# Accelerated Robot Skill Acquisition by Reinforcement Learning-Aided Sim-to-Real Domain Adaptation

Zvezdan Lončarević[1,2], Aleš Ude[1] and Andrej Gams[1]

***Abstract*— Since robot learning takes a lot of time and might be too time-consuming to perform on real robots, the learning process is often done in simulation. Bridging the sim-to-real gap, however, still requires real-world effort that might impose severe limitations due the still large number of the required iterations. In this paper we explore how to accelerate real-world skill learning with sim-to-real transfer learning. We propose a new methodology based on dimensionality reduction with deep autoencoders, followed by domain adaptation. The latter combines re-training of the complete deep neural network with real data and adaptation of a part of the network parameters with reinforcement learning. The results of our experiments demonstrate that a considerable acceleration of real-world learning is achieved by approach when learning complex robot skills.**

## I. INTRODUCTION

Practice makes perfect, thus repeated performance of the desired task is used also in robot learning to acquire new skills. Adaptation of robot actions through robot (deep) learning is one of the key technological enablers for robot applications in unstructured environments, and has been recently ever-more adopted for control of complex robot systems [1], multi-robot systems [2], and robot tasks in the form of end-to-end perception-to-control policies [3]. However, repeated task executions, i.e. training or learning, can be exceedingly expensive in terms of the required time and effort and in terms of the risk of damage for the robot and its immediate environment [4].

Fast, sample-efficient robot learning has thus been the focus of many research efforts [5], with several directions. One of the directions is in the reduction of the search space. Examples include constraining the search space between over- and under- execution [6] or reduction of the search space dimensionality. The latter can implemented through principal component analysis or (deep) autoencoder networks [7], [8]. One of the recently most applied methodologies is with transfer learning, where the task/behavior is learned in simulation and then transferred to the real-world (sim-to-real transfer learning) [9]–[11]. Success of such methods was for example demonstrated in quadruped-robot walking in [12].

As noted in a recent survey [11], several methodologies for sim-to-real transfer are commonly applied. These include, among others, zero-shot transfer, domain randomization and domain adaptation. Zero-shot transfer relies on very good matching of simulated and real-world behavior [13], often obtained through system identification [14]. Domain randomization can also be considered a zero-shot methodology, but it relies on highly randomized simulation, so that it covers the actual distribution of the real-world data [15]. It can be further split into visual and dynamics randomization. An example of dynamics randomization was shown through randomizing system physical properties for in-hand manipulation in [16].

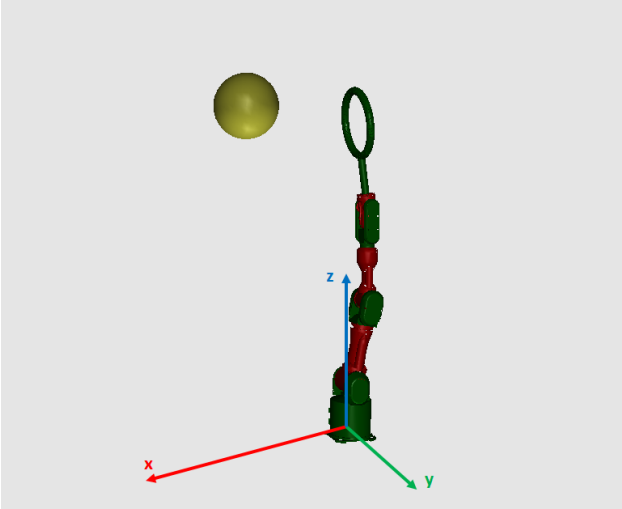


Fig. 1: Experimental setup - Mitsubishi PA10 robot in MuJoCo simulation.

In domain adaptation, data acquired in source domain are used to advance the behavior of a learned model in a different, target domain, where data is less available. Often it is applied in tasks where vision related tasks provide a priors to the subsequent reinforcement learning (RL) of tasks [17], [18]. Domain adaptation for speeding-up of real-world behavior was demonstrated for grasping in [18], showing up to 50-fold reduction of required real-world samples.

