Adaptive Domain Randomization for Sim-to-Real transfer in Mechatronic Systems

Stijn Woestenborghs
Supervisor: prof. df. ir. Guillaume Crevecoeur
Counsellor: ir. Jeroen Taets

***Abstract*—Solving continuous control problems using reinforcement learning has seen growing interest the past years. The idea is very appealing. By conducting a lot of experiments the controller is rewarded for actions that have led to good results, after which it is more or less likely to repeat some specific behaviour. In reinforcement learning the controller is called an agent and it is represented by a neural network. Due to the amount of experiments needed to reach the desired performance, training is done using a digital twin. A simulated environment provides data that is cheap, fast and easy to acquire and it enables the training of control policies using reinforcement learning. However, an agent that performs well in simulation does not always work on a real mechatronic system. This phenomenon is known as the reality gap. Much effort can be put into perfecting a digital twin but one can never model every aspect of the real system. Reality remains unpredictable, noisy and full of other unexpected influences. This master dissertation aims to address the problem of transferring control policies trained in simulation to the real world using domain randomization. By presenting a lot of unexpected scenarios during the training stage, domain randomization aims to make the control policies more robust in the real world. Certain environment parameters are randomized and sampled from a Gaussian distribution and it turns out that determining these distributions can be a very tedious process. In a second phase, adaptive domain randomization is investigated which automates this domain randomization process.**

I. Introduction

A reinforcement learning task is defined by two key characteristics: the environment and the agent. The environment is the world the agent interacts with. One can describe the environment by its state space s. From this an observation o is made which is a partial or total description of the state. This observation is used by the agent to determine the best corresponding action a. An action lies within the action space which can be either discrete or continuous. The action spaces of all used environments are (or are converted to) the continuous space. A visual representation of a basic reinforcement learning interaction loop is represented in Figure 1.

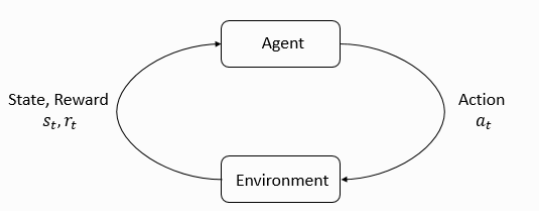

Fig. 1: Basic reinforcement learning interaction loop

A. Limitations

While the idea behind reinforcement learning controllers for mechatronic systems is simple, many complications emerge once the environment becomes a little more complex. Of the many difficulties that arise when integrating reinforcement learning into industry, some are listed below.

First, the process requires a measurement of the state of the system. Although this seems trivial at first sight, in reality it is sometimes difficult to obtain. In a closed system it is often physically impossible to place the required sensors and next to that measurements are prone to errors and noise. Even when one is able to make an observation of the state, this observation might not be reliable.

Secondly, even if a task is conceptually simple, defining an appropriate reward is difficult to define and measure in the real world. Also the reward would need to be measured by sensors and is therefore difficult to automate, not really scalable and subjected to errors and noise.

Finally, the reinforcement learning algorithm needs a very large amount of data. Learning by trial and error and being rewarded for desired behavior means that these real environments have to run in real time to collect all the data. Often such environments are not scalable in an industrial setting. Costs are rapidly rising. Running single consecutive trials is too time consuming and running multiple mechatronic systems in parallel is unfeasible in many cases.

A solution to these problems already exists today: the digital twin. Regarding reinforcement learing, the digital twin comes with very valuable advantages. The complete state of the system is known at all times, it is not subject to noise or measurement errors, reward functions can be integrated through mathematical equations and the digital twin is highly scalable. Simulation time-steps depend on computation power and not on real time which results in faster iterations during the reinforcement learning process. On top of that, simulation environments can easily run in parallel, which also drastically reduces training time.

