Arndt, Karol; Ghadirzadeh, Ali; Hazara, Murtaza; Kyrki, Ville

Few-shot model-based adaptation in noisy conditions

Published in:
IEEE Robotics and Automation Letters

DOI:
10.1109/LRA.2021.3068104

Published: 01/04/2021

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Arndt, K., Ghadirzadeh, A., Hazara, M., & Kyrki, V. (2021). Few-shot model-based adaptation in noisy conditions. *IEEE Robotics and Automation Letters*, *6*(2), 4193-4200. Article 9384205. https://doi.org/10.1109/LRA.2021.3068104

Few-shot model-based adaptation in noisy conditions

Karol Arndt[1], Ali Ghadirzadeh[2], Murtaza Hazara[1,3] and Ville Kyrki[1]

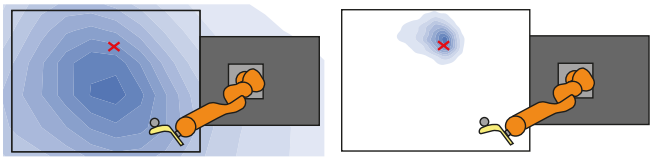
***Abstract*—Few-shot adaptation is a challenging problem in the context of simulation-to-real transfer in robotics, requiring safe and informative data collection. In physical systems, additional challenge may be posed by domain noise, which is present in virtually all real-world applications. In this paper, we propose to perform few-shot adaptation of dynamics models in noisy conditions using an uncertainty-aware Kalman filter-based neural network architecture. We show that the proposed method, which explicitly addresses domain noise, improves few-shot adaptation error over a blackbox adaptation LSTM baseline, and over a model-free on-policy reinforcement learning approach, which tries to learn an adaptable and informative policy at the same time. The proposed method also allows for system analysis by analyzing hidden states of the model during and after adaptation.**

I. Introduction

Applying machine learning techniques to robot control tasks is a challenging problem, largely due to low sample efficiency of most machine learning methods. In the case of reinforcement learning, additional challenge lies in the random exploration process, which takes place during learning, and poses a major risk of hardware damage.

A popular solution lies in using artificial data [1], [2] or physics simulators [3] to facilitate the training; yet, in many cases, the simulation is not accurate enough for the model to achieve optimal performance in the real world without additional adaptation [4]–[6].

While few-shot learning has been quite extensively studied by the machine learning community, especially within the framework of meta-learning [7], [8], most prior works assume that noise-free labels are available for the learner during the adaptation process. However, certain real-world physical systems can be noisy or stochastic; that is, executing the same action in the same state may result in different observations. In such scenarios, performing few-shot adaptation to real conditions may pose an additional challenge. It has been shown that modelling and utilization of uncertainty information is effective in guiding and speeding up the policy learning for novel robotic tasks [9]; yet, in many real-world problems, the noise characteristics are not known in advance and thus cannot be easily injected into the simulated data.



(a) (b)

Fig. 1: Before seeing any samples, the model produces large uncertainty about the final puck position (a). After performing some actions in the environment, the prediction uncertainty falls down (b). The red cross marks the true position.

In this paper, we present an uncertainty-aware meta-learning approach to adapt a dynamics model trained across a variety of conditions in simulation to the physical world. Our goal is to enhance sim-to-real transfer of dynamic skills in terms of data-efficiency and speed of adaptation through uncertainty-awareness. In contrast to previously explored gradient-based methods [5], [6], [8], which merely look at the mean gradient direction (ignoring the variance information), we propose to use a memory-based approach, which allow the model to keep track of uncertainty statistics [10]. This uncertainty can be projected directly onto the predictions, as shown in Figure 1; before adaptation (Figure 1a) the model reports large uncertainty about the position of the hockeypuck after a planned hit, while after adaptation the uncertainty falls down (Figure 1b). In our method, this is achieved by using a network architecture designed around a trainable Kalman filter [11]–[13].

The focus on model adaptation, as opposed to policy adaptation, removes the need to run a random exploratory policy altogether; instead, the data can be collected with the assistance of a human operator, or by a specifically designed policy which is known to be safe. It also allows for the adapted environment model to be reused for different tasks in the same environment—namely, the same adapted state transition model can be used to maximize different objective functions in the environment, provided that the state-action space region relevant for the new task has been explored, or its properties have been identified based on prior information from the meta-training phase.

The contributions of this paper are as follows: (1) presenting a novel noise-aware meta-learning method based on a trainable Kalman filter, (2) showing that the proposed model structure outperforms LSTM and MAML on domain adaptation tasks in noisy conditions, (3) demonstrating that the learned latent representations of dynamic conditions are interpretable, corre-

This work was supported by Academy of Finland grants 313966 and 317020. We also gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. The authors also wish to acknowledge CSC – IT Center for Science, Finland, for computational resources.

[1]Intelligent Robotics Group,Department of Electrical Engineering and Automation, Aalto University, Espoo, Finland first.last@aalto.fi

[2]Computer Science and Electrical Engineering at Stanford University ghadiri@stanford.edu

[3]Department of Mechanical Engineering, KU Leuven, Belgium murtaza.hazara@kuleuven.be

sponding to physical parameters, (4) showing that model adaptation through meta-learning is more data efficient compared to policy adaptation.

II. Related work

A. Meta-learning

Many meta-learning approaches are based on a recurrent neural network architecture, which the hidden state is updated based on the observed learning data [14]. As the optimization procedure itself is learned together with the state representations, these methods are often referred to as *blackbox meta-learning*.