As noted above, domain adaptation is often applied in combination with RL. RL has been extensively applied in robotics, and has shown remarkable success [19], [20], but can suffer from sample inefficiency and the curse of dimensionality [21]. Therefore, while vision-to-motion actions might require extensive amounts of data to provide a low-dimensional robot policy, RL can effectively be applied to fine-tune the policy for the given task parameter [1].

### A. Contribution

In this paper we show the possibility of improving the performance of transfer learning through RL.

In transfer learning, the performance in target domain is increased by mixing a lot of data in the source domain with

[1]Humanoid and Cognitive Robotics Lab, Dept. of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia. E-mail: {zvezdan.loncarevic, ales.ude, andrej.gams}@ijs.si.

[2]Jožef Stefan International Postgraduate School, Ljubljana, Slovenia.

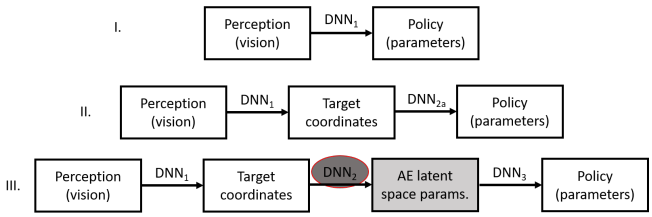


Fig. 2: Decomposition of the end-to-end task into sub-tasks. The main focus of the paper is on $\text{DNN}_2$.

small amount of data in the target domain. For example when training a neural network, one typically freezes a large part of the network and then modifies just part of it, for example one layer. Thus one can utilize the already learned feature extraction of a given, trained network [11]. Additionally, one can then unfreeze the network and do fine tuning. Given that only a (small) part of the network is changed, which will only have a small number of parameters, we propose to combine both RL and re-training to learn the values of these network parameters. Consequently, the proposed approach reduces the number of required real-world samples and accelerates real-world task learning. We demonstrate the approach on robot throwing, where we transfer the knowledge from purely kinematic simulation to dynamic simulation.

In the next Section we first outline the overall approach. This is followed by an explanation of simulated data acquisition and dimensionality reduction in Section III, RL-aided transfer learning in Section IV and experimental results in Section V. Section VI provides a discussion and Section VII the conclusions.

## II. Task Learning

The focus of this paper and the approach is highlighted in Fig. 2. The following explanation of the approach is for clarity adjusted to the example of throwing a ball at a target.

End-to-end, the robot should somehow perceive the location of the target, for example with vision, and generate policy parameters that define this trajectory. The subsequent execution can be relegated to the low-level controller. In our case, the motion policy is encoded in the form of dynamic movement primitives (DMPs) [22], so an end-to-end solution would take the image as the input and provide the DMP parameters as the output.

The end-to-end solution can be deconstructed into several sub-tasks, as shown in the second layer of Fig. 2. A dedicated deep neural network (DNN) can be used to extract the coordinates of the target for throwing, for example in the manner described in [23]. These coordinates can then be used as the input into a second DNN that maps the target coordinates to the policy parameters. Because the number of policy parameters that have to be set does matter, as was shown in [7], the task can be further deconstructed to include a dimensionality reduction layer.