B. The reality gap

In short, the simulation environment provides data that is cheap, fast and easy to acquire. It would make it interesting to integrate a reinforcement learning controller, but an agent that performs well in simulation does not always work on real mechatronic systems [1] [2]. This phenomenon is known as the reality gap. A simulation environment is often an idealised representation of reality and although much effort can be put into perfecting a digital twin, one can never model every aspect

of the real system. Reality remains unpredictable, noisy and full of other unexpected influences. In fact simulations are designed to make large assumptions to run faster like rigid bodies or discrete time steps... Even when reality is modelled acceptably good. Determining accurate model parameters is very hard. Model parameters like friction, damping and inertia are difficult to determine because they are not directly visible in measurement data. Additionally, the more accurate and complex the simulation model becomes, the more model parameters are to be determined.

C. Domain randomization

With a proven record in computer vision [3], domain randomization is making its entrance in sim-to-real transfer for mechatronic systems. [4] [1]. Related to sim to real transfer, there are also several studies on automating this domain randomization process. [5] [6] [7]. Domain randomization is trying to realise the policy transfer from sim to real in the following way.

Rather then using the actual parameters of the mechatronic system, one changes these environment parameters in each simulation. The hypothesis is that when the agent encounters a wide enough range of unrealistic situations. It will generalize in reality, completing the sim-to-real policy transfer.

II. Reinforcement Learning

In Figure 1 the agent was already introduced. The agent uses a policy that decides what actions to make. A stochastic policy is denoted by $a_t \sim \pi(.|s_t)$. Another critical component in reinforcement learning is the reward function $r_t = R(s_t, a_t, s_{t+1})$ which can be a function of the current state, current action and future state. The return is the sum of rewards during a complete trajectory τ in the environment $R(\tau) = \sum_{t=0}^{T} r_t$.

Policy optimization comes down to maximising or minimising the expected return. [8] Optimizing the stochastic policy can be formulated as:

$$\pi^* = arg\max_{\pi} J(\pi) = arg\max_{\pi} \underset{\tau\sim\pi}{E}[R(\tau)] \tag{1}$$

Which defines the central reinforcement learning problem in policy optimization. Two different reinforcement learning algorithms were implemented.

A. Reinforcement learning algorithms

1) Proximal Policy Optimisation: Proximal Policy Optimization (PPO) [9] is a policy gradient method for reinforcement learning which optimizes a surrogate objective function using stochastic gradient ascent. Exploration comes from sampling new actions from the last trained version of the stochastic policy and a training update only takes into account the last sampled data. During the training process exploration becomes less and less random, which may cause the policy to get blocked in a local minimum [8]. Since Proximal Policy Optimization optimizes a surrogate objective function $J'(\pi)$, the central optimization problem changes to:

$$\pi^* = arg\max_{\pi} J'(\pi) \tag{2}$$

For a full expression of the surrogated function $J'(\pi)$, see [9].

2) Soft Actor Critic: Unlike on-policy optimization based algorithms, Soft Actor-Critic (SAC) is a state of the art off-policy reinforcement learning method where all its data is used during training. Within SAC the actor aims to maximize the expected reward while also maximizing entropy, which is a measure of randomness in the policy. By means of off-policy updating, the method tries to provide the best possible return while also maximising its randomness. This can prevent the policy from converging too early into a local minimum [8] [10]. The additional maximisation of entropy changes the central optimisation problem to a multi-objective optimisation problem:

$$\pi^* = arg\max_{\pi} \underset{\tau\sim\pi}{E}[\sum_{t=0}^{T}(R(s_t, a_t, s_{t+1}) + \alpha H(\pi(.|s_t)))] \tag{3}$$

For the total trajectory τ going from $t = 0..T$, one optimizes the policy π which maximises both expected reward $r_t = R(s_t, a_t, s_{t+1})$ and expected entropy $H(\pi(.|s_t))$ of the policy. In the equation $\alpha > 0$ represents a trade-off coefficient.

B. Environments

OpenAI Gym [11] provides dozens of open source environments that are easy to interpret and flexible to customize. Three environments were isolated: CartPole-v1, Pendulum-v0 and Acrobot-v1. The three environments are represented respectively in Figure 2.

