Monolithic vs. hybrid controller for multi-objective Sim-to-Real learning

Atakan Dag[1], Alexandre Angleraud[2], Wenyan Yang[1], Nataliya Strokina[1], Roel S. Pieters[2], Minna Lanz[2], and Joni-Kristian Kämäräinen[1]

***Abstract*— Simulation to real (Sim-to-Real) is an attractive approach to construct controllers for robotic tasks that are easier to simulate than to analytically solve. Working Sim-to-Real solutions have been demonstrated for tasks with a clear single objective such as ”reach the target”. Real world applications, however, often consist of multiple simultaneous objectives such as ”reach the target” but ”avoid obstacles”. A straightforward solution in the context of reinforcement learning (RL) is to combine multiple objectives into a multi-term reward function and train a single monolithic controller. Recently, a hybrid solution based on pre-trained single objective controllers and a switching rule between them was proposed. In this work, we compare these two approaches in the multi-objective setting of a robot manipulator to reach a target while avoiding an obstacle. Our findings show that the training of a hybrid controller is easier and obtains a better success-failure trade-off than a monolithic controller. The controllers trained in simulator were verified by a real set-up.**

I. INTRODUCTION

While recent advances in simulation-driven Reinforcement Learning (RL) are impressive [1], [2], [3], the bar by which these are measured in robotics is their capability to be deployed to real robot tasks. In the ”Sim-to-Real” approach [4], [5] the objective is to train a controller in a simulated environment and then transfer the controller to the real environment. Successful Sim-to-Real controllers have been demonstrated for object manipulation tasks [6], [7], [8], [9], [10], [11], [3]. These tasks have one main objective, but practical applications often need multiple simultaneous objectives such as ”reach the target” and ”avoid obstacles”. For example, in collaborative human-robot manufacturing the robot should complete its task without compromising human safety [12], [13], [14].

A straightforward solution in the spirit of reinforcement learning (RL) is to combine multiple objectives into the reward function and train a *monolithic* controller. This was demonstrated for robot reaching and collision avoidance by Pham et al. [9] who trained a controller in a simulator and then deployed it to a real robot. Pham et al. defined a reward function that includes positive reward for reaching the target and negative reward (penalty) for being too close to an obstacle. The design burden of the monolithic approach is in ”reward engineering” to find and tune a suitable reward function. An alternative approach was recently proposed by Sangiovanni et al. [8] who propose a *hybrid* controller that consists of a number of pre-trained controllers for each task and a switching rule (the closest obstacle being too close to the manipulator). In their case the design burden is in defining a suitable switching rule. Moreover, Sangiovanni et al. did not experiment their hybrid controller with a real robot. In our work, we adopt the hybrid controller model for multi-objective Sim-to-Real tasks.

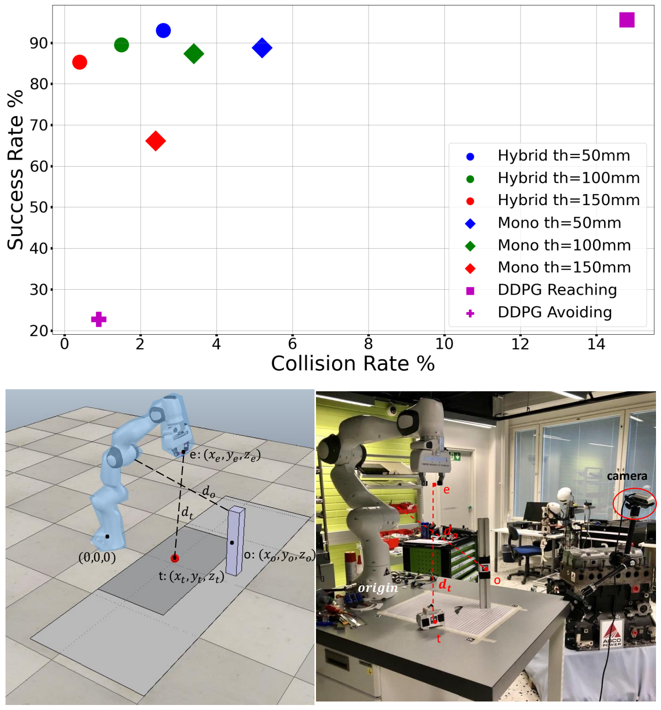


Fig. 1: Simulation results for the multi-objective ”Reach the target and avoid the obstacle” task (top graph). Average success and collision rates over 1000 episodes for the monolithic [9] and hybrid [8] controllers on three operation points each. The monolithic controller (diamonds) is not able to achieve the superior success-failure (collision) rates of the hybrid controller (circles). Moreover, the monolithic controller needs to be re-trained for each new operation point while the hybrid controller is trained only once. The simulated success and collision rates are verified by a real setup (bottom images).

