We present two architectures, one for forward model learning and one for inverse model learning. Both architectures significantly outperform existing model learning frameworks as well as analytical models in terms of prediction performance on a variety of real robot dynamics models.

Keywords: Recurrent Networks, Forward Dynamics Learning, Inverse Dynamics Learning, Action-Conditioning, Soft Robots

1 Introduction

Dynamics models are an integral part of many control architectures. Depending on the control approach, the control law relies either on a forward model, mapping from control input to the change of system state, or on an inverse model, mapping from the change of the system state to control signals. However, analytical dynamics models are either not available or often too inaccurate in situations such as robots driven by hydraulics, artificial muscles, or robots dealing with unknown contact situations. Hence, there is a significant need for data-driven approaches that can deal with such complex systems. Yet, modelling dynamics is a challenging task due to the inherent hysteresis effects, unmodelled friction and stiction phenomena, and unknown properties of the interacting objects. Additional challenges for modelling are given by the high data frequency, often up to 1kHz. Furthermore, many modern model-based architectures [1][2] for learning controllers rely on uncertainty estimates of the prediction. Hence, such probabilistic modelling ability is another point on the desiderata for model learning algorithms.

Correspondence to Vaisakh Shaj <v.shaj@kit.edu>

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

In this paper, we extend a recent recurrent neural network architecture [3], called Recurrent Kalman Networks (RKN), for learning forward and inverse dynamics models. The RKN provides probabilistic predictions of future states, can deal with noisy, high dimensional and missing inputs, and has been shown to outperform Long-Short Term-Memory Networks (LSTMs)[4] and Gated Recurrent Units (GRUs) [5] on many state estimation tasks. The authors [3] focussed on estimating the state of the system from images. We adapt the RKN architecture for prediction instead of filtering and employ it for learning robot forward and inverse dynamics. In this scenario, the observations are typically low dimensional, i.e., positions and velocities of the joints of the robot, but occur with a much higher frequency. Unlike original RKN, we explicitly model the control actions in our architecture. We modify the model of the latent transition dynamics using an action dependent non-linear additive factor in order to condition them with action variables. We call this architecture action-conditional RKN (ac-RKN) and show that our approach is more accurate than competing learning methods and even analytical models where they are available. Moreover, we extend the ac-RKN to learning inverse dynamics by adding an action decoder network. Also our learned inverse dynamics models outperform analytical models and competitive learning methods.

2 Related Works

Robot Dynamics Learning. Due to the increasing complexity of robot systems, analytical models are more difficult to obtain. This problem leads to a variety of model estimation techniques which allow the roboticist to acquire models from data. When learning the model, mostly standard regression techniques with Markov assumptions on the states and actions are applied to fit either the forward or inverse dynamics model to the training data. Authors used Linear Regression [6][7], Gaussian Mixture Regression [8][9], Gaussian Process Regression [10][11], Support Vector Regression [12], and feed forward neural networks [13]. However the Markov assumptions are violated in many cases, e.g., for learning the dynamics of complex robots like hydraulically and pneumatically actuated robots or soft robots. Here, the Markov assumptions for states and actions no longer hold due to hysteresis effects and unobserved dependencies. Similarly, learning the dynamics of robots in frictional contacts with unknown objects also violates the Markov assumption and requires reliance on recurrent models which can take into account the temporal dynamic behavior of input sequences while making predictions. Recently, using RNNs such as LSTMs or GRUs has been attempted. Those scale easily to big data, available due to the high data frequencies, and allow learning in $O(n)$ time [14]. However, the lack of a principled probabilistic modelling and action conditioning makes them less reliable to be applied for robotic control applications.

Action-Conditional Probabilistic Models. In model based control and reinforcement learning (RL) problems, learning to predict future states and observations conditioned on actions is a key component. Approaches such as [1], [15], [16], [17], [18] try to achieve this by using traditional models like GPs, feed forward neural networks, or LSTMs. The action conditioning is realized by concatenating the input observations and action signals [15][1] or via factored conditional units [19], where features from some number of inputs modulate multiplicatively and are then weighted to form network outputs. In each of these approaches action conditioning happens outside of the recurrent neural network cell which leads to sub-optimal performance as observations and actions are treated similarly. We propose an action-conditional RNN cell inspired by Kalman filter operations [3] which provide an elegant way to learn not just forward but regularized inverse models as well, giving significantly better performance compared to the current state of the art. Also, opposed to previous approaches for dynamics learning we evaluate not only on relatively simple electrically actuated robotic systems but on a wide variety of complex robotic systems, including large hydraulically actuated robots and pneumatically actuated soft robots for one-step-ahead and long term prediction tasks.

3 Recurrent Kalman Networks

Recurrent Kalman Networks (RKN) [3] integrate uncertainty estimates into deep time-series modelling by combining Kalman Filters [20] with deep neural networks. To efficiently address highly nonlinear dynamics the RKN employs a Kalman filter in a high dimensional, factorized latent space where the dynamics can be modelled using locally linear models. This space is learned by an encoder-decoder structure mapping from and back to the latent representation. Both encoder and

decoder is in practice realized as deep neural networks. However, the encoder does not only map the observation to the latent space but also emits an estimate of the uncertainty in the observation. This uncertainty estimate is used, together with latent state uncertainties inherently present in the Kalman filter, to control the update of the state/memory of the recurrent cell. This update procedure for the memory is in contrast to traditional deep recurrent approaches, such as LSTMs [4], where input and forget gates have a similar purpose but do not offer a probabilistic interpretation.

Formally, the RKN first uses a locally linear dynamics model, $\boldsymbol{A}_{t-1}$, to advance the last posterior belief $\mathcal{N}(\boldsymbol{z}_{t-1}^+, \boldsymbol{\Sigma}_{t-1}^+)$ in time and obtain a prior belief $\mathcal{N}(\boldsymbol{z}_t^-, \boldsymbol{\Sigma}_t^-)$ for the current time step t. Here, $\boldsymbol{A}_{t-1}$ depends on the posteriors mean, $\boldsymbol{z}_{t-1}^+$, i.e, $\boldsymbol{A}_{t-1} := \boldsymbol{A}(\boldsymbol{z}_{t-1}^+)$, and the transition noise is modelled by $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I} \cdot \boldsymbol{\sigma}^{\text{trans}})$, with a learned, vector value $\boldsymbol{\sigma}^{\text{trans}}$. This prior belief is then updated using a probabilistic representation of the observation $\mathcal{N}(\boldsymbol{w}_t, \boldsymbol{\sigma}_t^{\text{obs}})$, provided by the encoder. Classical Kalman update equations then infer the Kalman gain $\boldsymbol{Q}_t$ and, subsequently, the current posterior $\mathcal{N}(\boldsymbol{z}_t^+, \boldsymbol{\Sigma}_t^+)$. Ultimately, the decoder maps from the posterior to the desired output, where we can learn separate decoders for the mean and the variance estimates. The network is trained end to end via backpropagation through time.

A detailed description of the RKN, the exact factorization assumptions and parameterization of the locally linear transition model is given in [3] and is also summarized in the appendix.