Fig. 2: OpenAI environments: CartPole, Pendulum and Acrobot

C. Simreal

Throughout the dissertation special attention is paid to usability and algorithm performance. One of the reasons reinforcement learning struggles to be integrated into industrial setting is due to budget constraints. Adaptive domain randomization proposes a way to make reinforcement learning controllers attractive through a fast and highly scalable solution, but it requires to make changes to the reinforcement learning algorithms. In order to guarantee accessibility and performance, it was decided to isolate each reinforcement learning baseline and all environments from their parent library and to combine them in Simreal. Additionally, GPU acceleration is enabled and verified using a local GPU. [1].

[1]The used GPU was NVIDIA GeForce GTX 1050 Ti

III. Domain Randomization

Unrealistic situations are generated by domain randomization. Selected domain parameters for each environment are depicted in table I.

TABLE I: Environment specific randomization parameters

CartPole	Pendulum	Acrobot
mass cart: m_l	mass: m	mass rod 1: m_1
mass pole: m_p	length: l	mass rod 2: m_2
lengthpole: l_p		length rod 1: l_1
		length rod 2: l_2

There are two main arguments that define how domain parameters are randomized during the training process:

- Probability distribution characteristics ϕ from which the parameters are sampled.
- Randomization frequency

A. Probability distribution characteristics

The domain parameters are sampled from a Gaussian distribution. Therefore, two characteristics are required for each domain parameter (e.g. the mean and standard deviation[2]). The probability distribution characteristics are summarized in a matrix ϕ. An example for the Pendulum environment is given below.

$$\phi = \begin{bmatrix} \mu_l & \sigma_l \\ \mu_m & \sigma_m \end{bmatrix}$$

B. Randomization frequency

Secondly, the randomization frequency is defined by two arguments: Type, which is either *every_episode* or *every_variation* and Frequency, which can take any positive integer excluding zero. For example, one configuration might be (*every_variation*, 50), which triggers a randomization of the domain parameters every 50^{th} step in the environment. In addition, the domain parameter will be sampled from their corresponding probability distribution characteristics ϕ.

IV. The Evaluation Environment

It is possible to provide a different evaluation environment with the reinforcement learning engine, which allows for connecting a realistic environment at this location. This section aims to enable fast evaluation by constructing an evaluation environment that is a more complex version of the simulated one. Reality is imitated by adding noise and friction to the standard Gym environment. In reality, this evaluation environment is meant to be replaced by the real one. But for now this allows for quick iterations and evaluations in the current sim-to-real solution. The implemented corrections are:

- Actuation noise
- Observation noise
- Parameter offset
- Friction

[2]There are other possibilities but using mean and standard deviation results in the least amount of parameters

It is worth mentioning that implementing friction required changes to the dynamical equations of each environment. These changes are therefore environment specific. The other corrections could be implemented in a way that was compatible with the three different environments.

A. Parameter Study

First of all, training agents with and without domain randomization for the CartPole and Pendulum environment results in the maximum possible return in the corresponding noisy evaluation environment. This observation makes it impossible to evaluate the implied sim-to-real solution.

This was not the case with the Acrobot. Performance without domain randomization was bad in the evaluation environment. This observation is a reflection of the reality gap and is depicted in Figure 3.

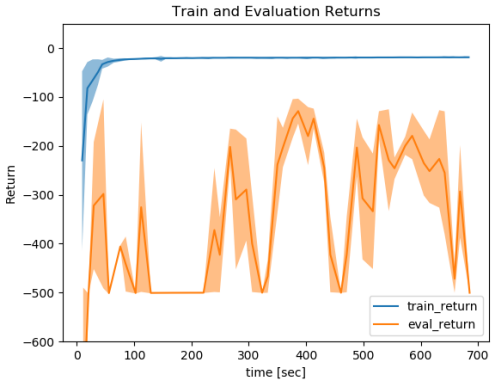

Fig. 3: Acrobot training curves without domain randomization

Additionally, when using domain randomization, performance in the evaluation environment strongly depends on the used randomization parameters (e.g. the used mean and standard deviation from which the domain parameters are sampled). Another observation that is worth mentioning is that performance also depended heavily on the used randomization seed.