Depicted in the third layer of Fig. 2, the approach has three separate sub-aspects. First, we can use a dedicated vision-to-coordinates deep neural network. This is not the focus of this paper. Skipping the second step, we introduce a third step that maps latent space parameters of a deep autoencoder onto DMP parameters. The autoencoder is trained on a huge amount of synthetic data, synthesized in a manner that covers most expected robot motions. Note that the autoencoder works on DMP parameter data, and thus doesn't need any real-world execution data. Thus, we are now only left with the second step, which maps the target coordinates to the latent-space parameters of a deep autoencoder. Again, a dedicated DNN can be used for this mapping. Accelerated learning of the dedicated DNN (in this case $\text{DNN}_2$) for mapping between the throwing target coordinates and the deep autoencoder latent space parameters is the main focus of this paper,

$$\text{DNN}_2 : \boldsymbol{T} \mapsto \boldsymbol{\theta}^l. \tag{1}$$

Here $\boldsymbol{T}$ indicates the coordinates of the target and $\boldsymbol{\theta}^l$ indicate the AE latent space parameters. Separate steps of the process and acquisition of $\boldsymbol{T}$ and $\boldsymbol{\theta}^l$ are detailed in the next Sections.

## III. Dimensionality reduction

Dimensionality reduction, such as with PCA or deep autoencoders, requires data to extract the principal components or to train the deep autoencoder network. However, obtaining this data with the real robot (or in the target domain) is too time consuming.

### A. Data acquisition

Let's look at using a deep AE network to reduce the data of motion that is encoded in the form of the DMP. DMPs provide reference data for the controller to execute. This reference data is encoded in the form of a set of weights (typically for each degree of freedom), starting points and timing. These parameters, thus, have no physical meaning, because they provide the reference for the low-level controller only through the time-evolution of the DMPs (execution). Therefore, we can easily synthesize large amounts of data, use it to train a deep AE network, and use the dimensionally reduced AE parameters. Now, these provide – through the decoder part of the deep AE network – the reference for the low-level controller.

While the data for training of the deep AE network can provide any data, for example of purely random motor babbling, it makes sense that it is similar to the actual desired behavior. In the case of throwing, this would constitute throwing trajectories. One advantage of this is that the AE can then only provide results that are similar to the training data - and not motor babbling.

In order to have a good initial approximation of the movement, we created a kinematic database of robot-executable throws with mathematical modelling. The calculations are the same as in [7]. A brief explanation is provided in the following for completeness of the paper.

We determined a set of targets that are within a range of our throwing robot. Then, in a two step process, we first calculate the robot joint configuration where the smallest joint velocity is needed to hit the target. We used the computed release joint position and velocity values to create

the throwing trajectory, encoded as a fifth degree polynomial. Finally, we calculated the release and end times of the trajectories. These trajectories were encoded as DMPs, which were used to train the deep AE network. For more details on the database acquisition we refer the reader to [7] and [24].

### B. Deep Autoencoders

Deep autoencoder neural networks are often used for dimensionality reduction. When training the AE network, it learns to copy its input data to the output with the highest possible precision, but data is pushed through different sized layers. The AEs consist of two parts: an encoder and a decoder part. The encoder part is used to push the input data through the network until the bottleneck is reached. This bottleneck is called latent space and it extracts the most important features of the data. The second (decoder) part is used to decode the data from the latent space to original representation with minimal loss of information. In our case, we used 3-dimensional latent space:

$$\boldsymbol{\theta}^l = [\theta_1^l, \theta_2^l, \theta_3^l]^T. \tag{2}$$

As input/output of the network, we used DMP parameters (Eq. 4) describing the trajectories in Cartesian space in $x$ and $z$ axes as well as the orientation around the $y$ axis (note that we consider throwing a planar problem) together with parameter describing the time in which the trajectory is executed - $\tau$. This sets the input of AE network to:

$$\boldsymbol{\theta} = \left[\boldsymbol{\Theta}_x^T, \boldsymbol{\Theta}_z^T, \boldsymbol{\Theta}_{rot_y}^T, \tau\right]^T, \tag{3}$$

$$\boldsymbol{\Theta}_{\{.\}} = [\mathbf{w}_{\{.\}}, y_{0,\{.\}}, g_{\{.\}}], \tag{4}$$

with $y_0$ the starting point and $g$ the goal. For more information on DMPs see [22].

In our implementation we used 20 weights per each DoF and different time scaling factors, which summed up to having 67 input/output parameters of the AE neural network. The neural network was fully connected with [67 15 3 15 67] neurons in each layer and all the values were normalized in range [-1,1].