4 Robot Dynamics Learning with Action-Conditional RKNs

In this paper, we are interested in learning accurate forward or inverse dynamics models of complex robots with non-Markovian dynamics. Learning such models requires action-dependent prediction models of high accuracy in order to be useful for robot control. Moreover, we deal with systems with high sample frequencies, up to 1kHz. We extend the RKN approach to robot dynamics learning with the following innovations: **(i)** We use an **action-conditional prediction** update which provides a natural way to incorporate control inputs into the latent dynamical system. **(ii)** We modify the architecture to perform prediction instead of filtering and learn **accurate forward dynamics models** for multiple robots with complex actuator dynamics. **(iii)** We incorporate an inverse dynamics decoder in the RKN architecture, allowing us to learn jointly forward and **inverse dynamics models**. We refer to the resulting approach as action-conditional RKN (ac-RKN).

4.1 Action Conditioning

To achieve action conditioning within the recurrent cell, we modify the transition model proposed in [3] to include a control model $\boldsymbol{b}(\boldsymbol{a}_t)$ in addition to the locally linear transition model $\boldsymbol{A}_t$. The control model $\boldsymbol{b}(\boldsymbol{a}_t)$ is added to the locally linear state transition, i.e., $\boldsymbol{z}_{t+1}^- = \boldsymbol{A}_t \boldsymbol{z}_t^+ + \boldsymbol{b}(\boldsymbol{a}_t)$. As illustrated in Figure 1, this formulation of latent dynamics captures the causal effect of the actions variables $\boldsymbol{a}_t$ on the state transitions in a more principled manner than including the action as an observation, which is the common approach for action conditioning in RNNs [3]. The control model $\boldsymbol{b}(\boldsymbol{a}_t)$ can be represented in several ways, i.e.:

- **(i) Linear:** $\boldsymbol{b}_l(\boldsymbol{a}_t) = \boldsymbol{B}\boldsymbol{a}_t$, where $\boldsymbol{B}$ is a linear transformation matrix.
- **(ii) Locally-Linear:** $\boldsymbol{b}_m(\boldsymbol{a}_t) = \boldsymbol{B}_t\boldsymbol{a}_t$, where $\boldsymbol{B}_t = \sum_{k=0}^{K} \beta^{(k)}(\boldsymbol{z_t})\boldsymbol{B}^{(k)}$ is a linear combination of k linear control models $\boldsymbol{B}^{(k)}$. A small neural network with softmax output is used to learn $\beta^{(k)}$.
- **(iii) Non-Linear:** $\boldsymbol{b}_n(\boldsymbol{a}_t) = \boldsymbol{f}(\boldsymbol{a}_t)$, where $\boldsymbol{f}(.)$ can be any non-linear function approximator. We use a multi-layer neural network regressor with ReLU activations.

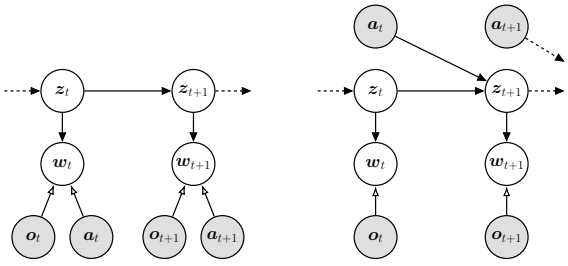


Figure 1: Graphical models for actions treated as fake observations as in [3] (left) and ac-RKN (right) which treats actions in a principled manner by capturing its causal effect on the state transition via additive interactions with the latent state. The hollow arrows denote deterministic transformation leading to implicit distributions.

Note that the prediction step for the variance is not affected by this choice of action conditioning, i.e., $\boldsymbol{\Sigma}_{t+1}^{-} = \boldsymbol{A}_t \boldsymbol{\Sigma}_t^{+} \boldsymbol{A}_t^T + \boldsymbol{I} \cdot \boldsymbol{\sigma}^{\mathbf{trans}}$ as the action is known and not uncertain. Thus, unlike the state transition model $\boldsymbol{A}_t$, we do not need to constrain the model $\boldsymbol{b}$ to be linear or locally-linear as it neither affects the Kalman gain nor how the covariances are updated. Hence, we use the non-linear approach, $\boldsymbol{b}_n(\boldsymbol{a}_t)$, as it provides the most flexibility and achieved the best performance as shown in section 5.1. The Kalman update step is also unaffected by the new action-conditioned prediction step and therefore remains as presented in [3].

Our principled treatment of control signals is crucial for learning accurate forward dynamics models, as seen in section 4.2. Moreover, the disentangled representation of actions in the latent space gives us more flexibility in manipulating the control actions for different applications including inverse dynamics learning (section 4.3), extensions to model-based reinforcement learning and planning. For long-term prediction using different action sequences, we can still apply the latent transition dynamics without using observations for future time steps.

4.2 Forward Dynamics Learning

We want to learn a forward dynamics model $f : \boldsymbol{o}_{1:t}, \boldsymbol{a}_{1:t} \mapsto \boldsymbol{o}_{t+1}$ that predicts the next observation $\boldsymbol{o}_{t+1}$ given histories of observation $\boldsymbol{o}_{1:t}$ and actions $\boldsymbol{a}_{1:t}$. We explicitly focus on non-Markovian systems, where the observations are typically joint angles and, if available, joint velocities of all degrees of freedom of the robot. In our experiments, we aim to predict the joint angles at the next time step given a sequence of joint angles and control actions. Due to unmodelled effects such as hysteresis or unknown contacts, the state transitions are non-Markovian even if joint angles and velocities are known. We can assume that the observations are almost noise-free since the measurement errors for our observations (joint angles) are minimal. Nevertheless, as the observations do not contain the full state information of the system, we still have to model uncertainty in our latent state using the RKN.

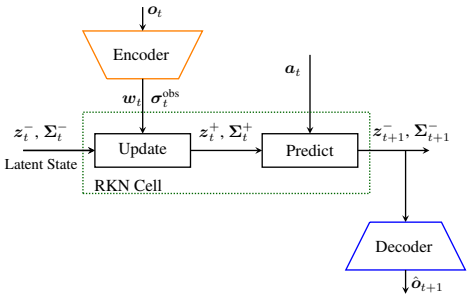


Figure 2: Schematic diagram of ac-RKN for forward dynamics learning. Action conditioning is implemented by adding a latent control vector $\boldsymbol{b}(\boldsymbol{a}_t)$ to the dynamics model of the RKN. The output of the predict stage, which forms the prior for the next time step $\left(\boldsymbol{z}_{t+1}^-, \boldsymbol{\Sigma}_{t+1}^-\right)$ is decoded to get the prediction of the next observation.

The function f can be challenging to learn when the observations, i.e., joint positions and velocities, of two subsequent time-steps are too similar. In this case, the action has seemingly little effect on the output and the learned strategy is often to copy the previous state for the next state prediction. This difficulty becomes more pronounced as the time step between states becomes smaller ,e.g., 1ms, as minor errors in absolute states estimates can already create unrealistic dynamics. Hence, a standard method for model learning is to predict the normalized differences between subsequent observations or states instead of the absolute values [1], i.e., during training, the predicted next observation is $\hat{\boldsymbol{o}}_{t+1} = \boldsymbol{o}_t + \text{dec}\left(\boldsymbol{z}_{t+1}^-\right)$, where $\boldsymbol{o}_t$ is the true observation at t and $\text{dec}\left(\boldsymbol{z}_{t+1}^-\right)$ is the output of the actual decoder network. During inference, we introduce an additional memory $\boldsymbol{m}_t$ which stores the last prediction and uses it as observation in case of a missing observation, i.e., $\hat{\boldsymbol{o}}_{t+1} = \boldsymbol{m}_t + \text{dec}\left(\boldsymbol{z}_{t+1}^-\right)$ where we use the true observation, $\boldsymbol{m}_t = \boldsymbol{o}_t$ if available and the predicted observation, $\boldsymbol{m}_t = \hat{\boldsymbol{o}}_t$, otherwise. The architecture of the ac-RKN is summarized in Figure 2.