With the advancement of deep learning, another family of meta-learning methods—optimization-based meta-learning—was introduced by Finn et al. with the model-agnostic meta-learning algorithm (MAML) [8]. Multiple extensions to the original algorithm were proposed later [15], [16]. These methods, however, rely only on the mean gradient direction calculated over multiple samples while discarding the variance, and do not encode the uncertainty information in any way. Various probabilistic extensions to MAML were introduced [17], [18]; these approaches rely on ensemble models or Stein variational gradient descent to introduce sampling operations to the network during training, in order to encode uncertainty in the network output.

Our approach to model-based meta adaptation is based on the idea of training Kalman filters via backpropagation, as proposed in [12]. In that work, a Kalman filter is embedded inside a neural network and trained via backpropagation. The architecture proposed in [12] is shown to improve performance on standard state estimation tasks with deep learning over other recurrent architectures. We propose to use a similar architecture for adaptation of dynamics models, rather than standard state estimation.

Recently, some considerations on uncertainty-aware meta-learning were put forward by Gordon et al. [19], where it is proposed to view meta learning as learning a distribution over task-specific parameters. Our method can be viewed as a special case of this generalized framework, where the parameters are modeled as Gaussian distributions and the inference is performed using standard Kalman filter update rules.

B. Sim-to-real transfer in robotics

The problem of sim-to-real transfer in robotics has been widely addressed in recent years, especially within the context of reinforcement learning. In the most basic approach, policies are trained in simulation and reused in the real world. This, however, requires tedious fine-tuning of simulation parameters, and sometimes requires extreme measures, such as disassembling the robot and measuring the individual components in order to fine-tune the simulation [20]. Moreover, some physical phenomena such as backlash cannot be modeled in simulation, making the policies learned in simulation inefficient for real world.

The view-invariant servoing approach presented in [21] can be thought of as an example of blackbox memory-based meta-learning; the recurrent model is trained over a variety of different simulated conditions such that it can adapt to real world conditions, as more and more samples are collected during operation. This method, however, focuses on optimizing a task-specific policy, which requires that policy to be run on the physical setup before being exposed to any real world data. Additionally, the collected data cannot be reused for other tasks, and the method does not account for the uncertainty present in real-world data.

Previous work on meta-learning for sim-to-real transfer focused on policy adaptation [5], [6], [22]. With these approaches, the final policy is a direct result of an on-policy update performed on the policy used to collect the data, as in [8]. As such, not only do the meta-policy parameters need to constitute a starting point for further adaptation, but also the meta-policy itself needs to explore the environment in a way which provides meaningful information for adaptation [16]; there is, however, no straightforward and universal way of balancing between these two objectives [16], [23], [24]. Using our method, on the other hand, a dynamic model can be optimized using data collected from a policy which is known to be safe, and is capable of providing informative samples.

Model adaptation with meta-learning has been previously utilized by Clavera et al. [25] as a method of regularizing and stabilizing model-based reinforcement learning. This method, however, only considered minor discrepancies between models, and—like other gradient-based methods—discarded uncertainty information in the update rule. The method we propose in this paper, in contrast, aims at adapting to a wide variety of conditions and explicitly keeps track of the uncertainty (as expressed by the variance of the task-specific parameter vector).

III. Method

In this section, we first provide a formal statement of the problem we are addressing in this work. We then proceed with a more detailed description of our approach to the presented problem.

A. Problem formulation and solution overview

Given a state s_i and an empty initial database $\mathcal{D}_0$ of state transitions (s, a, s'), we consider the problem of successively choosing one of N actions. Each action a_i leads to a new state s'_i. In addition, the state observation process may be disturbed by random noise, leading to noisy observation $\tilde{s}'$. Successively, we update the current database $\mathcal{D}_i = \mathcal{D}_{i-1} \cup \{(s_i, a_i, s'_i)\}$ with the newly observed state action pair (s_i, a_i, s'_i). We assume that the dynamics of the considered sequential decision making problem can be modeled as a supervised learning problem:

$$\begin{pmatrix} s'_i & \phi_i \end{pmatrix}^\top = f_\theta(s_i, a_i, \phi_{i-1}) \tag{1}$$

using a function approximator $f_\theta(\cdot)$ where θ denotes its parameters and ϕ_{i-1} represents the hidden state which is governed by a function g of previous action and noisy state observations:

$$\phi_{i-1} = g(\mathcal{D}_{i-1}) \tag{2}$$

In the proposed method, the hidden state ϕ is a task-specific parameter vector, describing the unknown system dynamics.

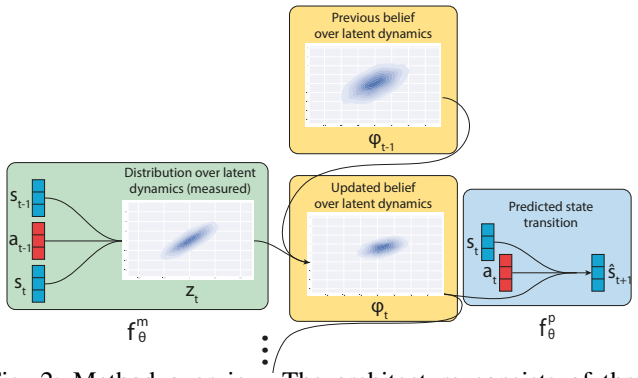


Fig. 2: Method overview. The architecture consists of three main parts: measurement (green), state integration (yellow), and prediction (blue).