## IV. RL-Aided Domain Adaptation

As mentioned in the introduction, the main focus of this paper is put on $DNN_2$ network that maps the target of throwing to the latent space parameter values. In the process of learning, we combine training of the network and RL algorithm called reward weighted policy learning (RWPL), which can be interpreted as a simplified version of PoWER algorithm [7].

In the beginning, $DNN_2$ is trained only with the dataset of kinematic trajectories, obtained as described in Section III. The first step in our algorithm is to check which weights and biases change the most when changing the domain, i. e., when going to dynamical simulation. For this, we execute several (we used 9) throws in dynamical simulation and measure the landing spot. These can be throws to random targets, or they can be evenly distributed over the target area. The latter is less likely to over-fit to one random area, but it requires more knowledge and input by the user. With this small amount of data, we retrain $DNN_2$ network in only several epochs (we used 10), and check the difference in the weights and biases. We sort the weights and biases by the difference and use several ones (we used $N = 15$ of both) that change the most to be modified by the RL algorithm:

$$\boldsymbol{\theta}_{RL} = \{\omega_n, b_n\}_{n=1}^N. \tag{5}$$

Since these parameters are changed by both the RL and retraining, and the others only by retraining, it is very similar as freezing the other parameters.

In the next step of transfer learning, we choose queries that are equally spread over the robot workspace. For this, we have chosen $M$ equally spaced targets (we used separately 12 and 16) in the same range as in our kinematic database. In order to improve performance of $DNN_2$, we learned to throw simultaneously on all $M$ targets and we were assigning them a cumulative reward. After executing throws at the targets, we retrained the network in only one epoch by using newly obtained data, and updated $\boldsymbol{\theta}_{RL}$ with RWPL. For clarity, the procedure describing the learning steps is given in Algorithm 1.

**Algorithm 1:** Procedure of improving neural network connecting targets with the latent space values.

**procedure** Improve $DNN_2$
  **set** targets $\mathbf{T} = \{x_m, z_m\}_{m=1}^M$
  **set** $\boldsymbol{\theta}_1^{RL}$ based on the difference between two NNs
  **set** $i = 0$
  **repeat**
    increase $i$
    add noise on RL parameters $\boldsymbol{\theta}_i^{RL} = \boldsymbol{\theta}_i^{RL} + \boldsymbol{\varepsilon}$
    **set** corresponding $DNN_2$ parameters to $\boldsymbol{\theta}_i^{RL}$
    **set** R=0
    **for** m=1:M
      use $DNN_2$ to map: $\boldsymbol{T}_m$ -> $\boldsymbol{\theta}_m^l$
      use decoder to map: $\boldsymbol{\theta}_m^l$ -> $\boldsymbol{\theta}_m$
      **execute** motion in the new domain
      measure landing spot $\boldsymbol{H}_m = [x'_m, z'_m]$
      give reward $r_m = ||\boldsymbol{T}_m - \boldsymbol{H}_m||$
      $R = R + r_m/M$
    **end**
    train $DNN_2$ with $\boldsymbol{H}_m$ -> $\boldsymbol{\theta}_m^l$, m=1...M;
    **use** RWPL to update $\boldsymbol{\theta}_{i+1}^{RL} = f(\boldsymbol{\theta}^{RL}, R)$
    reduce search noise $\boldsymbol{\varepsilon}$
  **until** $i$ =number of updates

## V. Experimental Evaluation

We evaluated the increase in precision of robotic throwing when going from one domain to another. In our case, we changed the domain from kinematic to dynamic simulation. We compared out approach with continuous update of the network – lazy learning.

