Proceedings of Machine Learning Research vol 144:1–12, 2021

Domain Adaptation Using System Invariant Dynamics Models

Sean J. Wang SJW2@ANDREW.CMU.EDU
Aaron M. Johnson AMJ1@ANDREW.CMU.EDU
Department of Mechanical Engineering, Carnegie Mellon University

Abstract

Reinforcement learning requires large amounts of training data. For many systems, especially mobile robots, collecting this training data can be expensive and time consuming. We propose a novel domain adaptation method to reduce the amount of training data needed for model-based reinforcement learning methods to train policies for a target system. Using our method, the required amount of target system training data can be reduced by collecting data on a proxy system with similar, but not identical, dynamics on which training data is cheaper to collect. Our method models the underlying dynamics shared between the two systems using a System Invariant Dynamics Model (SIDM), and models each system's relationship to the SIDM using encoders and decoders. When only limited amounts of target system training data is available, using target and proxy data to train the SIDM, encoders, and decoders can lead to more accurate dynamics models for the target system than using target system data alone. We demonstrate this approach using simulated wheeled robots driving over rough terrain, varying dynamics parameters between the target and proxy system, and find a reduction of 5-20x in the amount of data needed for these systems.

Keywords: Domain Adaptation, Sim2Real, Model-Based Reinforcement Learning

1. Introduction

Wheeled mobile robots have demonstrated their value in a variety of indoor applications such as vacuuming floors, warehouse logistics, and hospital delivery. Although there has been an effort to improve the mobility of these robots, they still struggle in environments that are not relatively flat and smooth, such as curbs or rocks, as well as in loose terrain, such as gravel, mud, or snow. Autonomous driving over rough terrain is difficult due to our inability to model the system dynamics exactly. Contact forces between the wheel and the terrain are difficult to model and parameters such as inertia, joint friction, motor and battery performance may not be known precisely. This results in poor performance in model dependent autonomy such as trajectory optimization or path planning.

Alternatively, data-driven approaches can be used to interpret collected motion data and create control policies that perform well on the actual system. Reinforcement learning has shown promising results for creating control policies for complex systems (Levine et al., 2018). Despite these recent advances, learning control policies still remains difficult for many systems. Generating well performing control policies using reinforcement learning algorithms often requires many robot trials to collect large sets of training data. This is true even for more sample efficient reinforcement algorithms such as off-policy methods (Munos et al., 2016) or model-based methods (Atkeson and Santamaria, 1997; Kocijan et al., 2004). For computationally intensive simulated systems, running enough robot trials can be slow and expensive. On real world systems, the process is even more time consuming and also likely to lead to extensive wear of components and unrecoverable robot states. This requires a human operators to stand by to reset or repair the robot whenever failure occurs.