In order to estimate this state from noisy measurements, we build a neural network, as shown in Figure 2. The architecture consists of three parts: measurement (which estimates the system dynamics based on a noisy observation of a state transition), state integration (which integrates the parameter estimations together), and prediction (which estimates the next state given the previous state and the action).

B. Measurement model

The measurement model, shown in green in Figure 2, estimates the distribution of system dynamics $\tilde{z}$ given a state transition: $\tilde{z}_i \sim f_\theta^m(s_i, a_{i-1}, \tilde{s}'_{i-1})$. To make the system robust to observation noise, we corrupt the state observations with Gaussian noise: $\tilde{s} = s + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \Sigma_s)$. As we assume that we do not know the exact level of noise variance of the real environment, we incorporate Σ_s into the task description, with Σ_s being a diagonal matrix with values sampled from a uniform distribution:

$$diag(\Sigma_s) \sim \mathcal{U}(0, \sigma^2_{s,max}) \tag{3}$$

The dynamics representations z and ϕ are learned by the neural network during training with backpropagation, as visualized in Figure 2. The distribution described by f^m is modeled as a Gaussian parametrized by mean and covariance: $\tilde{z} = \mathcal{N}(\mu_z, \Sigma_z)$. This procedure corresponds to step 9 of Algorithm 1.

For many systems, it is impossible to uniquely determine the dynamic conditions based on a single measurement; i.e., the system can be seen as underdetermined from the parameter estimation perspective. Thus, in addition to uncertainty introduced by measurement noise, the measurement covariance matrix has to convey the uncertainty caused by the possible underdetermined nature of the system. We hence use heteroskedastic uncertainty, with covariance Σ_z predicted for each measurement. The returned measurement can thus be interpreted as a probability distribution over all systems from which the observed state transition could originate. It is worth noting that the non-linear measurement model can map an arbitrary continuous state distribution to a Gaussian in the latent space. Thus, despite using a Kalman filter (which assumes all variables it operates on to be normally distributed), the proposed method can, in theory, handle arbitrary noise distribution in the input space by mapping them to Gaussian distributions parameterized by μ_z, Σ_z.

C. Integration

The measurement distribution returned by the measurement model is passed to the recurrent part of the network, marked in yellow in Figure 2, which is responsible for integrating the observations together. In our method, this is achieved by a deep Kalman filter [12]—a Kalman filter embedded within a neural network, where the parameter matrices of the filter are trained with backpropagation together with the rest of the network. Such a network has the inference procedure built-in in the computation graph, which was shown to improve its ability to integrate information from individual samples and reason about uncertainty [12].

Algorithm 1: Training the dynamics model

Input: Dataset $\{(s, a, s')\}$ of state transitions
Result: Network parameters θ

randomly initialize network parameters θ;
while *not converged* **do**
  Reset internal parameters (μ_ϕ and Σ_ϕ);
  Sample task τ (dynamics and noise variance, Eq. (3));
  Sample a sequence of N state transitions (s, a, s') from τ;
  for *each transition* (s, a, s') *in the sequence* **do**
    Generate noise: $\tilde{s'} \sim \mathcal{N}(s', \Sigma_s)$;
    Update state estimate (Eqs. 4 and 5);
    Estimate the dynamic conditions $\mu_z, \Sigma_z = f_\theta^m(s, a, \tilde{s'})$;
    Update μ_ϕ, Σ_ϕ given μ_z, Σ_z (Eqs. 6, 7, 8);
    Predict $\hat{s'} = f_\theta^p(s, a, \mu_\phi)$;
  end
  Calculate loss over the sequence (Eq. 9) ;
  Calculate $\nabla_\theta \mathcal{L}$ and update θ using Adam ;
end

The first step of a Kalman filter performs a state prediction based on a learned model

$$\mu_\phi^{t+1|t} = A\mu_\phi^{t|t} + Bu, \tag{4}$$

where $\mu_\phi^{t|t}$ represents the mean of the current belief about the state of the system, and u represents external input. In our case, where the Kalman filter state actually represents a belief over the latent dynamic parameters of the system, this update represents a temporal change in dynamics—thus, for a stationary system, A can be explicitly set to the identity matrix. In this formulation, the action u would correspond to an external action which directly impacts the dynamics. We assume that no such action takes place, and thus we set B to zero.

The state estimation is modeled as a Gaussian with the covariance

$$\Sigma_\phi^{t+1|t} = A^\top \Sigma_\phi^{t|t} A + Q. \tag{5}$$

In the state dynamics prediction formulation, a small non-zero value of the covariance matrix Q can be used to approximate

unmodeled drift in dynamic conditions and prevent the estimation covariance from approaching zero in limit, enabling lifelong learning. This procedure corresponds to step 8 of Algorithm 1.

The state prediction is updated in step 10 of Algorithm 1 by integrating information coming from a measurement, following the standard Kalman filter equations [12], and resulting in $\mu_\phi^{t+1|t+1}$ and $\Sigma_\phi^{t+1|t+1}$. First, the Kalman gain is calculated with

$$K_t = \Sigma_\phi^{t+1|t} C_z^\top (C_z \Sigma_\phi^{t+1|t} C_z^\top + \Sigma_z)^{-1} \tag{6}$$

Then, the belief about the mean μ_ϕ is updated with

$$\mu_\phi^{t+1|t+1} = \mu_\phi^{t+1|t} + K_t(z_t - C_z \mu_\phi^{t+1|t}) \tag{7}$$

Finally, the covariance of the state belief is updated by

$$\Sigma_\phi^{t+1|t+1} = (I - K_t C_z)\Sigma_\phi^{t+1|t} \tag{8}$$

We use a standard linear Kalman filter, as the non-linear correspondence between the measurements and internal states is already addressed by the non-linear measurement system. The values of C_z and the initial state distribution (parametrized by μ_ϕ^0 and Σ_ϕ^0) are learned during outer-loop optimization. The use of Kalman filters allows us to introduce additional information in the prediction model—namely, we assume that the system is stationary (the transition matrix $A = I$), while allowing for some temporal drift to encourage the system to keep adapting over long periods of time and to improve numerical stability ($Q = \epsilon I$).