The RKN cell is well equipped to handle missing observations, as the update step can just be omitted in this case, i.e., the posterior is equal to the prior if no observation is available. Thus, we can also use the presented architecture to predict several time steps into the future. In this case, we repeatedly apply the RKN prediction step and update the memory $\boldsymbol{m}_t$ with the corresponding prediction while omitting the Kalman update.

Loss And Training. For training forward dynamic models, we optimize the root mean square error (RMSE) between the decoded output and normalized differences to the the next true state

$(\boldsymbol{o}_{t+1} - \boldsymbol{o}_t)$ as in [15],

$$\mathcal{L}_{\text{fwd}} = \sqrt{\frac{1}{T}\sum_{t=1}^{T} \parallel (\boldsymbol{o}_{t+1} - \boldsymbol{o}_t) - \text{dec}\left(\boldsymbol{z}_{t+1}^{-}\right) \parallel^2}.$$

Note that we decode the prior mean $\boldsymbol{z}_{t+1}^{-}$ for predicting $\boldsymbol{o}_{t+1}$. The prior mean $\boldsymbol{z}_{t+1}^{-}$ integrates all information up to time step t, including $\boldsymbol{a}_t$, but already denotes the belief for the next time step. Hence, as we are interested in prediction, we work with this prior. In contrast, [3] used the posterior belief since their goal was filtering rather than prediction.

Further, in [3], the Gaussian log-likelihood is used instead of the RMSE. While this is in principle also possible here, it is only necessary if uncertainty estimates in the outputs are required. If this is not the case, training on RMSE yields slightly better predictions and allows for a fair comparison with deterministic baselines like feed-forward neural networks, LSTMs and analytical models. Thus, in this work, we chose to work with the RMSE loss.

4.3 Inverse Dynamics Learning

For the inverse dynamics case we want to learn a model $f^{-1} : \boldsymbol{o}_{1:t}, \boldsymbol{a}_{1:t-1}, \boldsymbol{o}_{t+1} \mapsto \hat{\boldsymbol{a}}_t$ where $\boldsymbol{o}_{t+1}$ is the desired next observation for time step $t+1$ and $\hat{\boldsymbol{a}}_t$ is the predicted action, i.e., the one to be applied. We introduce an action decoder which decodes the latent posterior $\left(\boldsymbol{z}_t^{+}, \boldsymbol{\Sigma}_t^{+}\right)$ and estimates the action required to move to the desired next observation. The action decoder also gets information regarding the next observation as an input. During training this corresponds to the next observation in the data. For control a desired next observation is used to obtain the action required to reach that observation. The joint angles and velocities are treated as observations in our inverse dynamics experiments.

As seen in Figure 3, instead of merely learning the inverse dynamics, our approach learns inverse and forward dynamics simultaneously by feeding back the executed action to the action-conditional predict stage, which predicts forward in latent space. Note that the action decoder also gets the same action as the target during training. However, the model sees the true action only after the prediction since the action decoder works with the posterior estimate $\left(\boldsymbol{z}_t^{+}, \boldsymbol{\Sigma}_t^{+}\right)$. Hence, the prediction of the inverse dynamics model for the current time step is made independent of this feedback. We found that enforcing this causal feedback, a necessary structural component of the Bayesian network of the underlying latent dynamical system, improves the performance of the inverse model, as seen in Figure 6c.

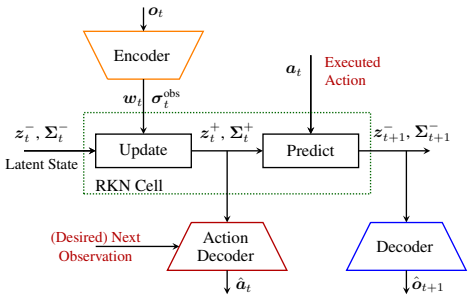
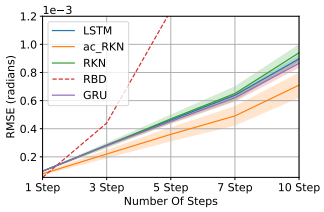


Figure 3: Schematic diagram of the inverse dynamics learning architecture. Here the posterior $\left(\boldsymbol{z}_t^{+}, \boldsymbol{\Sigma}_t^{+}\right)$, at the current time step is fed to an action decoder along with the desired target. In the next time-step, the executed action $\boldsymbol{a}_t$ is fed to the predict stage of RKN cell. Further, the next predicted prior $\left(\boldsymbol{z}_{t+1}^{-}, \boldsymbol{\Sigma}_{t+1}^{-}\right)$ is decoded to get the next state $\hat{\boldsymbol{o}}_{t+1}$.

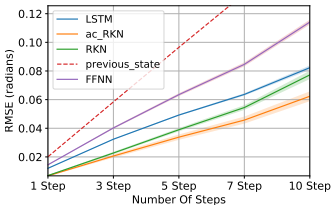
Loss And Training. The dual output architecture for our inverse model leads to two different loss functions for the action and observation decoders which are optimized jointly. Our experiments showed that the loss of the forward model is an excellent auxiliary loss function for the inverse model and learning this implicit forward model jointly with the inverse model results in much better performance for the inverse model. We assume the reasons for this effect is that the forward model loss is providing crucial gradient information to form an informative latent state representation that is also useful for the inverse model. In order to deal with high frequency data we again chose to predict the normalized differences to the previously executed action, i.e. $\hat{\boldsymbol{a}}_t = \boldsymbol{a}_{t-1} + \text{dec}_{\text{action}}(\boldsymbol{z}_t^{+})$. Here $\boldsymbol{a}_{t-1}$ is the executed action at $t-1$ and $\text{dec}_{\text{action}}(\boldsymbol{z}_t^{+})$ the output of the actual action decoder network. The combined loss function for the inverse dynamic case is given by

	RMSE	NLL
ac-RKN	**0.0144**	**6.247**
RKN	0.0282	4.930
LSTM	0.2067	0.952
GRU	0.2015	1.186

(a) Hydraulic Brokk 40



(b) Franka Emika Panda



(c) Pneumatic Arm

Figure 4: **(a)** Performance comparison of various recurrent models for the BROKK-40 hydraulically actuated robot arm. **(b)** and **(c)** Comparison of multi-step ahead prediction for different algorithms for Franka Emika and the pneumatically actuated muscular robot arm. The shaded region shows standard deviations. The dashed line in (b) denotes the prediction performance of the rigid body dynamics model and in (c) it denotes the prediction performance if we just predict the previous state. The results show that all learning approaches outperform the rigid body dynamics model or the previous state prediction, while our ac-RKN architecture outperformed other learning methods.