### A. Experimental Setup

We performed experiments in MuJoCo Haptix simulation connected to MATLAB for the task of throwing action. For this purpose, the 7 DoF Mitsubishi PA-10 robot equipped with a throwing spoon was used. The experimental setup is shown in Figure 1. Assuming that the robot is directed towards the target (that its orientation is correct), the throwing problem can be considered as a planar problem in the saggital plane of the robot. Because of this, targets could be described by only two parameters (the distance from the base in x and z axis) and only three joints that contribute to performing a throw were kept active and taken into the account when calculating inverse kinematics for the initial database.

### B. Data

In order to evaluate our approach, we compared our method to simple lazy learning method. After preparing the database of executable trajectories containing 3247 targets and corresponding trajectories, all the trajectories were encoded as DMPs and used for training of the autoencoder network that contained 3 latent space values. $DNN_2$ network containing [2 3 15 5 3] neurons in its layers was trained to match these targets to corresponding latent space values as given by (1).

We repeated two sets of experiments with different amounts of targets for learning. In the first set, we learned M=16 targets equally spread within the range of the robot. In the second set of experiments, we repeated with M=12 targets. We learned all M targets in 9 iterations, which sums the learning procedure to 144 shots for the case of learning 16 targets and 108 shots for the case when learning 12 targets. After each iteration of learning, the performance of $DNN_2$ was tested on 64 targets shown with red circles in Figures 4-6. For lazy learning, we used 200 randomly chosen targets. After each shot, $DNN_2$ was updated with the new data in 10 epochs (same as in our algorithm) and the performance was measured on the same 64 targets as in the case of learning with the method given in Alg. 1.

### C. Results

Figure 3 shows the convergence of the error when doing transfer learning using 16 targets, 12 targets and when using lazy learning. Error (and reward) is given by

$$r_m = ||\boldsymbol{T}_m - \boldsymbol{H}_m||, \tag{6}$$

where, just as in Alg. 1, $\boldsymbol{T}_m$ is the target of $m$-th throw and $\boldsymbol{H}_m$ is the landing spot of the same throw. Data for the case of learning 16 targets is given in red, for 12 targets in green and for the case of lazy learning in blue. Upper graphs show the average error convergence and lower graphs show the the maximal error convergence. We can see that the proposed algorithm evidently outperformed lazy learning in the number of required executions and achieved accuracy in both cases when using 12 and 16 targets. While the top plot shows that the average error of throwing was even better for

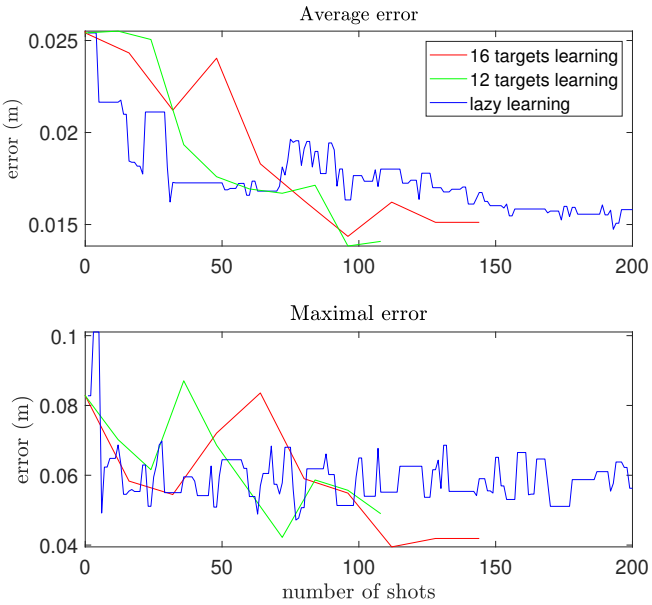


Fig. 3: Convergence of the error when we used our method on 16 targets is shown with red line, with 12 targets with green line and with lazy learning with blue line.

12 targets, the maximal error was lowest when using our approach and 16 targets.

Results in Fig. 4, presented as a contour plot, show an error of throwing with $DNN_2$ in the new domain (dynamic simulation) trained only using the initial kinematic database.