D. State prediction

Finally, the future state of the environment is predicted for the queried state-action pair by the *prediction model* $\hat{s}_i' = f_\theta^p(s_i, a_i, \mu_\phi)$. This model, like the measurement model, is a neural network. The state prediction takes place in step 8 of Algorithm 1.

During meta-training on data collected in a simulated environment, the optimization objective $\mathcal{L}$ is calculated as the mean-squared error between the predictions of the entire model at each timestep over the whole sequence, and the noiseless ground-truth observations s_i':

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(\hat{s_i}' - s_i')^2. \tag{9}$$

The loss calculation and model optimization are performed, respectively, in steps 13 and 14 of Algorithm 1.

The whole model—the measurement model f_θ^m, the prediction model f_θ^p, and the parameters of the Kalman filter C_z and $\mu_{\phi,0}$—is trained as a whole in an end-to-end fashion.

When the model is deployed, the parameters are initialized to the learned values (θ, $\mu_{\phi,0}$, and $\Sigma_{\phi,0}$), and steps 6–12 of the algorithm are repeated for each collected data point.

IV. Experiments

In this section, we provide the details of the experimental evaluation of our method and present the results. In order to illustrate the details of the adaptation process, we first study the performance of the method on a simple regression problem with a step-by-step walkthrough. For the sake of demonstration, we visualize how the state and output distributions are changing as more samples are observed. Then, we evaluate the method's ability to adapt to different simulated conditions with different noise levels in MuJoCo on FetchSlide, an environment from OpenAI Gym [26]. Finally, we evaluate the sim-to-real performance of the method on a hockeypuck hitting task, using simulated data for training and data collected from a real robot for adaptation and evaluation.

Parameter	Distribution
Linear friction	$\mathcal{U}(0.05, 0.7)$
Torsional friction	$\mathcal{U}(10^{-4}, 10^{-2})$
Rotational friction	$\mathcal{U}(5 \cdot 10^{-5}, 10^{-3})$

TABLE I: FetchSlide randomization parameters

The comparison is performed both against gradient-based adaptation methods [5], [8] and against an LSTM baseline [14], which uses a blackbox adaptation scheme and does not explicitly account for noise.

A. Linear regression

In order to illustrate how the internal state ϕ and the prediction are changing during the adaptation process on a basic example, we devise a simple regression problem, where the goal is to predict values of a linear function based on a small amount of noisy observations $(x_i, \tilde{y}_i)$. In this setup, instead of taking state-action pairs (s, a) and predicting future states $\hat{s}'$, the model receives x as input and predicts the value of the function, $\hat{y}$.

The changes in state and prediction distributions are shown in Figure 3. Figure 3a shows the prior state distribution over parameters. This distribution is learned during outer-loop optimization in the meta-learning phase and represents the prior over all dynamics conditions seen during training, which minimizes the expected prediction error over all samples, before any data is observed. This behaviour arises as a result of including the initial prediction error in the optimization objective (Eq. 9). We sample values of the output $\hat{y}$ by sampling hidden state values ϕ from $_\phi, \pm_\phi$ and passing the sampled hidden state values to the prediciton model. This has been shown in Figure 3b). The shaded area shows one standard deviation in state space projected onto the output domain.

After two samples are obtained, the variance of the latent variable noticeably goes down (Figure 3c). Due to noise, the observed points do not exactly align with the true function, and the predicted line does not exactly go through the observed points; rather, it is still influenced by the prior (Figure 3d), as represented as the initial state belief.

As more samples are observed, the state and prediction variance further goes down and the prediction is more accurate (Figures 3e and 3f), despite the observations being very noisy. The impact of the prior is also reduced as more samples are observed. Due to non-zero value of Q, the prediction uncertainty will never go down to zero, allowing for lifelong adaptation to changing conditions.

B. FetchSlide

To verify that the method is suitable for domain adaptation in noisy conditions, we built an experimental setup based on the FetchSlide environment from OpenAI Gym [26]. In this

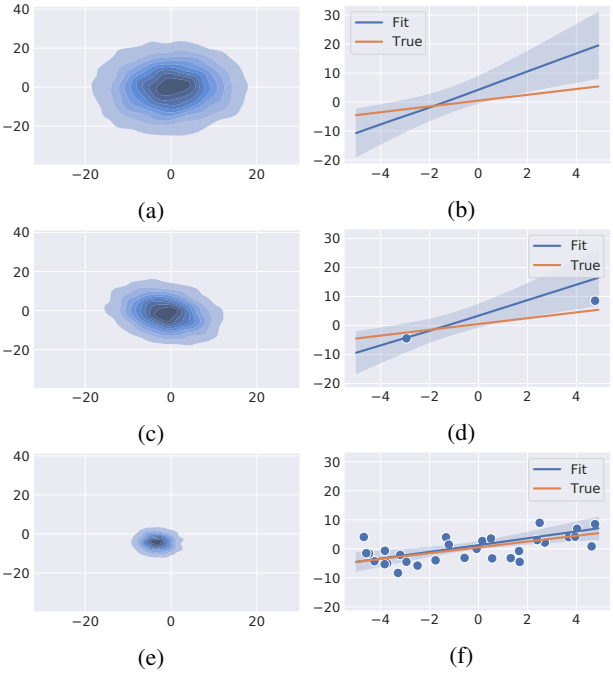


Fig. 3: Analysis of adaptation mechanism, visualized on linear regression with 0 (a, b), 2 (c, d), and 30 samples (e, f). The left column shows hidden state belief distributions, and the right column shows the corresponding function, with mean and standard deviation. The blue dots in (d) and (f) show the observed samples.