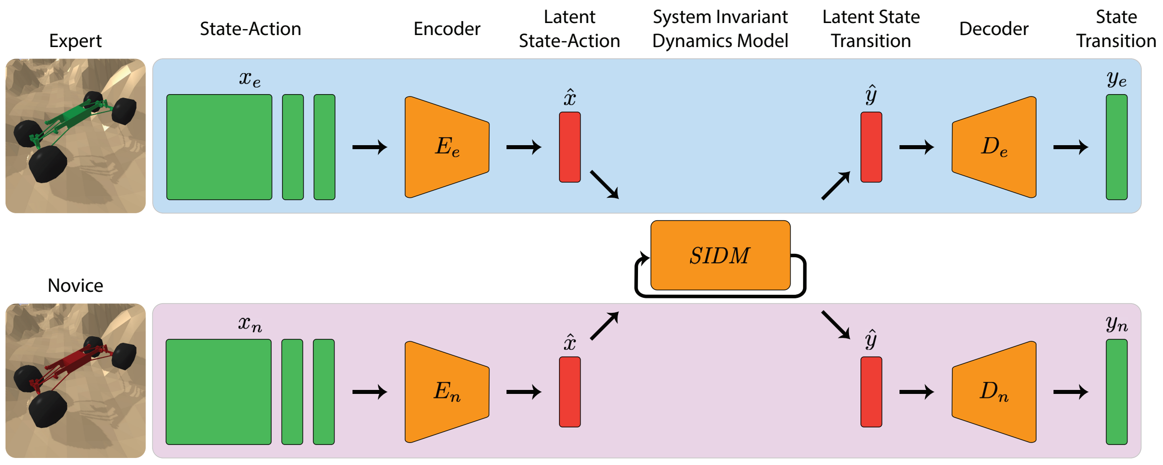

Figure 1: The prediction pathway for the expert is shown on the top and the prediction pathway for the novice is shown on the bottom. The expert and novice share a system invariant dynamics model (SIDM) which models the underlying dynamics shared by the systems. Each system has an encoder and decoder to model their relationship to the SIDM.

In this paper, we aim to make reinforcement learning more practical for a target system (novice), by leveraging a similar proxy systems (expert) on which trials are cheaper to perform. We propose a novel domain adaptation method, summarized in Figure 1, that allows model-based reinforcement learning algorithms to use the large expert dataset to learn more accurate novice dynamics models when novice data is limited. We introduce a System Invariant Dynamnics Model (SIDM) that models underlying dynamics shared between the expert and novice. We provide an algorithm to train the SIDM as well as learn the relationship each particular system has to it. We show through experimentation that using novice and expert data to learn a SIDM and the novice's relation to it results in more accurate predictions for the novice compared to if a dynamics model was trained using solely novice data. Using SIDM has a few advantages over previous domain adaptation methods. (1) It does not require novice policies (for the desired or proxy task) to be known a priori. (2) It allows for high variance in dynamics between the expert and the novice.

2. Related Work

Model Based Reinforcement Learning In model-based reinforcement learning (Atkeson and Santamaria, 1997; Kocijan et al., 2004), a predictive model of the robot's dynamics is learned. This dynamics model is then used to generate control policies using methods such as the cross-entropy method (CEM) of optimization (Botev et al., 2013). Model-based reinforcement learning methods are appealing due to their sample efficiency as well as ability to generalize to new tasks due to the reward independent nature of the dynamics model. However, model-based reinforcement learning methods are sensitive to the prediction accuracy of the dynamics model due to the compounding nature of their error. In this paper, we focus on using expert training data to improve model prediction accuracy for a novice system in cases where novice training data is limited.

Domain Adaptation In domain adaptation, learning is performed on a expert robot and knowledge is transferred to a novice robot. One common approach for domain adaptation is in Sim2Real, where the goal is to transfer knowledge learned in simulation to the real robot. The simplest approach to the Sim2Real problem is system identification, where parameters of the simulation are fine tuned to closely match the real robot's dynamics. Another Sim2Real approach is domain randomization (Tobin et al., 2017; Sadeghi and Levine, 2017), where training is performed on multiple simulations with randomized parameters. Domain randomization allows for robust policies that perform well in a wide range of domains, hopefully including the desired real world domain. Alternatively, training could be performed on a single expert domain, and the relationship of the novice domain to the expert domain could be learned. Bousmalis et al. (2018) restylizes images from a novice domain to make them more similar to images from the expert domain. Golemo et al. (2018) trains a network to transform predictions for the expert to true predictions for the novice. Since these methods train the base policy or dynamics model to be specific to the expert, the novice can only vary slightly from the expert dynamics.

Another approach is to use autoencoders to form system invariant latent representations of expert and novice states. Gupta et al. (2017) focus on transferring new policies from the expert to the novice. A new novice policy is trained to output similar states as the known expert policy, using the latent state representation for state similarity comparisons. Kim et al. (2020) focus on corresponding expert and novice policies by learning the forward dynamics of the policies in the latent state space, with the hope that this correspondence knowledge can be used to accelerate policy training on the novice. One challenge to learning system invariant representations of states is ensuring that latent state representations from different systems are similar. Gupta et al. (2017) uses Dynamic Time Warping (Müller, 2007) to find corresponding states from novice policies and expert policies trained for proxy tasks. The encoder is trained to maximize similarity of the corresponding states' latent representation. Unfortunately, this method requires a priori knowledge of similar proxy task policies on the novice and the expert. Kim et al. (2020) use Iterative Closet Point (ICP) (Besl and McKay, 1992) to correspond the shape of the latent state space for each batch given similar novice and expert policies. Using ICP to match the latent space requires large batch sizes during training, making computation difficult especially in memory limited cases. Kim et al. (2020) used a batch size of 4096 during their experiments.