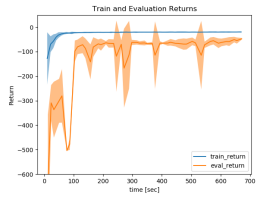

Fig. 4: Acrobot training curves with domain randomization and bad performance

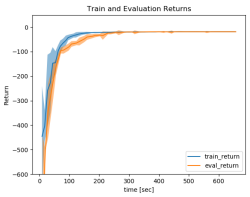

Fig. 5: Acrobot training curves with domain randomization and good performance

In other words, the current problem has become much more complex. Multiple unknowns determine whether the sim-to-real transfer is successful or does not converge to a

solution at all. Moreover, we have no idea which set of distribution parameters leads to the best performance in the noisy evaluation environment.

This is why a parameter study was performed where mean and standard deviation of all domain parameters of the Acrobot where shifted together. [3] Additionally the parameter study was done for five different seeds.

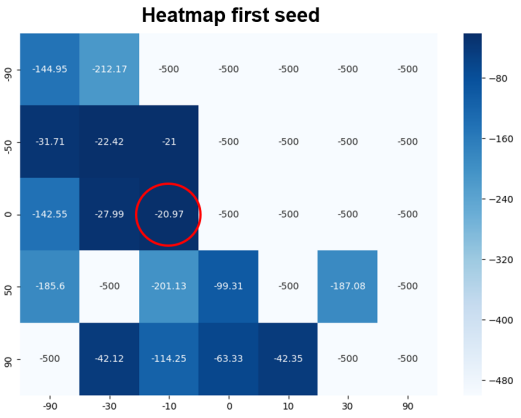

Fig. 6: Parameter study: heatmap first seed

Optimal results for all seeds are given in table II at the end of the paper. An example for the first seed is represented in Figure 6. Here the x-axis represents variation in means, while the y-axis represents variations in standard deviations. Models that did not converge to a solution in the training environment or totally failed in the evaluation environment were left out and set to a return of -500, the worst possible return. This way one increases readability of the heat-maps, which can be justified since we would never consider a non-converged agent as a valid solution.

As one can see, there is a set of probability characteristics ϕ (that might differ from our starting point) which succeeds in closing the reality gap. Additionally, the optimal return with domain randomization is better than without domain randomization.

It is worth mentioning that for 5 seeds there are each 35 agents trained. The 175 agents led to the conclusion that there are sets of probability characteristics that are better than others but we cannot conclude that we found the optimal set since the parameter study shifted all means and standard deviations together. During adaptive domain randomization one aims to find this optimal set $\phi*$ automatically.

V. Adaptive Domain Randomzition

As shown in equation 1 the goal in adaptive domain randomization (ADR) is to train a policy π that maximises the expected return $\underset{\tau\sim\pi}{E}[R(\tau)]$ along a trajectory τ of the environment. In addition to this central optimisation problem the system dynamics of the training environment are now changing. Let ξ represent the domain parameters, sampled from a probability distribution p_ϕ. Let $P_{\xi\sim p_\phi}$ represent the system dynamics under the sampled domain parameters. Then the central problem for domain randomization reformulates itself to maximising the expected return under the system dynamics induced by the parameter distribution $p_\phi(\xi)$:

$$\pi^* = arg\max_{\pi} \underset{P_{\xi\sim p_\phi}}{E}[\underset{\tau\sim\pi}{E}[R(\tau)]] \tag{4}$$

For clarification, consider the concrete example of the Acrobot. The randomization parameters in this environment are:

$$\xi = \{l_1, m_1, l_2, m_2\}$$

The Simreal's domain randomization wrapper allows for these parameters to be sampled from a normal distribution p_ϕ characterised by a mean and standard deviation for each parameter. Representing the probability distribution characteristics ϕ.

$$\phi = \begin{bmatrix} \mu_{l_1} & \sigma_{l_1} \\ \mu_{m_1} & \sigma_{m_1} \\ \mu_{l_2} & \sigma_{l_2} \\ \mu_{m_2} & \sigma_{m_2} \end{bmatrix}$$

Still, with the above formulation, one must manually search for a set ϕ, that leads to the maximum expected return. The goal of adaptive domain randomization is to automatically optimize these distribution characteristics. To achieve these requirements, the following framework was implemented.