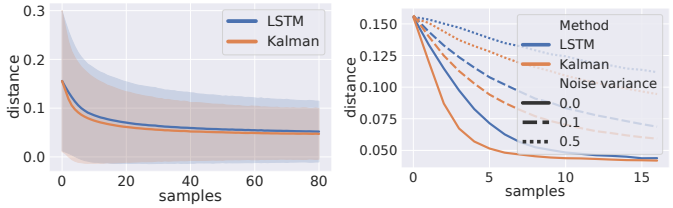


Fig. 4: Domain adaptation results in FetchSlide

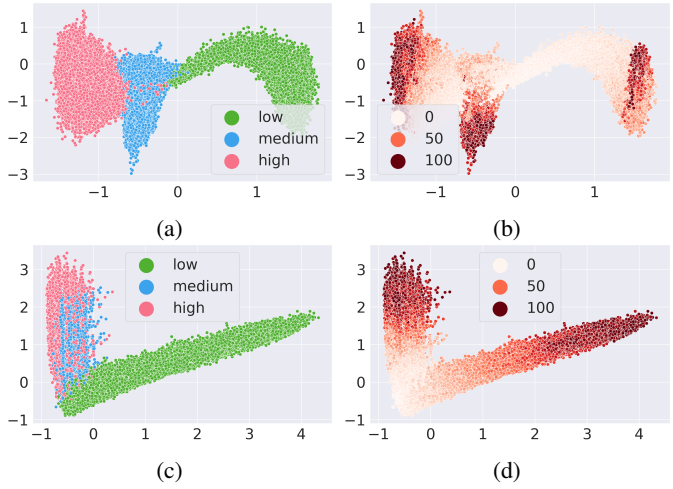


Fig. 5: PCA analysis of hidden states in FetchSlide for the proposed (a, b) and LSTM-based (c, d) model adaptation

environment, the goal is to hit an object located at a random reachable location on the table such that it slides to a given target location. In addition to the robot joint position, both the start and target locations are known to the agent and included in the state vector. The robot is controlled via velocity commands given in Cartesian space. We used 16 dimensions for the measurement and dynamics representations z and ϕ.

Following prior work on using generative models and latent space trajectory representations [1], [5], [27], we use a variational autoencoder trained on a set of task-specific trajectories provided by human experts. This approach turns a sequential decision making problem into a multi-armed bandit problem and provides a straightforward way of collecting data for model adaptation—as the latent distribution is known to be a unit Gaussian, new trajectories can be generated by simply sampling latent vectors from that distribution and passing the latent values to the decoder. We trained the trajectory model on a set of 5.6 million trajectories which move the arm to a random point on the table and push the object away from the robot with a random angle and velocity.

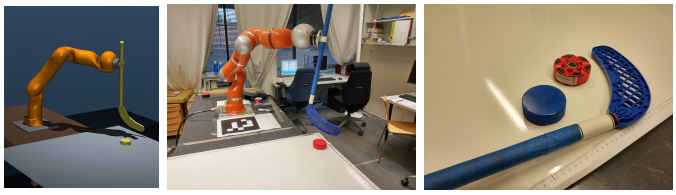
Using the trajectory model, we collected a set of 6.93 million hits under 28875 random friction conditions in simulation. The parameters used for friction distribution are presented in Table I. For both the baseline and the proposed method, we trained the final dynamics model on sequences of 100 samples. We used $\sigma^2_{s,max} = 0.3$. For each method, we trained four models with different random seeds with Adam [28] and averaged the test results.

The evaluation data was collected in the simulator under three scenarios—low, medium, and high friction. During evaluation, we additionally consider three scenarios—no observation noise, low noise ($\sigma^2 = 0.1$) and high noise ($\sigma^2 = 0.5$), where the high noise condition corresponds to larger noise variance than was seen in training. The results of this evaluation are shown in Figure 4.

We see that the proposed method outperforms the baseline most notably in the low-sample regime and high noise conditions. Improved ability to generalize to out-of-distribution tasks can be attributed to encoding optimization within the computation graph, similarly to how MAML improves generalization performance through embedding gradient descent [29].

We additionally analysed the principal components (PCs) of the hidden states ϕ in both networks. The results are shown in Figure 5. The left subplots (5a and 5c) are color-coded by the evaluation domain (low, medium or high friction) and the right ones (5b and 5d) by the number of samples observed by the model. Looking at Figure 5a, we can observe that the dynamics lie in the order of decreasing friction, with high friction on the left, medium friction in the middle, and low friction much farther to the right.

We also observe that the LSTM model encodes uncertainty, as expressed by the number of observed samples, in the hidden states. Additionally, it does not clearly separate the medium and high friction conditions; rather, the PCA projections span the same area even after observing 80 samples. The proposed Kalman filter-based architecture clearly splits the two domains already with less than 20 samples.