Figure 5 shows the error after the first update (after updating with 9 random targets that were used to determine important parameters for RL). Therefore, this plot shows the performance of $DNN_2$ at the beginning of the learning using our proposed approach.

Contour plot in Fig. 6 shows the performance of the network at the end of the training with our approach when using 16 targets.

Bar plots in Fig. 7 show the average and maximal error for the kinematic database, at the beginning of the learning and at the end of the learning for the case where 16 targets were used.

The results show obvious decrease in error at each step

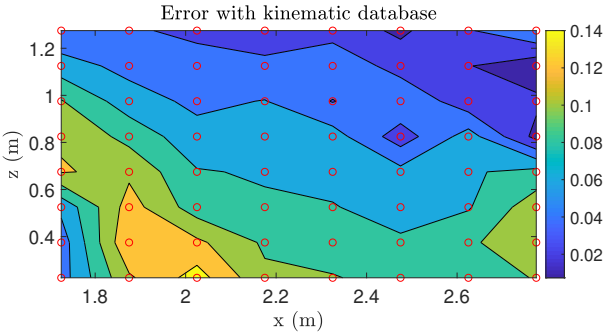


Fig. 4: Contour plot of the error with using the kinematic database for training the neural network

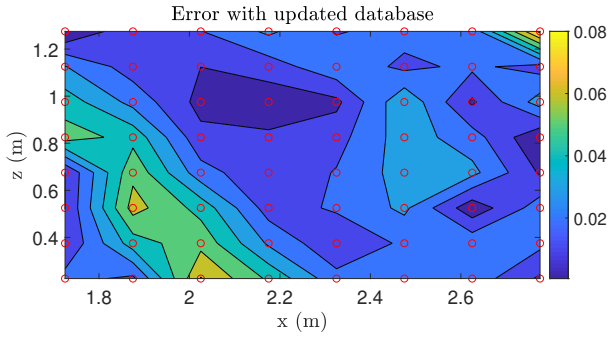


Fig. 5: Contour plot of the error after update of the neural network with only 9, equally spaced targets. This also represents the initial point for our proposed approach.

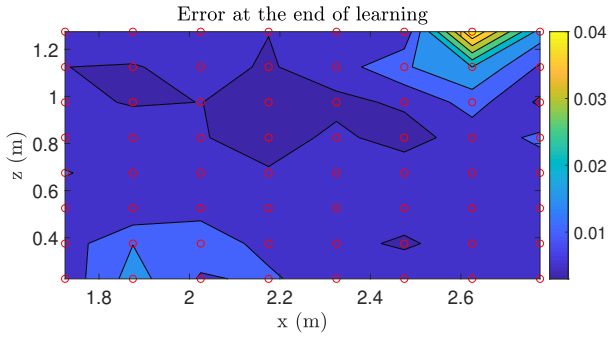


Fig. 6: Contour plot of the error at the end of learning.

of the learning process. Execution of only 9 targets and updating the network gives 64.5% decrease in average error. However, as it can be noticed from Fig. 3, this lazy learning method comes to a saturation point after approximately 100 iterations. On the other hand, a major benefit of our method, where retraining of the network is combined with performing RL, is that it constantly continued to improve the performance. After 144 iterations it gained additional 40.5% drop in the error. Table I sums-up the results.

| | $AE_k$ | $AE_{upd}$ | $AE_{upd}+RL$ | $DMP_k$ | $DMP_k$ |
|---|---|---|---|---|---|
| Avg | 0.072 | 0.025 | **0.015** | 1.293 | 0.918 |
| Max | 0.148 | 0.082 | **0.041** | 1.903 | 0.453 |
| NS | 3247 | 3247+9 | 3247+9+144 | 3247 | 3247+9 |

TABLE I: Average, maximal error and number of samples used for training of NN. Samples obtained in a new domain (dynamic simulation) are underlined.