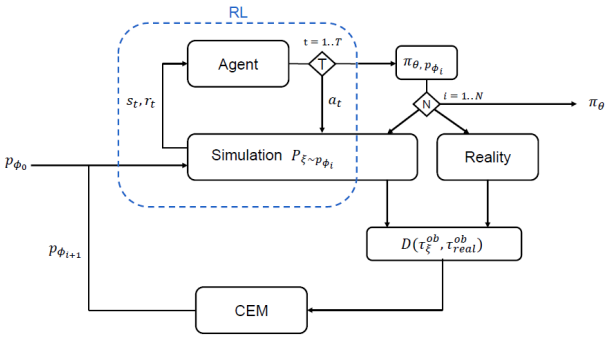

Fig. 7: Adaptive Domain Randomization framework

The outer optimisation loop aims to find a set of probability distribution characteristics that brings an observed trajectory in simulation τ_ξ^{ob} closer to a trajectory in reality τ_{real}^{ob}. This is achieved by minimising the discrepancy between the two trajectories $D(\tau_\xi^{ob}, \tau_{real}^{ob})$ using a stochastic optimisation algorithm, like the Cross-entropy method (CEM).

Additionally, formulating the second optimisation problem that arises for adaptive domain randomisation can be done as follows. Aside equation 4, we look for a set of probability distribution characteristics ϕ that, after policy training π_{p_ϕ} under sampled system dynamics $P_{\xi\sim p_\phi}$, minimises the expected discrepancy between trajectories in simulation and in reality.

$$\phi^* = arg\min_{\phi} \underset{P_{\xi\sim p_\phi}}{E}[\underset{\pi_{\theta,p_\phi}}{E}[D(\tau_\xi^{ob}, \tau_{real}^{ob})]] \tag{5}$$

[3] For simplicity the four domain parameters mean and standard deviation were shifted together. This reduces the 8 dimensional problem to a 2 dimensional one.

An important point to note is that this outer optimisation loop requires trajectories to be generated in reality. In order to limit real data generation as much as possible (due to cost in time and effort) only one reference trajectory is created per iteration. Assuming that computing power is not the constraining factor, one can compare this reference trajectory with as many simulation trajectories for different domain parameters as necessary. This way of working does not contradict the initial problem. Now, only about 10 evaluations will have to be done, whereas if the agent is trained in reality, sometimes thousands of evaluations are necessary.

A. Cross Entropy Method

An optimisation algorithm that is very suitable for the purpose of ADR is the Cross Entropy Method (CEM) [12]. CEM can be applied very easily to our optimisation loop. A high level overview can be found in Figure 8.

The input of the optimisation is ϕ_i and the corresponding trained policy π_{p_ϕ}. A first step is to evaluate the policy on the evaluation environment and save its observed trajectory τ_{real}. We do this only once every CEM optimisation in order to keep costs low.

Secondly, a population of samples is generated. One sample $\xi_{i,k}$ represents one unique set of domain parameters sampled from its own distribution. Then, the new trajectories τ_ξ are simulated and saved for each generated sample in the population.

The next step in the Cross Entropy Method is to evaluate each sample's trajectory against the single reference from the evaluation environment. This is done using a self-defined discrepancy function $D(\tau_\xi^{ob}, \tau_{real}^{ob})$. The next paragraph describes this function more extensively but let's now assume that every sample has its own fitness.

The final step is to isolate the best samples (the elites) on the basis of fitness. For each domain parameter the mean and standard deviation of these elites are calculated, which then form the updated ϕ_{i+1}.