The method proposed in this paper is similar to Gupta et al. (2017) and Kim et al. (2020). We use system invariant latent spaces to correspond systems. However, instead of transferring policies, we focus on transferring models of the system's dynamics. We use a system invariant latent space to represent not only states, but also actions. We also use another system invariant latent space to represent state transitions. We ensure that latent state-actions from the expert and novice are similar by using an adversarial network for similarity comparison. A system invariant dynamics model is learned to predict latent state transitions given latent state-actions. This method allows model-based reinforcement learning algorithms to leverage large expert datasets to learn more accurate novice dynamics models when only small novice datasets are available.

3. Method

Our method assumes that similar systems share some underlying dynamics, as has been observed for many classes of dynamical tasks such as running and climbing (Full and Koditschek, 1999). We model the underlying dynamics of the systems with a System Invariant Dynamics Model (SIDM).

The inputs of the SIDM are system invariant representations of the robot state-action tuples. The output of the SIDM are predicted state transitions represented in a system invariant form. For each particular system, an encoder is used to transform the system specific state-action tuples to inputs for the SIDM. Similarly, each system has a decoder that transforms predictions from the SIDM to system specific state transition predictions. Our method trains an encoder and decoder specific to each system, and an SIDM shared between all systems. This prediction structure is shown in Figure 1. We use an adversarial network to force SIDM inputs and outputs to be similar between systems, and thus system invariant.

The encoder and decoder of each system can be modeled with relatively small networks, so they are less susceptible to overfitting. Although the SIDM network is required to be larger for precise predictions, it is trained using a larger dataset consisting of data from all systems. As a result, the novice prediction network graph, which consists of the novice encoder, novice decoder, and SIDM, can be trained for high accuracy before it becomes overfit.

3.1. Notation

We denote the expert and novice state-action spaces, which contain state-actions pairs for the expert or novice system, respectively, as X_e and X_n. We denote the state-transition spaces, which contains state transition vectors for the expert or novice system, as Y_e and Y_n. We also define two latent spaces. The latent state-action space, denoted as $\hat{X}$, contains system invariant representations of state-action pairs. The latent state-transition space, denoted as $\hat{Y}$, contains system invariant representations of state transitions.

We define an encoder that maps the state-action space to the latent state-action space of the expert or novice as $E_e : X_e \to \hat{X}$ and $E_n : X_n \to \hat{X}$. Similarly, we define a decoder that maps the latent state-transition space to the state-transition space for the expert or novice as $D_e : \hat{Y} \to Y_e$ and $D_n : \hat{Y} \to Y_n$. The system invariant dynamics model predicts resulting latent state transitions given latent state-actions and is denoted as $SIDM : \hat{X} \to \hat{Y}$.

We denote $\mathcal{D}_e = \{(x_e^{(i)}, y_e^{(i)})\}_{i=1}^N$ as the expert dataset containing N samples, where $x_e^{(i)} \in X_e$ is a state-action pair resulting in the state transition $y_e^{(i)} \in Y_e$. Likewise, we denote $\mathcal{D}_n = \{(x_n^{(i)}, y_n^{(i)})\}_{i=1}^M, x_n \in X_n, y_n \in Y_n$, as the novice dataset contains M samples. We assume that $M << N$. For simplicity of notation, we drop the e or n subscript denoting which system an element or function belongs to when it is clear from context.

3.2. Training Procedure

The training approach of our method follows closely to that of Adversarial Autoencoders (AAE) (Makhzani et al., 2015). For AAE, an autoencoder is trained with the two objectives of minimizing reconstruction loss and matching the aggregated posterior distribution of latent representations to an arbitrary prior distribution. For the second training objective, an adversarial training criterion is used to match generated samples (latent representations from the encoder) to the true samples (sampled from the prior distribution). As our application focus is on state transition prediction instead of reconstruction, our approach differs from AAE in two ways: (1) Instead of reconstructing the input and minimizing reconstruction error, our whole network predicts state transitions of the robot and we minimize prediction error. (2) We divide the single latent representation from AAE into two latent representations: a latent state-action representation and a latent state transition representation. We add a SIDM network to predict latent state transitions given latent state-actions.