$$\mathcal{L}_{\text{inv}} = \sqrt{\frac{1}{T}\sum_{i=1}^{T} \parallel (\boldsymbol{a}_{t+1} - \boldsymbol{a}_t) - \text{dec}_{\text{action}}\left(\boldsymbol{z}_t^+\right) \parallel^2 + \lambda \mathcal{L}_{\text{fwd}}}$$

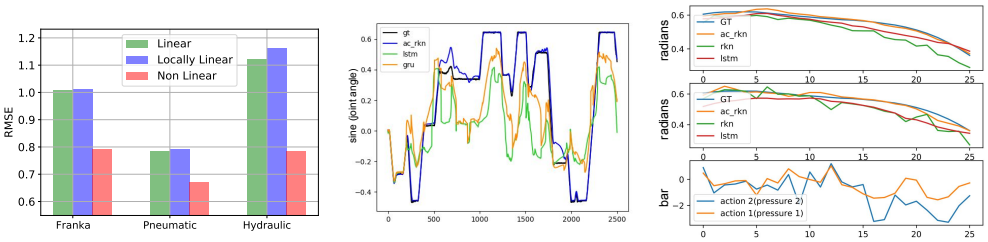
where λ chooses the trade-off between the inverse model loss and the forward model loss. The value of λ is chosen via hyperparameter optimization using GPyOpt [21]. Note that for the inverse model, the actions are decoded based on the posterior mean of the current time step while for the forward model the observations are decoded from the prior mean of the next time step.

5 Experimental Results

In this section, we discuss the experimental evaluation of ac-RKN on learning the forward and inverse dynamics models on robots with different actuator dynamics. A full listing of hyperparameters can be found in the supplementary material. We compare ac-RKN to both standard deep recurrent neural network baselines (LSTMs, GRUs and standard RKN), analytical baselines based on classical rigid body dynamics and non-recurrent baselines like FFNN. For RKN and LSTM baselines, we replaced the ac-RKN transition layer with generic LSTM and RKN layers. For recurrent models, the recurrent cell parameters are tuned via hyperparameter optimization using GPyOpt [21], but the encoder and decoder sizes are similar. The observations and actions are concatenated as in [15] and [1] for all baseline experiments during the forward dynamics learning, i.e., we treat actions as extended observations. The data, i.e., the states or observations, actions and targets are normalized (zero mean and unit variance) before training. We denormalize the predicted values during inference and evaluate their performance on the test set. We evaluate the model using RMSE to ensure a fair comparison with the deterministic models such as FFNNs and LSTMs.

5.1 Forward Dynamics Learning

We evaluate the ac-RKN on three robotic systems with different actuator dynamics each of which poses unique learning challenges: **(i) Hydraulically Actuated BROKK-40 Robot Arm.** The data consists of measured joint positions and the input current to the controller sampled at 100Hz from a hydraulic BROKK-40 demolition robot [22]. The position angle sensors are rotary linear potentiometers. We chose a similar experimental setup and metrics as in [3] to ensure a fair comparison. We trained the model to predict the joint position 2 seconds, i.e. 200 time-steps, into the future, given only control inputs. Afterwards, the model receives the next observation and the prediction process repeated. Learning the forward model here is difficult due to inherent hysteresis associated with hydraulic control. **(ii) Pneumatically Actuated Musculoskeletal Robot Arm.** This four DoF robotic arm is actuated by Pneumatic Artificial Muscles (PAMs) [23]. Each DoF is actuated by an antagonistic pair of PAMs, yielding a total of eight actuators. The robot arm reaches high joint angle



(a) Impact of control input models (b) Hydraulic Arm Predictions (c) Pneumatic Arm Predictions

Figure 5: **(a)** Comparison of different action-conditioning models discussed in 4.1. **(b)** Visualization of the predicted trajectories of the hydraulic arm for 200 step ahead predictions. **(c)** Visualization of the predicted trajectories of the pneumatic arm for 5 (top) and 10 (bottom) step ahead predictions for one of the joints. (b) and (c) clearly show that the predictions given by our approach are by far the most accurate.

accelerations of up to 28,000 deg$/s^2$ while avoiding dangerous joint limits thanks to the antagonistic actuation and limits on the air pressure ranges. The data consists of trajectories of hitting movements with varying speeds while playing table tennis [24] and is recorded at 100Hz. The fast motions with high accelerations of this robot are complicated to model due to hysteresis. **(iii) Franka Emika Panda Arm.** We collected the data from a 7 DoF Franka Emika Panda manipulator during free motion at a sampling frequency of 1kHz. It involved a mix of movements of different velocities from slow to swift motions. The high frequency, together with the abruptly changing movements results in complex dynamics which are interesting to analyze.

Multi Step Ahead Prediction. Figure 4 summarizes the test set performance for each of these robots. We benchmarked the performance of ac-RKN with state of the art deterministic deep recurrent models (LSTM and GRU) and non-recurrent models like feed-forward neural network (FFNN). In all three robots, the ac-RKN gave much better multi-step ahead prediction performance than these deterministic deep models. The performance improvement was more significant for robots with hydraulic and pneumatic actuator dynamics due to explicit non-markovian dynamics and hysteresis, which is often difficult to model via analytical models. We also validated the performance improvement due to our principled action-conditioning in the latent transition dynamics by comparing it with RKN [3], which treats actions as part of the observations by 'concatenation'. In all three robots, our principled treatment brought significant improvement in performance. Figure 5 (b,c) shows the predicted trajectories for the BROKK and the pneumatically actuated robot arm.

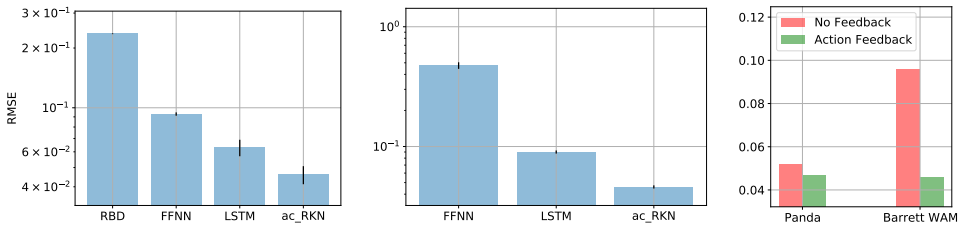
Comparison with Analytical Model. We were also interested in making a comparison with analytical models of Franka. For the analytical model, aside from the inertia properties of the links, the Coulomb friction was also identified for every joint. A detailed description for the same can be found in Appendix C. As seen in Figure 4 (b), the performance of the analytical model outperformed ac-RKN for one step ahead predictions, but for multi-step forward predictions, the data-driven models had a clear advantage with ac-RKN giving the most accurate results.

Ablation Study for Action-Conditioning. We evaluated different models for action conditioning as discussed in Section 4.2. The resulting evaluation can be seen in Figure 5(a) and shows the advantage of using non-linear models for the additive action-conditioning of the latent dynamics.

5.2 Inverse Dynamics Learning

We evaluated the performance of the proposed method for inverse dynamics learning on two real robots, Franka Emika Panda and Barrett WAM. Barret WAM is a robot with direct cable drives. The direct cable drives produce high torques, generating fast and dexterous movements but yield complex dynamics. Rigid-body dynamics cannot model this complex dynamics accurately due to the variable stiffness and lengths of the cables [25].