## VI. Discussion

The first topic of discussion is generality of the proposed approach, which was applied only for a single tasks and with a lot of user-knowledge input into the preparation. An example of inputted user knowledge is the even spread of the targets for learning. However, the overall approach is not different than transfer learning for visual tasks, where one utilizes the already learned feature extraction of a given, trained network [11], where taking a pretrained network and using it as a starting point to learn a new task is typically much faster. This new task might be something very specific, for example classification of some new, previously not learned objects. Even in this case the classification

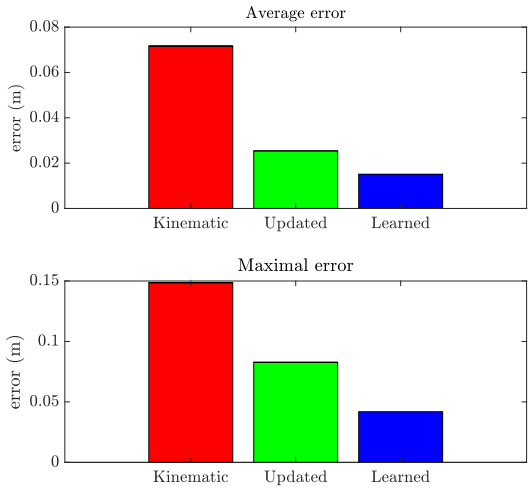


Fig. 7: Bar graphs of the mean and maximal error for the case of kinematic database, after the first update and at the end of the learning.

will be better if the images of the new objects cover as much viewpoints and diversity as possible. The latter are again provided by the user. Thus, the proposed approach is exactly the same – the user provides initial data that provides diversity.

The second topic of discussion is how would a robotic system function with separate DNN for each specific task. This is in fact no different that with humans, where we have separate centers in brains that are responsible for separate behaviors. Obviously, these are a part of a much larger network, but this is not the topic of this paper.

The approach presented in this paper has shown that even with a smaller number of required executions in the target domain (28% smaller for the case of 16 targets and 46% for 12 targets) were enough to outperform conventional lazy learning approach (Fig. 3). The approach can be further extended, by adding another layer, and having three or even more consecutive domains: kinematic $\mapsto$ dynamic $\mapsto$ real-world. However, as explained in Fig. 2, the approach already includes consecutive, chained DNNs. Chaining of DNNs can be applied, for example, to improve classification [25]. In our case, using a deep AE and its decoder, which is chained to the throwing $DNN_2$, has a profound impact on the effectiveness of the approach. because the dimensionality of the task is greatly reduced, a lower number of "real-world" trials and RL executions is needed. Figure 8 shows the error of throwing when the kinematic database is applied directly to DMP parameters, given with (3) and (4). As the results show, the error of throwing in the adapted domain (dynamic simulation) is in the range of 1m, so 50 times higher than when the dimensionality is reduced. Even when the throwing DNN is updated, the error remains in the range of half a meter and more. On the other hand, when using the dimensionally-reduced data, as in our approach, the error is in the range of a few cm.

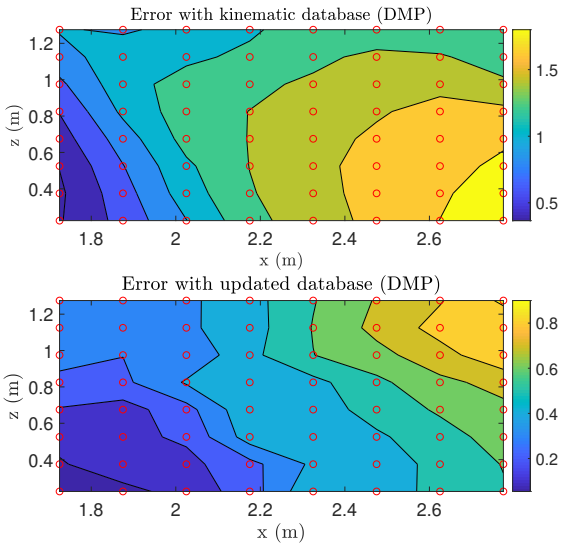


Fig. 8: Contour plot of the error with kinematic and updated database when using DMP instead of latent space.