Algorithm 1: Prediction Training

```
for number of prediction training iterations do
    $g \leftarrow 0$
    for prediction batch size ($b_{pred}$) do
        if randomly chosen to update expert with probability $p_e$ then
            $(x, y) \leftarrow$ random sample from expert dataset $\mathcal{D}_e$
            $y' \leftarrow D_e \circ SIDM \circ E_e(x)$
        else
            $(x, y) \leftarrow$ random sample from novice dataset $\mathcal{D}_n$
            $y' \leftarrow D_n \circ SIDM \circ E_n(x)$
        end
        $g \leftarrow g + \frac{1}{b_{pred}} \nabla_{E_e, E_n, SIDM, D_e, D_n} \mathcal{L}_{pred}(y', y)$
    end
    $E_e, E_n, SIDM, D_e, D_n \leftarrow$ update parameters using gradient $g$
end
```

The training procedure for our method alternates between two phases – the prediction phase, and the regularization phase. The prediction phase aims to increase prediction accuracy for both the expert and novice system. The regularization phase aims to match the distribution of latent state-actions over novice state-actions to the distribution of latent state-actions over expert state-actions. We only match the distribution of latent state-actions, since the SIDM network should map similarly distributed latent state-actions to similarly distributed latent state transitions. Both the prediction training and regularization phase can be executed on minibatches using stochastic optimization methods such as SGD or Adagrad (Duchi et al., 2011).

During the prediction training phase, samples are randomly chosen from either the expert dataset with probability p_e or the novice dataset with probability $1 - p_e$. For each sample $(x^{(i)}, y^{(i)})$, the appropriate encoder, either expert or novice, is used to map $x^{(i)}$ to a latent state-action $\hat{x}^{(i)}$. Given $\hat{x}^{(i)}$, the SIDM predicts a latent state-transition $\hat{y}^{(i)}$. Finally, the appropriate decoder is used to map $\hat{y}^{(i)}$ to a predicted state transitions $y'^{(i)}$. For the prediction phase, the loss function compares the predicted state transitions to true state transitions using the squared second norm difference,

$$\mathcal{L}_{pred}(y', y) = \|y' - y\|^2 \tag{1}$$

The procedure for the prediction training phase is summarized in Algorithm 1.

The regularization phase follows the Generative Adversarial Networks (GAN) framework (Goodfellow et al., 2014). An adversarial network (discriminator) is used to distinguish between latent state-actions from the expert encoder and latent state-actions from the novice encoder. The sigmoid output of the discriminator, $A(\hat{x})$, predicts the probability that a latent state-action came from the expert instead of the novice. At the start of the regularization phase, the adversarial network is trained to minimize expected negative log-likelihood,

$$\min_A \mathbb{E}_{x_e \sim \mathcal{D}_e, x_n \sim \mathcal{D}_n}[\mathcal{L}_{disc}(E_e(x_e), E_n(x_n), A)] \tag{2}$$

$$\mathcal{L}_{disc}(\hat{x}_e, \hat{x}_n, A) = -\log A(\hat{x}_e) - \log(1 - A(\hat{x}_n)) \tag{3}$$

Algorithm 2: Regularization Training

```
for number of discriminator training iterations do
    $g \leftarrow 0$
    for discriminator batch size ($b_d$) do
        $x_e, x_n \leftarrow$ random sample from expert and novice datasets $\mathcal{D}_n, \mathcal{D}_e$ respectively
        $\hat{x}_e, \hat{x}_n \leftarrow E_e(x_e), E_n(x_n)$ respectively
        $g \leftarrow g + \frac{1}{b_d}\nabla_A \mathcal{L}_{disc}(\hat{x}_e, \hat{x}_n, A)$
    end
    $A \leftarrow$ update parameters using gradient $g$
end
for number of distribution matching iterations do
    $g \leftarrow 0$
    for distribution matching batch size ($b_g$) do
        $x_n \leftarrow$ random sample from novice datasets $\mathcal{D}_n$
        $\hat{x}_n \leftarrow E_n(x_n)$
        $g \leftarrow g + \nabla_{E_n} \mathcal{L}_{gen}(\hat{x}_n, A)$
    end
    $E_n \leftarrow$ update parameters using gradient $g$
end
```