Joint Torque Prediction Task. We benchmark the representational capability of the latent state posterior, $\boldsymbol{z}_t^+$, of ac-RKN in accurately modelling the inverse dynamics of these robots in comparison to deterministic models like LSTMs and FFNN. It is clear from 6a and 6b that ac-RKN learns highly accurate models of these in contrast with other data-driven methods. This highly precise modelling is often a requirement for high fidelity and compliant robotic control.



(a) Franka Panda (b) Barret WAM (c) Action Feedback

Figure 6: **(a)** and **(b)** Joint torque prediction RMSE values in NM of Action-Conditional RKN, LSTM and FFNN for Panda and Barret WAM. A comparison is also provided with the analytical (RBD) model of Panda. **(c)** Comparison of ac-RKN for inverse dynamics learning with and without the action feedback as discussed in Section 4.3.

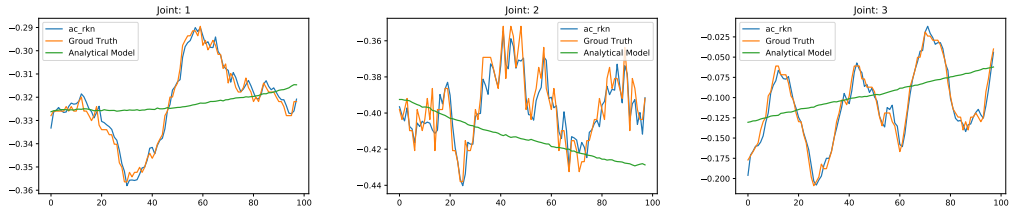


Figure 7: Predicted joint torques(normalized) for first 3 joints of the Panda robot arm. The learned inverse dynamics model by ac-RKN match closely with the ground truth data while the rigid body dynamics model can not capture the high-frequency variations in the data.

Impact of Action Feedback. We also perform an ablation study with and without the action feedback for the prediction step in the latent dynamics. As seen in the Figure 6c, the action feedback always results in better representational capability as this helps the implicit forward model in making better predictions by taking into account its causal effect on the state transitions.

Comparison to Analytical Models. Finally, we make a comparison with the analytical model of the Panda robot for inverse dynamics. Please refer to Appendix C for more details on the analytical model. As evident from Figure 7 analytical models gave predictions with much lesser accuracy in comparison to ac-RKN, as it does not consider unmodelled effects such as joint and link flexibilities, backlash, stiction and actuator dynamics. In such cases, the robot would continuously have to track its current position and compensate the errors with high-gain feedback control thus making it dangerous to interact with the real world and impossible to work in human-centred environments.

6 CONCLUSION

We introduced a probabilistic recurrent neural network architecture, called action-conditional Recurrent Kalman Networks (ac-RKN), capable of action conditioning within the recurrent cell in a principled manner. This formulation allows us to learn highly accurate forward dynamics models of sophisticated robots and scenarios for which analytical models do not exist. We demonstrated the effectiveness of the proposed approach on robots with hydraulic, pneumatic and electric actuators. Besides, we leveraged the action-conditional recurrent cell to propose a regularized inverse dynamics model which significantly outperformed the current state of the art on two benchmarks. We believe the disentangled representation of the control signal in the ac-RKN recurrent cell has the potential to be exploited for several other robot learning problems. Also, since our architecture is domain-independent, we expect that they will generalize to several other robot dynamics learning tasks. As future work, we will employ these models for real-time control on some of the robotic systems, and we will learn models that predict future reward in addition to future states or observations and evaluate the performance of our approach in model-based reinforcement learning. Additionally, we will explore the incremental learning of Kalman Filter parameters to adapt to changing workspace configurations and tasks.

Acknowledgments

This work was supported by the EPSRC UK (project NCNR, National Centre for Nuclear Robotics, EP/R02572X/1). We thank Aravinda Ramakrishnan Srinivasan for his support with the robots.

References

[1] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[2] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[3] P. Becker, H. Pandya, G. Gebhardt, C. Zhao, C. J. Taylor, and G. Neumann. Recurrent kalman networks: Factorized inference in high-dimensional deep feature spaces. In *International Conference on Machine Learning*, pages 544–552, 2019.

[4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, Nov. 1997. ISSN 0899-7667.

[5] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics, Oct. 2014.

[6] S. Schaal, C. G. Atkeson, and S. Vijayakumar. Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17(1):49–60, 2002.

[7] M. Haruno, D. M. Wolpert, and M. Kawato. Mosaic model for sensorimotor learning and control. *Neural computation*, 13(10):2201–2220, 2001.

[8] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard. Learning and reproduction of gestures by imitation. *IEEE Robotics & Automation Magazine*, 17(2):44–54, 2010.

[9] S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[10] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model learning with local gaussian process regression. *Advanced Robotics*, 23(15):2015–2034, 2009.

[11] D. Nguyen-Tuong and J. Peters. Using model knowledge for learning inverse dynamics. In *2010 IEEE International Conference on Robotics and Automation*, pages 2677–2682. IEEE, 2010.

[12] J. P. Ferreira, M. Crisostomo, A. P. Coimbra, and B. Ribeiro. Simulation control of a biped robot with support vector regression. In *2007 IEEE International Symposium on Intelligent Signal Processing*, pages 1–6. IEEE, 2007.

[13] A. S. Polydoros, L. Nalpantidis, and V. Krüger. Real-time deep learning of robotic manipulator inverse dynamics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3442–3448. IEEE.

[14] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters. Learning inverse dynamics models in o (n) time with lstm networks. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 811–816. IEEE, 2017.

[15] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[16] I. Lenz, R. A. Knepper, and A. Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy, 2015.

[17] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

[18] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

[19] G. W. Taylor and G. E. Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *Proceedings of the 26th annual international conference on machine learning*, pages 1025–1032. ACM, 2009.

[20] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[21] T. G. authors. GPyOpt: A bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016.

[22] C. J. Taylor and D. Robertson. State-dependent control of a hydraulically actuated nuclear decommissioning robot. *Control Engineering Practice*, 21(12):1716–1725, 2013.

[23] D. Büchler, H. Ott, and J. Peters. A lightweight robotic arm with pneumatic muscles for robot learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4086–4092. IEEE, 2016.

[24] D. Büchler, S. Guist, R. Calandra, V. Berenz, B. Schölkopf, and J. Peters. Learning to play table tennis from scratch using muscular robots. *arXiv preprint arXiv:2006.05935*, 2020.

[25] D. Nguyen-Tuong and J. Peters. Incremental online sparsification for model learning in real-time robot control. *Neurocomputing*, 74(11):1859–1867, 2011.

[26] W. Khalil, M. Gautier, and C. Enguehard. Identifiable parameters and optimum configurations for robots calibration. *Robotica*, 9(1), 1991. doi:10.1017/S0263574700015575.

[27] C. D. Sousa and R. CortesÃ£o. Inertia tensor properties in robot dynamics identification: A linear matrix inequality approach. *IEEE/ASME Transactions on Mechatronics*, 24(1):406–411, 2019.