## VII. Conclusion

The proposed approach provides a method for sim-to-real transfer learning with domain adaptation, where RL is applied to modify the most changed neural network parameters. Thus, the changing of these parameters is affected by both retraining and RL, which acts similar as freezing of a part of the network, because the most changed part is changed much more. Application to robot throwing and transfer from simple kinematic to complex dynamic simulation, which stands in place of the real-world, has shown that the approach can considerably accelerate the learning in the adapted domain.

In the future we will test the approach in consecutive kinematic $\mapsto$ dynamic $\mapsto$ real-world mapping. Despite dealing with only several hundred real-world throws, the task has turned out to be quite challenging. Furthermore, we will apply the proposed approach to a more complex robot such as the full-sized humanoid robot TALOS, where the difference between simulation and real-world behavior might be even more pronounced.

## Acknowledgment

This work has received funding from the program group Automation, robotics, and biocybernetics (P2-0076) supported by the Slovenian Research Agency and from the EU's Horizon 2020 grant ReconCycle (GA no. 871352).

## References

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[2] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[3] R. Pahič, A. Gams, and A. Ude, "Reconstructing spatial aspects of motion by image-to-path deep neural networks," *IEEE Robotics and Automation Letters*, vol. 6, no. 1, pp. 255–262, 2021.

[4] J. García, Fern, and o Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.

[5] K. Chatzilygeroudis, V. Vassiliades, F. Stulp, S. Calinon, and J.-B. Mouret, "A survey on policy search algorithms for learning robot controllers in a handful of trials," *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 328–347, 2020.

[6] B. Nemec, R. Vuga, and A. Ude, "Exploiting previous experience to constrain robot sensorimotor learning," in *IEEE-RAS International Conference on Humanoid Robots*, pp. 727–732, 2011.

[7] R. Pahič, Z. Lončarević, A. Gams, and A. Ude, "Robot skill learning in latent space of a deep autoencoder neural network," *Robotics and Autonomous Systems*, vol. 135, p. 103690, 2021.

[8] Z. Lončarević, R. Pahič, A. Ude, and A. Gams, "Generalization-based acquisition of training data for motor primitive learning by neural networks," *Applied Sciences*, vol. 11, no. 3, p. 1013, 2021.

[9] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[10] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, 2019.

[11] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.

[12] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020.

[13] S. Park, J. Kim, and H. J. Kim, "Zero-shot transfer learning of a throwing task via domain randomization," in *International Conference on Control, Automation and Systems (ICCAS)*, pp. 1026–1030, 2020.

[14] M. Kaspar, J. D. Muñoz Osorio, and J. Bock, "Sim2real transfer for reinforcement learning without dynamics randomization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4383–4388, 2020.

[15] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.

[16] O. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[17] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," pp. 12619–12629, 2019.

[18] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4243–4250, 2018.

[19] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[20] O. Sigaud and F. Stulp, "Policy search in continuous action domains: An overview," *Neural Networks*, vol. 113, pp. 28–40, 2019.

[21] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine learning*, vol. 84, p. 171–203, July 2011.

[22] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[23] L. He, G. Wang, and Z. Hu, "Learning depth from single images with deep neural network embedding focal length," *IEEE Transactions on Image Processing*, vol. 27, no. 9, pp. 4676–4689, 2018.

[24] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[25] K. Zaamout and J. Z. Zhang, "Improving neural networks classification through chaining," in *Artificial Neural Networks and Machine Learning – ICANN 2012* (A. E. P. Villa, W. Duch, P. Érdi, F. Masulli, and G. Palm, eds.), (Berlin, Heidelberg), pp. 288–295, Springer Berlin Heidelberg, 2012.