After updating the weights of the discriminator, the novice encoder (generator) is trained to minimize the negative log-likelihood of latent state-action of the novice coming from the expert.

$$\min_{E_n} \mathbb{E}_{x_n \sim \mathcal{D}_n}[\mathcal{L}_{gen}(E_n(x_n), A)] \tag{4}$$

$$\mathcal{L}_{gen}(\hat{x}_n, A) = -\log(A(\hat{x}_n)) \tag{5}$$

The procedure for the regularization training phase is summarized in Algorithm 2.

In this formulation, all mappings are deterministic. Alternatively, an approach similar to that taken by Kingma and Welling (2014) could be used, where mappings are nondeterministic. In this alternate approach, the encoders, decoders, and SIDM would estimate posterior distributions for the latent state-actions, latent state transitions, and system specific state transitions. Latent state-actions and latent state transitions would be randomly sampled from the estimated distributions, and negative log-likelihood of the true state transition given estimated posterior distribution from the decoder would be used as the prediction phase loss function.

4. Experiment Setup

We demonstrate the value of using SIDM by training dynamics models for novice systems with and without knowledge transfer from expert systems using SIDM. The improvement in motion prediction accuracy for the novice is compared over varying amounts of novice training data. The expert and novice systems used were both wheeled robots driving over randomly generated rough terrain simulated with PyBullet (Coumans and Bai, 2016–2019), as shown in Figure 2. The robot actions corresponded to desired wheel velocity and steering angle applied using PD controllers. The expert and novice had different scaling of actions, contact friction, link masses, link inertia, suspension heights, suspension limits, spring constants, and damping constants.

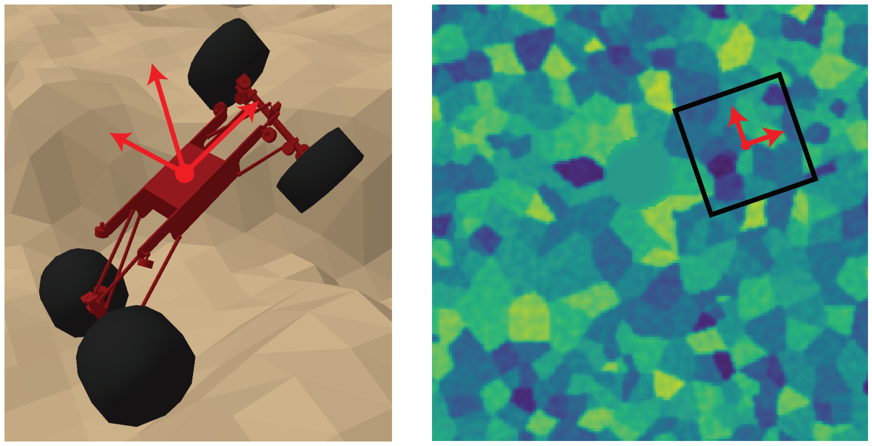

Figure 2: Left shows the simulation environment used for experiments. Right shows the world terrain height map with the robot centered map outlined in black. The robot frame is shown in red. State-Actions for prediction included the robot body velocity, gravity vector relative to the robot frame, robot centered terrain map, and action (throttle, steering). Predicted state transitions included the new robot frame relative to the previous, and the new robot body velocity.

The state-action input (x) to the dynamics models included the robot's tilt (represented as the gravity vector relative to the robot frame), body velocity (represented as a twist), terrain height map centered about the robot, and action taken. The robot centered terrain height maps were generated from global terrain height maps, as shown in Figure 2. We find it realistic to assume the robot has knowledge of the global terrain height map due to existing methods for generating these maps from raw sensor data (Fankhauser et al., 2014, 2018). Alternatively, SIDM could also be used in other approaches where state transitions are predicted directly from raw depth sensor data (Kahn et al., 2021). The dynamics model's predicted state transition (y) consists of the relative movement of the robot's body (represented as a translation and quaternion orientation vector) and the new body velocity of the robot. In total, the state-action tuple consisted of a 11 dimension vector and a 300 by 300 matrix (terrain map) while the state transition vector had a size of 13.