Appendix

Vaisakh Shaj[1,2], **Philipp Becker**[1], **Dieter Büchler**[3], **Harit Pandya**[2], **Niels van Duijkeren**[4], **C. James Taylor**[5], **Marc Hanheide**[2], and **Gerhard Neumann**[1]

[1]Autonomous Learning Robots, KIT, Germany
[2]LCAS, University Of Lincoln, UK
[3]Max Planck Institute for Intelligent Systems, Tübingen, Germany
[4]Bosch Corporate Research, Renningen, Germany
[5]Lancaster University, UK

Appendix A: RKN - Summary and Conceptual Components

The Recurrent Kalman Network (RKN) [3] integrates uncertainty estimates into deep time-series modelling by incorporating Kalman Filters [20] into deep recurrent models. While Kalman filtering in the original state space requires approximations due to the non-linear models, the RKN uses a learned high-dimensional latent state representation that allows for efficient inference using locally linear transition models and a factorized belief state representation. The RKN consists of the following conceptual components.

Observation and Latent State Representation

The RKN transforms the observations at each time step $\boldsymbol{o}_t$ to a high dimensional space using an encoder network which emits high dimensional latent features $\boldsymbol{w}_t$ and an estimate of the uncertainty in those features via a variance vector $\boldsymbol{\sigma}_t^{\text{obs}}$.

The probabilistic recurrent module uses a latent state vector $\boldsymbol{z}_t$ and corresponding covariance $\boldsymbol{\Sigma}_t$ whose transitions are governed by the Kalman Filter in the RKN memory cell. The latent state vector $\boldsymbol{z}_t$ has been designed to contain two conceptual parts, a vector $\boldsymbol{p}_t$ for holding information that can directly be extracted from the observations and a vector $\boldsymbol{m}_t$ to store information inferred over time, e.g., velocities. The former is referred to as the observation or upper part and the latter as the memory or lower part of the latent state by the authors [3]. For an ordinary dynamical system and images as observations the former may correspond to positions while the latter corresponds to velocities. The corresponding posterior and prior covariance matrices $\boldsymbol{\Sigma}_t^+$ and $\boldsymbol{\Sigma}_t^-$ have a chosen factorized representation to yield simple Kalman updates, i.e.,

$$\boldsymbol{\Sigma}_t = \begin{bmatrix} \boldsymbol{\Sigma}_t^{\text{u}} & \boldsymbol{\Sigma}_t^{\text{s}} \\ \boldsymbol{\Sigma}_t^{\text{s}} & \boldsymbol{\Sigma}_t^{\text{l}} \end{bmatrix},$$

where each of $\boldsymbol{\Sigma}_t^{\text{u}}, \boldsymbol{\Sigma}_t^{\text{s}}, \boldsymbol{\Sigma}_t^{\text{l}} \in \mathbb{R}^{m \times m}$ is a diagonal matrix. The vectors $\boldsymbol{\sigma}_t^{\text{u}}, \boldsymbol{\sigma}_t^{\text{l}}$ and $\boldsymbol{\sigma}_t^{\text{s}}$ denote the vectors containing the diagonal values of those matrices. This structure with $\boldsymbol{\Sigma}_t^{\text{s}}$ ensures that the correlation between the memory and the observation parts are not neglected as opposed to the case of designing $\boldsymbol{\Sigma}_t$ as a diagonal covariance matrix. This representation was exploited to avoid the expensive and numerically problematic matrix inversions involved in the KF equations as shown below.

Locally Linear Transition Model

The state transitions in the predict stage of the Kalman filter is governed by a locally linear transition model. To obtain a locally linear transition model, the RKN learns K constant transition matrices $\boldsymbol{A}^{(k)}$ and combines them using state dependent coefficients $\alpha^{(k)}(\boldsymbol{z_t})$, i.e.,

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

$\boldsymbol{A}_t = \sum_{k=0}^{K} \alpha^{(k)}(\boldsymbol{z_t})\boldsymbol{A}^{(k)}$. A small neural network with softmax output is used to learn $\alpha^{(k)}$. Each $\boldsymbol{A}^{(k)}$ is designed to consist of four band matrices as in [3] in order to reduce the number parameters without affecting the performance.

Observation Model

The latent state space $\mathcal{Z} = \mathbb{R}^n$ of the RKN is related to the observation space $\mathcal{W}$ by the linear latent observation model $\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_m & \boldsymbol{0}_{m\times(n-m)} \end{bmatrix}$, i.e., $\boldsymbol{w} = \boldsymbol{H}\boldsymbol{z}$ with $\boldsymbol{w} \in \mathcal{W}$ and $\boldsymbol{z} \in \mathcal{Z}$, where $\boldsymbol{I}_m$ denotes the $m \times m$ identity matrix and $\boldsymbol{0}_{m\times(n-m)}$ denotes a $m \times (n-m)$ matrix filled with zeros. Typically, m is set to $n/2$. This corresponds to the assumption that the first half of the state can be directly observed while the second half is unobserved and contains information inferred over time.

Kalman Update Step

The Kalman update involves computing the Kalman gain matrix $\boldsymbol{Q}_t$, which requires computationally expensive matrix inversions that are difficult to backpropagate, at least for high dimensional latent state representations. However, the choice of a locally linear transition model, the factorized covariance $\boldsymbol{\Sigma}_t$, and the special observation model simplify the Kalman update to scalar operations as shown below. As the network is free to choose its own state representation, it finds a representation where such assumptions works well in practice [3].

Similar to the state, the Kalman gain matrix $\boldsymbol{Q}_t$ is split into an upper $\boldsymbol{Q}_t^{\mathrm{u}}$ and a lower part $\boldsymbol{Q}_t^{\mathrm{l}}$. Both $\boldsymbol{Q}_t^{\mathrm{u}}$ and $\boldsymbol{Q}_t^{\mathrm{l}}$ are squared matrices. Due to the simple latent observation model $\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_m & \boldsymbol{0}_{m\times(n-m)} \end{bmatrix}$ and the factorized covariances, all off-diagonal entries of $\boldsymbol{Q}_t^{\mathrm{u}}$ and $\boldsymbol{Q}_t^{\mathrm{l}}$ are zero and one can again work with vectors representing the diagonals, i.e., $\boldsymbol{q}_t^{\mathrm{u}}$ and $\boldsymbol{q}_t^{\mathrm{l}}$. Those are obtained by

$$\begin{aligned} \boldsymbol{q}_t^{\mathrm{u}} &= \boldsymbol{\sigma}_t^{\mathrm{u},-} \oslash \left(\boldsymbol{\sigma}_t^{\mathrm{u},-} + \boldsymbol{\sigma}_t^{\mathrm{obs}}\right) \\ \boldsymbol{q}_t^{\mathrm{l}} &= \boldsymbol{\sigma}_t^{\mathrm{s},-} \oslash \left(\boldsymbol{\sigma}_t^{\mathrm{u},-} + \boldsymbol{\sigma}_t^{\mathrm{obs}}\right), \end{aligned}$$

where $\oslash$ denotes an elementwise vector division. The update equation for the mean therefore simplifies to

$$\boldsymbol{z}_t^{+} = \boldsymbol{z}_t^{-} + \begin{bmatrix} \boldsymbol{q}_t^{\mathrm{u}} \\ \boldsymbol{q}_t^{\mathrm{l}} \end{bmatrix} \odot \begin{bmatrix} \boldsymbol{w}_t - \boldsymbol{z}_t^{\mathrm{u},-} \\ \boldsymbol{w}_t - \boldsymbol{z}_t^{\mathrm{u},-} \end{bmatrix},$$