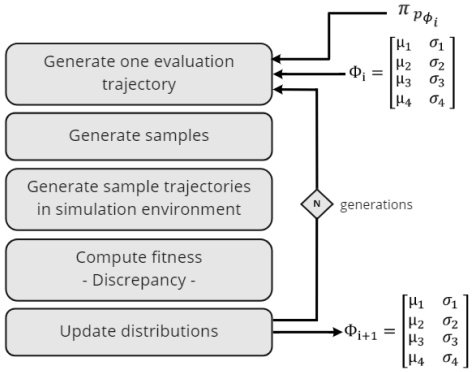

Fig. 8: Cross Entropy Method framework

B. Discrepancy Function

The discrepancy function measures the difference between a trajectory in simulation and a trajectory in evaluation. In the CEM method the idea is to optimise the parameter distribution characteristics in such a way that the difference in trajectory between sim and real is as small as possible. We are literally trying to minimise the gap between sim and real here. However, simulation trajectories are faced with two complications:

- It is not guaranteed that trajectories in simulation are as long as the reference trajectory.
- Once a simulation trajectory starts deviating, it can lead to completely different results compared to the results of the reference. This is the result of compounding errors.

To address the issues above, a simulation trajectory was generated based on the state and action from the real reference trajectory.

$$x_{t+1,\xi} = \pi_{p_\phi}(a_{t,real})|_{x_{t,real}} \tag{6}$$

Once the simulation environment is initialised to first contain the correct parameters $\xi_{i,k}$ and to be in the correct state, the same action is applied to the environment as in the reference. The successive observations can now be directly compared with the observation from that reference.

The only thing left is to formulate an objective function $D(\tau_\xi^{ob}, \tau_{real}^{ob})$ that defines the difference between a real trajectory and a simulated one. After some iterations the following discrepancy function was implemented:

$$D(\tau_\xi^{ob}, \tau_{real}^{ob})|_{i,k} = \sum_{t=1}^{n} \frac{\sum_{j=1}^{dim(x)} w(j)|x_{t,real}(j) - x_{t,\xi}(j)|}{n} \tag{7}$$

Where the weights $w(j)$ are manually optimised. It is worth mentioning that the function is not convex. However, in a setup where evaluation and simulation environments are the same, it is verified that CEM optimisation results in convergence to the actual domain parameters. This way, one gets a rough idea of the stability of the discrepancy function and the optimisation method.

C. Experiments

The complete pipeline is evaluated for the same five seeds we considered earlier in the parameter study. The adaptive domain randomization results are given in Table II below.

TABLE II: ADR experiment results

Seed number	Without DR	Parameter study	ADR
1	-64.27	-20.97	-18.96
2	-19.73	-20.73	-19.23
3	-163.22	-21.20	-24.74
4	-25.07	-19.39	-19.71
5	-23.91	-20.04	-19.30

Adaptive domain randomization also manages to achieve policy transfer from simulation to the noisy evaluation environment. In fact, instead of having to carry out a parameter study for each seed, ADR achieves to obtain better results in only one experiment. Only seed 3 seems to have converged to a local minimum. Recall that in the parameter study 35 experiments were performed per seed. Hence, the proposed

ADR solution leads to large time savings and is easily scalable with an increasing number of randomization parameters.

It is worth mentioning that for each seed, ADR was carried out only once in order to get good results. Only the fourth seed did not converge in evaluation for the first time. But after slightly increasing the standard deviation of randomization, the agent did converge the second run. This demonstrates the increase in efficiency compared to having to carry out a parameter study.

During the ADR training process all domain parameters were tracked. As an example the used domain parameters from the first seed are shown in Figure 9.

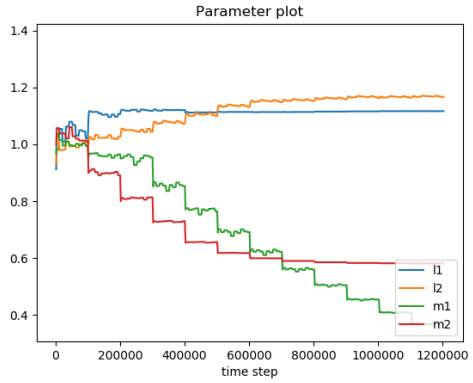

Fig. 9: ADR Domain Parameters: First seed

Rather than plotting all used domain parameters, one can visualise the probability distribution characteristics ϕ, which is updated across 10 ADR iterations. In Figure 10, the darker probability distributions represent successive iterations. One notices that especially l_1 and m_1 are gradually converging to a fixed value while randomization of l_2 and m_2 shifts but the standard deviation remains large.