Motion data was collected using random actions and PyTorch (Paszke et al., 2019) was used to train dynamics models. During training, 75 percent of the novice data was used for training and 25 percent of the data was used for testing. We stopped training once the test prediction loss started increasing indicating overfitting of the dynamics model. We then validated the prediction loss of each trained model using a separate novice validation dataset.

When training without SIDM, the prediction phase only sampled data from the novice dataset ($p_e = 0$) and the regularization phase was ignored. To keep the comparison consistent, the same novice dynamics model network structure was used when training without SIDM, and the expert encoder and decoder were simply removed. Anecdotally, we found that changing the network structure when training without SIDM led to minimal change in prediction loss.

Each encoder network consisted of a two layer Convolutional Neural Network (CNN) (Krizhevsky et al., 2017) to process the terrain height map, consisting of an 8 channel layer and a 4 channel layer, both with a kernel size of 4. All CNN layers used no padding, and a stride of 1. The output of the CNN was reshaped into a vector, concatenated with the rest of the state and action input, and fed into a single hidden fully connected (FC) layer and an output FC layer, both of size 128 (latent state-action vector, $\hat{x}$). Each SIDM network consisted of four Long-Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) layers, each with a hidden state size of 1024. The output of the LSTM layers was fed into a FC output layer of size 64 (latent state transition vector, $\hat{y}$). Each decoder network consisted of a single hidden FC layer of size 64, and an output FC layer of size 13. All CNN and FC layers used leaky ReLu except for the output FC layer of the decoder which did not have an activation function. Adam (Kingma and Ba, 2015) with weight decay (Loshchilov and Hutter, 2018) was used to optimize the neural network parameters.

It is important to use an appropriate value for p_e, otherwise the SIDM network will be biased towards a specific system and not be system invariant. Low p_e values cause training of the SIDM network to be biased towards the novice and not take advantage of the large expert dataset. High p_e values cause the SIDM network to be biased towards the expert and lead to poor performance on the novice. In cases where the novice dataset is much smaller than the expert dataset, choosing a p_e value proportional to the dataset sizes will cause p_e to be too high. In our experiments, we use a p_e value of 0.5 when the dynamics of the novice and expert differed greatly, and a p_e value of 0.75 when the novice and expert were more similar. For the same reason, the SIDM network cannot be pretrained using just expert data, although we believe the SIDM could be pretrained using multiple expert systems.

It is also important not to train in any one phase for too many iterations. Since the prediction and regularization training phases update the weights of the novice encoder for different loss functions, training in one phase for too long led to high losses for the other phase and unstable training. In our experiments, we set the number of prediction iterations to 100, and the number of distribution matching iterations to 20.

5. Results

To show the accuracy improvements using SIDM, we ran training experiments with varying amount of novice data on two different expert and novice systems. In the first system the expert dynamics were more similar to the novice dynamics, while in the second system the expert dynamics greatly varied from the novice dynamics. The difference in the systems was most pronounced in the suspension parameters. In the first system, the suspension of the novice was 5 times stiffer, had 2 times more damping, but had the same amount of travel compared to that of the expert. In the second system, the suspension of the novice was 667 times stiffer, had 100 times more damping, and about 1.5 times less travel compared to that of the expert, leading to an almost completely rigid suspension[1]. For both, we collected about 14 simulation hours worth of motion data for the expert system. For the first system we varied the novice dataset size between 1 minute and 2 hours worth of motion data. For the second system, we increased this range so that the novice dataset ranged between 1 minute and 10 hours worth of motion data, as the improvements using SIDM did not converge to the non-SIDM system by 2 hours. For each system and novice dataset size, we ran 10 trials, for a total of 320 trials overall. Figure 3 shows the validation losses for these experiments.

In addition to the prediction validation loss, we also tested the improvement using SIDM by simulating 25 step (2.5 simulation second) trajectories for the novice robot (from the first system).