where $\odot$ denotes the elementwise vector product. The update equations for the individual parts of covariance are given by

$$\begin{aligned} \boldsymbol{\sigma}_t^{\mathrm{u},+} &= (\boldsymbol{1}_m - \boldsymbol{q}_t^{\mathrm{u}}) \odot \boldsymbol{\sigma}_t^{\mathrm{u},-}, \\ \boldsymbol{\sigma}_t^{\mathrm{s},+} &= (\boldsymbol{1}_m - \boldsymbol{q}_t^{\mathrm{u}}) \odot \boldsymbol{\sigma}_t^{\mathrm{s},-}, \\ \boldsymbol{\sigma}_t^{\mathrm{l},+} &= \boldsymbol{\sigma}_t^{\mathrm{l},-} - \boldsymbol{q}_t^{\mathrm{l}} \odot \boldsymbol{\sigma}_t^{\mathrm{s},-}, \end{aligned}$$

where $\boldsymbol{1}_m$ denotes the m dimensional vector consisting of ones.

Appendix B: Robots and Data

The experiments are performed on data from four different robots. The details of robots, data and data preprocessing is explained below:

Hydraulic Brokk 40 Robot Arm

Observation and Data Set: The data was obtained from a HYDROLEK–7W 6 degree–of–freedom manipulator with a continuous (360 degree) jaw rotation mechanism. We actuate the joints via hydraulic pistons, which are powered via an auxiliary output from the hydraulic pump. Thus learning the forward model is difficult due to inherent hysteresis associated with hydraulic control.

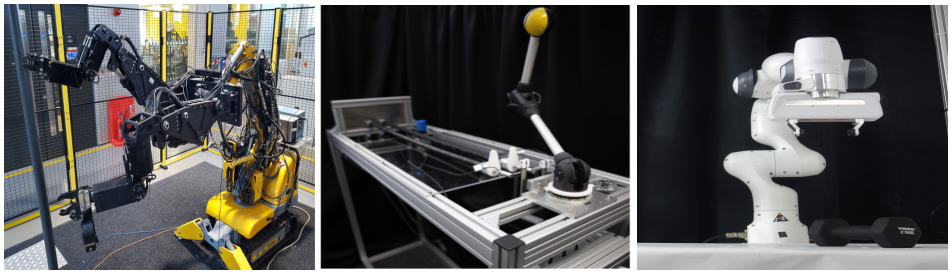


Figure 8: The experiments are performed on data from robots with different actuator dynamics. From left to right these include: Hydraulically actuated BROKK-40 [22], Pneumatically actuated artificial muscles [23], Franka Emika Panda Robotic Arm.

For this robot, only one joint is moved at a time, so we have independent time series per joint. The joint data consists of measured joint positions and the input current to the controller of the joint sampled at 100Hz.

Training Procedure: During training, we work with sequences of length 500. For the first 300 time steps those sequences consist of the full observation, i.e., the joint position and current. We give only the current signals in the remaining 200 time steps. The models have to impute the missing joint positions in an uninformed fashion, i.e., we only indicate the absence of a position by unrealistically high values.

Musculoskeletal Robot Arm

Observation and Data Set: For this soft robot we have 4 dimensional observation inputs(joint angles) and 8 dimensional action inputs(pressures). We collected the data of a four DoF robot actuated by Pneumatic Artificial Muscles (PAMs). The robot arm has eight PAMs in total with each DoF actuated by an antagonistic pair. The robot arm reaches high joint angle accelerations of up to 28, 000deg/s2 while avoiding dangerous joint limits thanks to the antagonistic actuation and limits on the air pressure ranges. The data consists of trajectories collected while training with a model-free reinforcement learning algorithm to hit balls while playing table tennis. We sampled the data at 100Hz. The hysteresis associated with the pneumatic actuators used in this robot is challenging to model and is relevant to the soft robotics in general.

Training Procedure: During training, we randomly removed three-quarters of the states from the sequences and tasked the models with imputing those missing states, only based on the knowledge of available actions/control commands, i.e., we train the models to perform action conditional future predictions to impute missing states. The imputation employs the model for multi-step ahead predictions in a convenient way. One could instead go for a dedicated loss function as in approaches like [18], [17] for long term predictions.

Franka Emika Panda Robot Arm

Observation and Data Set: We collected the data from a 7 DoF Franka Emika Panda manipulator during free motion. We chose this task since the robot exhibits different dynamics behaviour due to electric actuators and high frequencies(1kHz). The raw joint positions, velocities and torques were recorded using Franka Interfaces while the joint accelerations were computed by finite differences on filtered velocity data (obtained using a zero-phase 8th-order digital Butterworth filter with a cut-off frequency of 5Hz). The observations for the forward model consist of the seven joint angles in radians, and the corresponding actions were joint Torques in Nm. While the inverse model use both joint angles and velocities as observations. The data was divided into train and test sets in the ratio 4:1. We divide the data into sequences of length 300 while training the recurrent models for forward dynamics and use sequences of length 50 for inverse dynamics.

Training Procedure Forward Dynamics: Similar to the multi-step ahead training procedure in 6, during training we randomly removed three-quarters of the observations(joint angles) from the se-

quences and tasked the models with imputing those missing observations, only based on the knowledge of available actions/control commands.

Training Procedure Inverse Dynamics: The recurrent models (LSTM, ac-RKN) uses a similar architecture, as shown in Figure 3 of the main paper, except for the recurrent module. The hyperparameters including learning rate, latent state and observation dimensions, learning rate, control model architecture, action decoder architecture and regularization parameter for the joint forward-inverse dynamics loss function are searched via GpyOpt[21] and is mentioned in Appendix D. The observation encoder and decoder architecture is chosen to be of the same size across the models being compared. For all models, we use the joint positions and velocities as the observation input and differences to the next state as desired observation. The FFNN gets the current observation and desired observation as input and is tasked to predict the joint Torques directly(unlike differences in recurrent models) as in previous regression approaches[11].

Barrett WAM Robot Arm

Observation and Data Set: The Barett task is based on a publicly available dataset comprising joint positions, velocities, acceleration and torques of a seven degrees-of-freedom real Barett WAM robot. The original training dataset (12, 000 data points) is split into sequences of length 98. Twenty-four out of the total 119 episodes are utilized for testing, whereas the other 95 are used for training. The direct cable drives which drive this robot produce high torques, generating fast and dexterous movements but yield complex dynamics. Rigid-body dynamics cannot be model this complex dynamics due to the variable stiffness and lengths of the cables.

Training Procedure Inverse Dynamics: The training procedure is repeated as in 6

Appendix C: Details Of Rigid Body Dynamics Model

The analytical model for Franka Emika Panda is a rigid-body dynamics model that was identified in its so-called base parameters [26]. Due to the friction compensation in the joints, we observed that the viscous friction is negligible, whereas the observed Coulomb friction is very small yet included in our parameterization. Which results is a model with 50 parameters. The base parameterization is computed based on provided kinematic properties of the robotic arm and provides a linear relation between the base parameters and joint torques for a given set of joint positions, velocities and accelerations.