(a) (b) (c)

Fig. 6: The simulated (a) and real (b) experimental setups, with a close-up of tools used for the experiments (c)

C. Hockey puck

For the sim-to-real experiments, we used the same hockey puck setup as in [5]. This setup consists of a KUKA LBR4+ robot arm equipped with a floorball stick, a horizontally placed whiteboard, and a two hockey pucks with different masses and friction characteristics: an inline hockey puck and an ice hockey puck. The goal is to hit the hockey puck with the stick such that the puck stops at a given target location. Solving this problem requires the model to learn the friction parameters between the puck and the surface it is sliding on, as, after the puck is hit, friction is the main force causing it to stop. This setup is very similar to FetchSlide, with the additional challenge posed by some phenomena which occur in the physical world, but are not modelled in the simulator, such as hockey blade bending due to elasticity or the whiteboard surface not being perfectly flat and uniform. The state space contains the robot joint position and the target position of the puck. Unlike in FetchSlide, the starting position is not included in the state space, since small deviations are considered a part of the task space that the model has to adapt to. The experimental setup and the tools are presented in Figure 6.

The position of the hockeypuck is measured by a top-mounted Kinect camera. While the camera itself has quite small inaccuracy, we noticed that the system dynamics are not fully deterministic; executing the same trajectory with the same starting puck position (up to a margin of manual positioning error) may produce different results with the standard deviation in position around 5–10 cm, depending on the trajectory and the puck used. We use the same trajectory generation approach as in the FetchSlide experiments, with the trajectories being generated in joint space instead of Cartesian space.

1) Hockeypuck simulation experiments: The simulated setup was constructed in MuJoCo [30], based on the physical setup. In order to generate training data from a wide variety of conditions, we randomized the friction between the hockey puck and the whiteboard surface, as well as the mass of the puck. To account for slight misalignments between the physical and simulated setups, we additionally randomized the starting position of the puck. We added random noise in a similar fashion to FetchSlide, with noise variance sampled from $\sigma_s^2 \sim \mathcal{U}(0, 0.5)$. These randomizations make up for a much wider range of system dynamics than tested in FetchSlide, and match the configurations presented in [5].

We also compare the performance to model adaptation with MAML [8]. Due to the gradient update rule in MAML being batched, as opposed to recurrent architectures, we test MAML

Parameter	Distribution
Puck mass	$\mathcal{U}(0.01, 0.1)$
Linear friction μ_x	$\mathcal{U}(0.15, 0.95)$
Linear friction μ_y	$\mathcal{U}(0.7\mu_x, 1.3\mu_x)$
Torsional friction	$\mathcal{U}(0.001, 0.05)$
Rotational friction	$\mathcal{U}(0.01, 0.3)$
Initial position error	$\mathcal{N}(0, 0.02)$

TABLE II: Randomization parameters

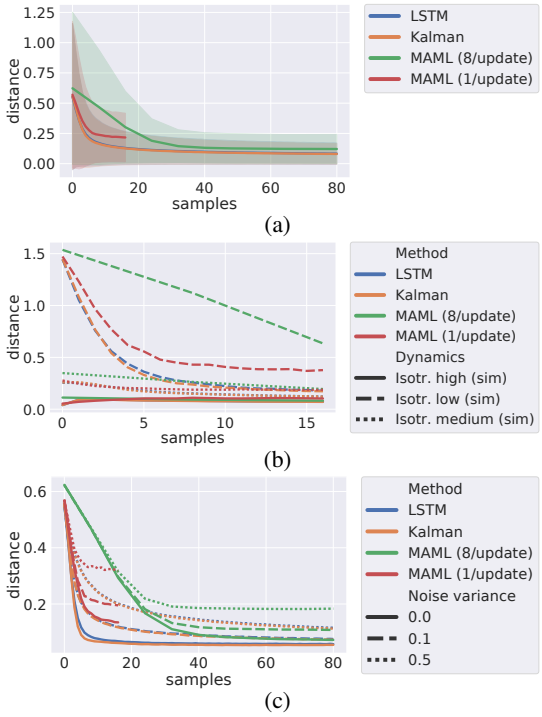


Fig. 7: Performance in HockeyPuck (simulation), averaged over all test conditions (a), shown separately for different simulated conditions (b), and for each noise level (c)

models trained with different amounts of environment rollouts used per update step, where smaller batch sizes allow the model to adapt with fewer samples, but the updates are less accurate. The network used for MAML consists of 4 fully connected layers with 128 neurons each. We performed up to 3 adaptation steps with MAML during training, as more steps led to instability during training and adaptation. The gradient update step size α is learned during training.

The results of the evaluation in simulation, as expressed by distance between the predicted and true position in meters, are shown in Figure 7. We can see that, in a scenario where both models were trained and evaluated on data from simulation (which falls within the task distribution seen during training), there is no significant change in performance between the proposed architecture and LSTM; both methods perform the same with similar error margins (the shaded areas indicate the standard deviation of prediction errors).

In this benchmark, both recurrent methods also outperform MAML [8], as shown in Figure 7. In Figure 7a, we observe that performing MAML updates with 1 sample per update achieves comparable performance improvements at first, but due to noise converges to a large average error of over 20 cm. Performing more ”stable” MAML updates with 8 samples per update achieves comparable, but slightly worse,

final performance, yet performs much worse when only a few samples are available.