1. Simulation code and system parameters can be found at https://github.com/robomechanics/SIDM.

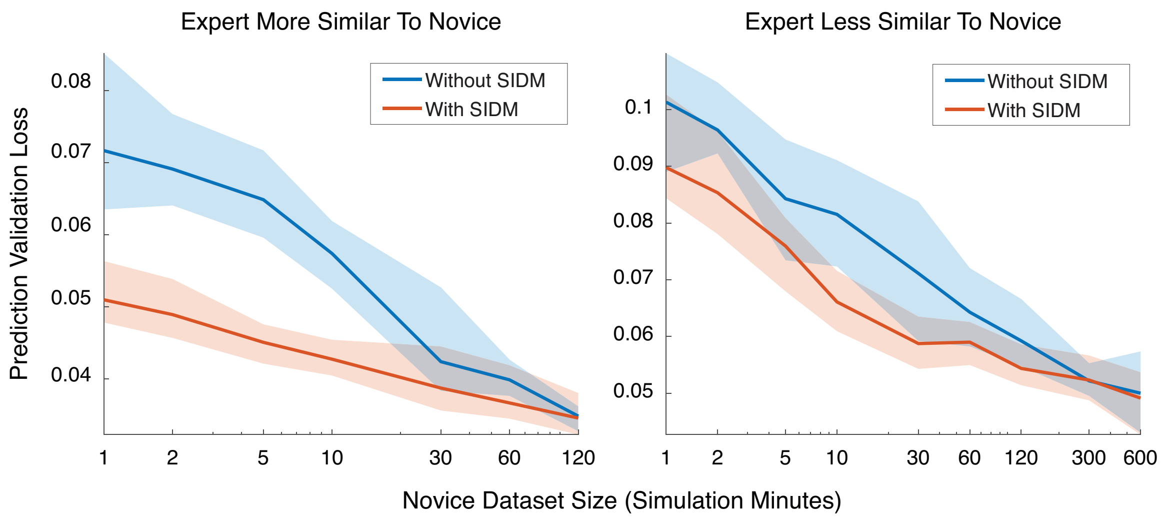

Figure 3: Dynamics models were trained for two different novice systems with and without knowledge transfer from expert systems using SIDM. On the left, the dynamics of the expert were more similar to the dynamics of the novice. On the right, the dynamics of the expert were less similar to the dynamics of the novice. The line shows the mean validation loss for 10 trials, while the shading shows the range between the minimum and maximum.

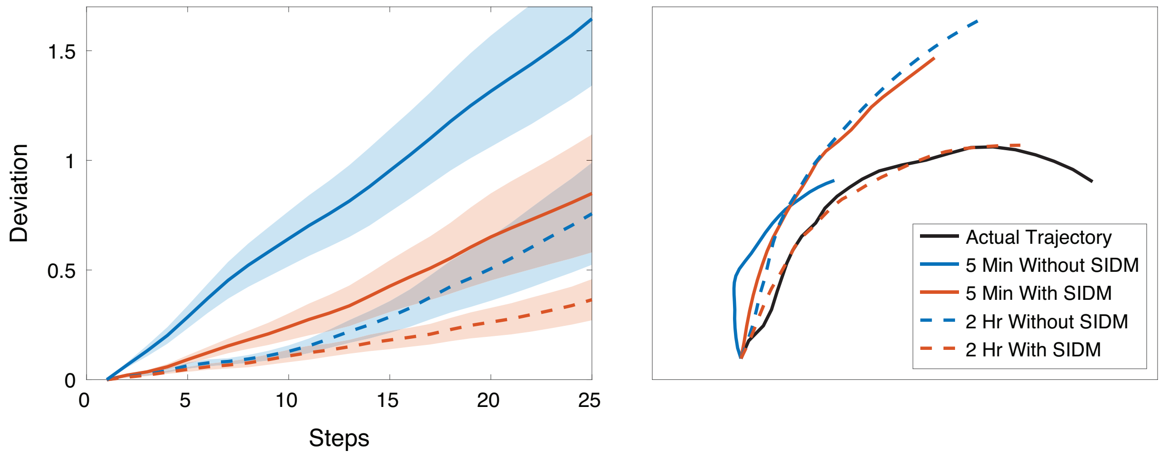

Figure 4: The robot's trajectory was predicted for 25 steps (2.5 seconds) using the trained dynamics models. The left figure shows the deviation of the predicted trajectory from the actual trajectory. Each line shows the mean and the shaded region shows the standard error over 10 trials. The right figure shows the actual trajectory and predicted trajectories for one example trial.