Due to this linearity, the regression problem can be solved using a linear least-squares method, although additional linear matrix inequality constraints must be fullfilled to ensure that the resulting parameters are physically realizable [27]. In order to perform forward simulation of the robot dynamics, we numerically solve an initial value problem for the implicit set of differential equations defined by base parameterization of the rigid-body model. Note that, the model does not parameterize actuator dynamics, nor does it model joint flexibilities, link flexibilities, or stiction. The focus here is to provide a reference baseline to show which effects the acRKN captures in comparison to a text-book robot model.

Appendix D: Hyperparameters

Pneumatic Musculoskeltal Robot Arm

Table 1: Forward Dynamics Hyperparameters For Pneumatic Musculoskeltal Robot.

Hyperparameter	ac-RKN	RKN	LSTM
Learning Rate	3.1e-3	1.9e-3	6.6e-3
Latent Observation Dimension	60	60	60
Latent State Dimension	120	120	120

Encoder (ac-RKN,RKN,LSTM): 1 fully connected + linear output (elu + 1)

- Fully Connected 1: 120, ReLU

Observation Decoder (ac-RKN,RKN,LSTM): 1 fully connected + linear output:

- Fully Connected 1: 120, ReLU

Transition Model (ac-RKN,RKN): bandwidth: 3, number of basis: 15

- $\alpha(\boldsymbol{z}_t)$: No hidden layers - softmax output

Control Model (ac-RKN): 3 fully connected + linear output

- Fully Connected 1: 120, ReLU
- Fully Connected 2: 120, ReLU
- Fully Connected 3: 120, ReLU

Architecture For FFNN Baseline 2 fully connected + linear output

- Fully Connected 1: 6000, ReLU
- Fully Connected 2: 3000, ReLU

Dropout Regularization - 0.512
Learning Rate - 1.39e-2
Optimizer Used: Adam Optimizer

Hydraulic Brokk-40 Robot Arm

Table 2: Forward Dynamics Hyperparameters For Pneumatic Musculoskeltal Robot.

Hyperparameter	ac-RKN	RKN	LSTM	GRU
Learning Rate	5e-4	5e-4	9.1e-4	2.1e-3
Latent Observation Dimension	30	30	30	30
Latent State Dimension	60	60	60	60

Encoder (ac-RKN,RKN,LSTM,GRU): 1 fully connected + linear output (elu + 1)

- Fully Connected 1: 30, ReLU

Observation Decoder (ac-RKN,RKN,LSTM,GRU): 1 fully connected + linear output:

- Fully Connected 1: 30, ReLU

Transition Model (ac-RKN,RKN): bandwidth: 3, number of basis: 32

- $\alpha(\boldsymbol{z}_t)$: No hidden layers - softmax output

Control Model (ac-RKN): 1 fully connected + linear output

- Fully Connected 1: 120, ReLU

Franka Emika Panda - Forward Dynamics Learning

Table 3: Forward Dynamics Learning Hyperparameters For Panda.

Hyperparameter	ac-RKN	RKN	LSTM	GRU
Learning Rate	3.1e-3	1.7e-3	6.6e-3	8.72e-3
Latent Observation Dimension	45	30	30	45
Latent State Dimension	90	60	60	90

Encoder (ac-RKN,RKN,LSTM,GRU): 1 fully connected + linear output (elu + 1)

- Fully Connected 1: 120, ReLU

Observation Decoder (ac-RKN,RKN,LSTM,GRU): 1 fully connected + linear output:

- Fully Connected 1: 240, ReLU

Transition Model (ac-RKN,RKN): bandwidth: 3, number of basis: 15

- $\alpha(\boldsymbol{z}_t)$: No hidden layers - softmax output

Control Model (ac-RKN): 3 fully connected + linear output

- Fully Connected 1: 30, ReLU
- Fully Connected 2: 30, ReLU
- Fully Connected 3: 30, ReLU

Architecture For FFNN Baseline - Forward Dynamics 3 fully connected + linear output

- Fully Connected 1: 1000, ReLU
- Fully Connected 2: 1000, ReLU
- Fully Connected 3: 1000, ReLU

Dropout Regularization - 0.1147
Learning Rate - 8.39e-3
Optimizer Used: SGD Optimizer

Franka Emika Panda - Inverse Dynamics Learning

Table 4: Inverse Dynamics Learning Hyperparameters For Panda.

Hyperparameter	ac-RKN	RKN (No Action Feedback)	LSTM
Learning Rate	7.62e-3	3.5e-3	9.89e-3
Latent Observation Dimension	15	30	30
Latent State Dimension	30	60	60
Regularization Factor (λ)	0.158	0.179	0.196

Encoder (ac-RKN,RKN,LSTM): 1 fully connected + linear output (elu + 1)

- Fully Connected 1: 120, ReLU

Observation Decoder (ac-RKN,RKN,LSTM): 1 fully connected + linear output:

- Fully Connected 1: 240, ReLU

Action Decoder (ac-RKN,RKN,LSTM): 1 fully connected + linear output:

- Fully Connected 1: 512, ReLU

Transition Model (ac-RKN,RKN): bandwidth: 3, number of basis: 15

- $\alpha(\boldsymbol{z}_t)$: No hidden layers - softmax output

Control Model (ac-RKN): 1 fully connected + linear output

- Fully Connected 1: 45, ReLU

Architecture For FFNN Baseline - Inverse Dynamics 3 fully connected + linear output

- Fully Connected 1: 500, ReLU
- Fully Connected 2: 500, ReLU
- Fully Connected 3: 500, ReLU

Dropout Regularization - 0.563
Learning Rate - 1.39e-2
Optimizer Used: SGD Optimizer

Barett WAM - Inverse Dynamics Learning

Table 5: Inverse Dynamics Learning Hyperparameters Barett WAM.

Hyperparameter	ac-RKN	RKN (No Action Feedback)	LSTM
Learning Rate	7.7e-3	1.7e-3	9.33e-3
Latent Observation Dimension	15	30	45
Latent State Dimension	30	60	90
Regularization Factor (λ)	0.176	0	3.42e-3

Encoder (ac-RKN,RKN,LSTM): 1 fully connected + linear output (elu + 1)

- Fully Connected 1: 120, ReLU

Observation Decoder (ac-RKN,RKN,LSTM): 1 fully connected + linear output:

- Fully Connected 1: 240, ReLU

Action Decoder (ac-RKN): 2 fully connected + linear output:

- Fully Connected 1: 256, ReLU
- Fully Connected 1: 256, ReLU

Action Decoder (RKN,LSTM): 1 fully connected + linear output:

- Fully Connected 1: 512, ReLU

Transition Model (ac-RKN,RKN): bandwidth: 3, number of basis: 15

- $\alpha(\boldsymbol{z}_t)$: No hidden layers - softmax output

Control Model (ac-RKN): 1 fully connected + linear output

- Fully Connected 1: 45, ReLU

Architecture For FFNN Baseline 3 fully connected + linear output

- Fully Connected 1: 500, ReLU
- Fully Connected 2: 500, ReLU
- Fully Connected 3: 500, ReLU

Dropout Regularization - 0.563
Learning Rate - 1e-5
Optimizer Used: SGD Optimizer