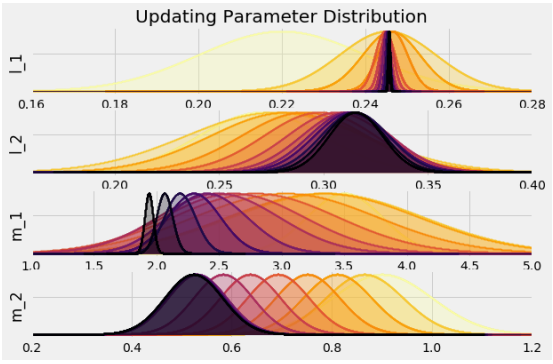

Fig. 10: ADR Updating Parameter Distribution ϕ: First seed

VI. Conclusions

Deploying reinforcement learning controllers for mechatronic systems in the real world is a novel, appealing and very promising idea. Domain randomization provides a possible solution to close the reality gap when the agent is trained in a simulated environment. Even though this master dissertation focused mainly on a defined task in the Acrobot environment, it succeeded in policy transfer from simulation to a more complex and noisy imitation of reality. It is too early to generalise these findings to the majority of mechatronic systems. This requires testing of agents in reality for several different mechatronic systems and tasks. However, a single success for a specific setting is a good starting point for the larger objective.

Optimal implementation of domain randomization can be a very tedious process. It is not straightforward to find the optimal set of probability characteristics. A parameter study can provide some insights, but this is not scalable when the number of domain parameters increases. The proposed adaptive domain randomization framework offers a solution to this inconvenient complexity. Additionally, in its current setting, ADR is able to achieve robust sim to real policy transfer regardless of the used seed.

Simreal, the library that was created, allows a user-friendly implementation of (adaptive) domain randomization. At the heart of the problem lies the cost and difficulty of real data generation and that is why the architecture of Simreal is designed to allow for easy expansion to new environments. Consistently attention was paid to maintain solid performance.

References

[1] Peng, X. B., Andrychowicz, M., Zaremba, W., Abbeel, P. (2018). Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*. Published. https://doi.org/10.1109/icra.2018.8460528.

[2] Pieter Abbeel. (2020, 16 februari). *Lecture 22 Sim2Real and Domain Randomization – Advanced Robotics at UC Berkeley.* YouTube. https://www.youtube.com/watch?v=ac_W9IgKX2c.

[3] Tobin J., Fong R., Ray A., Schneider J., Zaremba W., Abbeel P. (2017) Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017)* https://arxiv.org/abs/1703.06907.

[4] OpenAI et al. (2019) Solving Rubik's Cube with a Robot Hand https://arxiv.org/abs/1910.07113.

[5] Muratore F., Eilers C., Gienger M., Peters J. (2020) Data-efficient Domain Randomization with Bayesian Optimization https://arxiv.org/abs/2003.02471.

[6] Ramos F., Carvalhaes Possas R., Fox D. (2019) BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators *Robotics Science and Systems (RSS) 2019* https://arxiv.org/abs/1906.01728.

[7] Chebotar Y., Handa A., Makoviychuk V., Macklin M., Issac J., Ratliff N., Fox D. (2019). Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience. *2019 International Conference on Robotics and Automation (ICRA).* Published. https://doi.org/10.1109/icra.2019.8793789.

[8] OpenAI Spinning up (2021) *Spinning Up documentation. Key Concepts in RL* https://spinningup.openai.com/en/latest/spinningup/rl_intro.html, 2021-05-24.

[9] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. (2017) Proximal Policy Optimization Algorithms https://arxiv.org/abs/1707.06347.

[10] Haarnoja T., Zhou A., Abbeel P., Levine S. (2018) Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor https://arxiv.org/abs/1801.01290.

[11] OpenAI. (2021) *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. https://gym.openai.com/envs/CartPole-v1/, 2021-05-24.

[12] D. Ha, "A visual guide to evolution strategies," *blog.otoro.net*, 2017, https://blog.otoro.net/2017/10/29/visual-evolution-strategies/.