For each trajectory, an action was randomly selected and applied over the length of the trajectory. We use the learned novice dynamics models to predict the resulting trajectories over rough terrain given the action taken. We integrate the predicted state transitions to predicted the robot's absolute position and orientation over the course of the trajectory. At each time step, the predicted absolute position and orientation was used to generate a new robot centered terrain height map for the input of the next prediction. Figure 4 shows the results comparing the actual trajectory of the robot to the predicted trajectory of the robot using 5 minutes and 2 hour of novice training data both with and without SIDM.

Using the SIDM to transfer knowledge from the expert system to the novice system resulted in a significant improvement in novice predictions especially when the novice dataset was small. When the expert and novice were more similar, training using SIDM resulted in similar performance to training with up to 20 times the amount of data as training without SIDM. When the expert and novice were less similar, training using SIDM resulted in similar performance to training with about 5 times the amount of data as training without SIDM. As expected, when the novice dataset becomes larger the performance with and without SIDM become comparable (at about 2 hours and 5 hours of novice training data for the two systems, respectively). Although using SIDM does not increase overfitting as the dataset becomes larger, it does make the network harder to train. We recommend reducing expert train probability, discriminator training iterations, and distribution matching iterations as the novice dataset grows. Doing so will effectively cause training to be done only on the novice system data once the benefits of SIDM are expected to be minimal.

6. Conclusion

In this paper, we propose a new method of domain adaptation that allows model-based reinforcement learning algorithms to learn more accurate dynamics models for a novice system by leveraging motion data collected on expert system that have different, though similar, dynamics. This method uses what we call a System Invariant Dynamics Model (SIDM) to model the underlying dynamics shared between the two systems. The SIDM is trained using motion data from both the expert and the novice. We provide an algorithm to train the SIDM as well as an encoder and decoder for each system that describe each system's relationship to the SIDM. We show through experimentation that training dynamics models for the novice using SIDM and expert data leads to more accurate novice dynamics models especially in cases where motion data for the novice is limited.

In the future, we believe that further improvements could be made by pretraining the SIDM with multiple experts. Pretraining with multiple experts may lead to a more accurate and generalizable SIDM that is truly system invariant. It may also lead to faster training times for the novice as the SIDM is trained a priori and only the novice encoder and decoder need to be learned during novice training. We also believe that this method could be adapted to learn system invariant policies instead of dynamics models for model-free reinforcement learning algorithms. Finally, we plan to implement this approach to train a real-world novice robot with relatively little data by leveraging a larger simulated expert dataset.

References

Christopher G Atkeson and Juan Carlos Santamaria. A comparison of direct and model-based reinforcement learning. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3557–3564, 1997.

Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier, 2013.

Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *IEEE International Conference on Robotics and Automation*, pages 4243–4250, 2018.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Péter Fankhauser, Michael Bloesch, Christian Gehring, Marco Hutter, and Roland Siegwart. Robot-centric elevation mapping with uncertainty estimates. In *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3019–3026, 2018. doi: 10.1109/LRA.2018.2849506.

Robert J Full and Daniel E Koditschek. Templates and anchors: neuromechanical hypotheses of legged locomotion on land. *Journal of experimental biology*, 202(23):3325–3332, 1999.

Florian Golemo, Adrien Ali Taiga, Aaron Courville, and Pierre-Yves Oudeyer. Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pages 817–828, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Abhishek Gupta, Coline Devin, Yuxuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Gregory Kahn, Pieter Abbeel, and Sergey Levine. Badgr: An autonomous self-supervised learning-based navigation system. *IEEE Robotics and Automation Letters*, 6(2):1312–1319, 2021.

Nam Hee Kim, Zhaoming Xie, and Michiel Panne. Learning to correspond dynamical systems. In *Learning for Dynamics and Control*, pages 105–117. PMLR, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE, 2004.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4–5):421–436, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017. doi: 10.15607/RSS.2017.XIII.034.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 23–30, 2017.