In Figure 7b, we detail the adaptation error for a selection of different conditions: low, medium and high isotropic frictions in the low sample regime. We can observe that the adaptation error is the highest in the low friction case. Figure 7c shows the error for each method in various noise conditions. We again observe that both the LSTM and Kalman methods outperform MAML, with the Kalman approach providing slightly better performance. We also observe that, as expected, the difference between noisy and noiseless conditions for MAML is more pronounced when the update batch size is small.

2) Physical experiments: In order to test the suitability of our method for sim-to-real transfer, we generated random trajectories using the generative model, executed them on the physical setup, recorded the resulting puck positions, and evaluated each method on the resulting data. In the real world evaluation, no additional noise is added—thus, step 7 of Algorithm 1 is skipped. The results are shown in Figure 8.

In Figure 8a, we observe that both recurrent methods outperform MAML in the low sample area. Analyzing the performance of various model-based MAML updates, we observe that, similarly to simulation, updating with one sample achieves the best performance at first, but converges to a fairly large error. With 4 samples per update, MAML is able to match the performance of recurrent after about 20 samples (around 3 times more). We can also observe that the Kalman filter-based method outperforms the blackbox LSTM architecture in the low-sample regime, as highlighted in Figure 8b. We observe that the Kalman filter-based approach is able reach error below 14cm with 8 samples, while the LSTM approach requires 12 (50% more). We believe that, due to inaccuracies between simulated and real conditions, the real-world data lies a bit out of the training distribution of the model, especially in terms of the noise distribution; thus, having the inference procedure embedded in the graph, can improve generalization to out-of-task distributions. In [29], it was observed that enforcing the inner-loop update to gradient descent improves generalization to classification tasks with data coming from outside the training domain, in comparison to learned update rules; this is because, even for out-of-domain data, gradient descent still constitutes a sensible update rule. In a similar fashion, based on our results, we can state that the Kalman filter update rules provide better handling of out-of-domain data distributions in comparison to learned update rules, as is the case with LSTM.

In Figure 8c, we show the estimation error for each method for the two hockeypucks we used for the evaluation, zoomed in to the low sample area. We observe that both recurrent methods outperform MAML, with the proposed Kalman filter-based approach outperforming LSTM for both pucks.

We also compare our results to the results obtained in [5], where a reinforcement learning policy was trained in simulation for the same setup and task. In that work, the average position error for the red puck was 14.4 cm after observing 16 real-world samples; with the model-based adaptation approach, we achieve a comparable error (14.2 cm) after observing only 7 samples. Similarly, for the blue puck the average error after observing 16 samples was 27.7 cm, while the proposed method

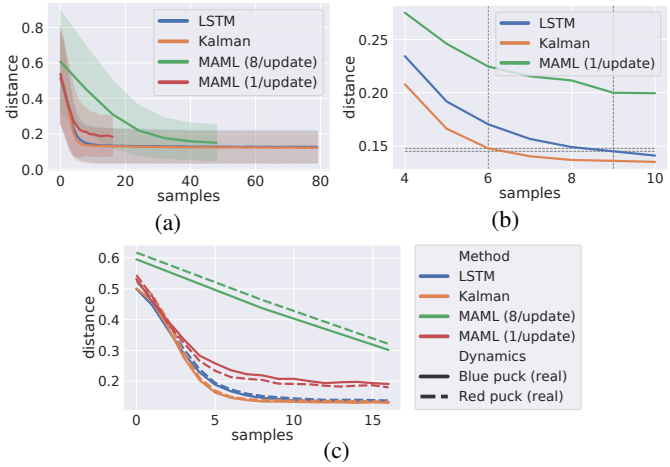


Fig. 8: Performance in real conditions averaged over two pucks (a), zoomed in (b), and detailed for each puck in (c)

achieves the same performance with only 3 samples, reaching the average prediction error of 13.3cm with 16 samples. After 64 observations, the average error in [5] was 13.8 cm; with the proposed method, we were able to reach this value with only 7 samples. Thus, the trained dynamics model could be used to optimize a much better policy (e.g. by backpropagating through the learned environment model or by training an inverse model) than the policy trained in [5], and, if data is scarce, noticeably better than backpropagating through the LSTM model.

In Figure 9, we visualize the state space of the Kalman filter using PCA, similarly to previous FetchSlide analysis. We can also see that the real conditions we used for testing (blue and red) actually lie very close to each other, between medium and low friction (green and black), in the direction of anisotropic friction with lower value along y (pink). Based on this, we can say that the randomization range used in the simulator for generating the training data was excessive. We also suspect that the shift towards anisotropic effects is a result of unmodeled phenomena, such as the hockey blade slightly bending during the hit. This analysis can be used to search for a range of randomization parameters that encapsulate the real conditions with a much smaller margin of error (by adjusting the parameters given in Table II) Similarly, comparing cluster sizes between real and simulated conditions with varying amounts of noise can provide a more accurate measure about the noise level in the real domain. Such an analysis would provide a more accurate state prior for the dynamics model and could be used to train a model which achieves even smaller error in the low sample regime with the same training method.

We also observe that the hidden state space has learned friction representations which disentangle the magnitude and direction of friction—the low friction domain lies to the left, with the friction increasing along the first PC. Based on the position of the anisotropic friction domains, we can state that the second PC encodes the direction of friction—the domain with lower friction along x lies above the isotropic medium friction domain, while the domain with lower y friction lies below it. Thus, we observe that the method learns interpretable and useful dynamics representations.

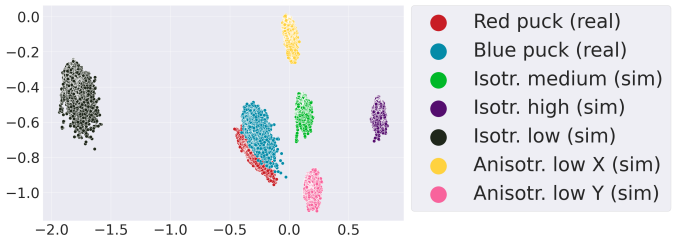


Fig. 9: PCA analysis of hidden states in Hockeypuck for Kalman filter-based adaptation, showing real and simulated domains after observing 20 samples.

V. Conclusions

In this paper, we have have presented a novel, uncertainty-aware meta learning algorithm and demonstrated that its performance on a model adaptation in the context of sim-to-real transfer. The proposed method, which directly takes into account uncertainty, performs better than both an LSTM-based approach (where the optimal way of handling uncertainty is learned) and than MAML (where the uncertainty information is discarded, and only the mean is taken into account). The difference between the proposed approach and the LSTM-based learner is visible mainly in the sim-to-real scenario, where both the data and the noise distribution does not exactly match what was seen during training, showing that the KF update rule is more robust to out-of-domain adaptation than the learned update rule of an LSTM. We also show that model adaptation, as opposed to policy adaptation, is a more data efficient approach—compared to a previous approach [5], all model-based methods offer superior performance. We also demonstrated how the proposed method learns an interpretable, disentangled space of dynamics representations.

We hypothesize that the additional benefit of using a Kalman filter-based approach lies in the way the uncertainty is explicitly handled in the inner loop update; the method used in the Kalman filter (which is Bayes-optimal for Gaussian noise) generalizes to non-Gaussian distributions better than the blackbox approach learned by the LSTM.

In our experiments, we assumed that the whole trajectory can be generated in advance and passed to the robot for execution. For some complex problems, however, it is necessary to use a feedback policy, which adds time complexity to the system. While the general idea behind our method would also be applicable to such problems, we leave the performance to be evaluated in the future.

Additionally, removing time complexity and using generative trajectory models gives us a natural way of exploring the environment through sampling from the latent distribution of trajectories. While we showed that this simple method can be sufficient, it is likely that a more informed search strategy (for example based on uncertainty information) would provide more informative samples, and thus adapt to real conditions with fewer samples. Finding proper search strategies is especially crucial in order to generalize the method for feedback policies, where there is no latent space of safe trajectories to sample from.

It is also interesting to see how uncertainty awareness translates into more general few-shot function estimation, e.g., in classification tasks involving label ambiguity.

References

[1] A. Hämäläinen, K. Arndt, A. Ghadirzadeh, and V. Kyrki, "Affordance learning for end-to-end visuomotor robot control," in *IROS*, 2019.

[2] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IROS*, 2017.

[3] OpenAI, "Learning dexterous in-hand manipulation," *IJRR*, vol. 39, 2020.

[4] M. Hazara and V. Kyrki, "Transferring generalizable motor primitives from simulation to real world," *IEEE RA-L*, vol. 4, 2019.

[5] K. Arndt, M. Hazara, A. Ghadirzadeh, and V. Kyrki, "Meta reinforcement learning for sim-to-real domain adaptation," in *ICRA*, 2019.

[6] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *ICLR*, 2019.

[7] J. Schmidhuber, "Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook," diploma thesis, Technische Universitat Munchen, Germany, 1987.

[8] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[9] M. Hazara and V. Kyrki, "Speeding up incremental learning using data efficient guided exploration," in *ICRA*, 2018.

[10] P. A. Ortega et al., "Meta-learning of sequential strategies," tech. rep., DeepMind, 2019.

[11] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, 1960.

[12] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel, "Backprop kf: Learning discriminative deterministic state estimators," in *NIPS*, 2016.

[13] P. Becker, H. Pandya, G. Gebhardt, C. Zhao, C. Taylor, and G. Neumann, "Recurrent kalman networks: factorized inference in high-dimensional deep feature spaces," in *ICML*, 2019.

[14] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *ICANN*, 2001.

[15] S. Flennerhag, A. A. Rusu, R. Pascanu, H. Yin, and R. Hadsell, "Meta-learning with warped gradient descent," in *ICLR*, 2020.

[16] B. Stadie, G. Yang, R. Houthooft, P. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever, "The importance of sampling in meta-reinforcement learning," in *NIPS*, 2018.

[17] C. Finn, K. Xu, and S. Levine, "Probabilistic model-agnostic meta-learning," in *NIPS*, 2018.

[18] J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn, "Bayesian model-agnostic meta-learning," in *NIPS*, 2018.

[19] J. Gordon, J. Bronskill, M. Bauer, S. Nowozin, and R. E. Turner, "Meta-learning probabilistic inference for prediction," in *ICLR*, 2019.

[20] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *RSS*, 2018.

[21] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2real view invariant visual servoing by recurrent control," in *CVPR*, 2018.

[22] X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan, "Rapidly adaptable legged robots via evolutionary meta-learning," in *IROS*, 2020.

[23] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *NIPS*, 2018.

[24] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "Promp: Proximal meta-policy search," in *ICLR*, 2019.

[25] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *CoRL*, 2018.

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

[27] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," in *IROS*, 2017.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[29] C. Finn and S. Levine, "Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm," in *ICLR*, 2018.

[30] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.