In this work, we compare two approaches, monolithic and hybrid, in the multi-objective setting of a robot manipulator to reach the target and avoid collision with an obstacle. The problem is solved using the Sim-to-Real approach where all controllers are trained in a simulator and then transferred to the real environment. Our main contributions are:

1) reinforcement learning based implementations of the

[1]Computing Sciences and [2]Automation Technology and Mechanical Engineering, Tampere University, Finland.

monolithic and hybrid controllers for the ”reach and avoid” task in a simulated environment,
2) comparison results of the two controller models in the simulated environment indicating that the hybrid model is easier to train and tune, and it achieves better success-failure trade-off (Figure 1); and
3) verification of the results in Sim-to-Real by transferring and experimenting the controllers with a real Franka Emika Panda manipulator.

All code and data are publicly available at `https://github.com/atakandag/multi-objective-sim2real`.

II. RELATED WORK

Sim-to-Real – The Sim-to-Real research challenge is to train a controller in a simulated environment for the target task and then transfer the controller to a real environment. A popular benchmark task is robot reaching where a static or moving target is reached in the presence of one or multiple static or dynamic obstacles. Reinforcement learning based controllers have been demonstrated in simulators [6], [7], [8], and a number of Sim-to-Real setups have been demonstrated [9], [10], [11], [3]. These works differ by the selected RL algorithm, sensor inputs and RL reward functions used.

Matas et al. [5] study the Sim-to-Real problem for deformable objects that are more difficult to model than the rigid objects. Golemo et al. [15] address a more general problem of how to close the gap between the simulated worlds and the real world with more noise and distortions. Golemo et al. propose a Neural-Augmented Simulation (NAS) method that provides better policies than those trained only on simulated data. Peng et al. [16] show that by adding randomness to system dynamics during training the performance in real world tasks is improved. The most similar to our work are Pham et al. [9] and Sangiovanni et al. [8] who both address the problem of a multi-objective task where the target must be reached (success) but without hitting an obstacle (collision/failure). In our work, we compare the monolithic controller used by Pham et al. and the hybrid controller of Sangiovanni et al. for the multi-objective reaching task.

Multi-objective RL – Most of the current multi-objective RL approaches are tested and validated in simulated toy environments and often with a discrete state-action space [17]. Single-policy methods suggest construction of a compound reward function that combines rewards for each objective with different weights [9], [18]. There is a number of ”multi-policy” methods that assume a set of Pareto optimal policies to solve the task and the goal is to approximate these policies [19]. A recent work in this direction [20] proposes an evolutionary learning algorithm to find Pareto set approximation for continuous control problems. However, results have only been demonstrated for tasks in simulated environments. The hybrid controller model of Sangiovanni et al. provides a new alternative which combines single task controllers for a more complex multi-objective task.

III. METHODS

Reinforcement Learning (RL) provides a model-free approach to learn a controller from real or simulated episodes of a task. The objective is to learn a policy controller that maximizes the system reward at all system states. RL approaches can be divided to tabular and approximate methods [2]. Tabular methods, such as Temporal Difference Learning [21] and Q-learning [22], are suitable for problems with a small number of states and actions. Approximate methods are more suitable for large scale problems since the value and policy functions are learned by continuous function approximators such as neural networks. A neuron-type function approximation was already proposed as *actor-critic* by Barto et al. [23], but they decoded continuous inputs to a small number of discrete states.

Continuous policy approximation was proposed for the REINFORCE algorithm in [24]. A more general definition for the *Policy Gradient* (PG) approach was presented in [25]. Silver et al. [26] introduced the Deterministic Policy Gradient (DPG) method that optimizes policy function using an estimated action-value function (actor-critic). Neural networks can be used as the function approximators for DPG methods. For example, Deep DPG (DDPG) was introduced by Lillicrap et al. [1] and it was extended with Hindsight Experience Replay in [27]. The advanced methods are particularly useful for sparse rewards, but for simplicity we adopt the DDPG algorithm and use dense continuous rewards.

A. Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) by Lillicrap et al. [1] is a popular choice for off-policy RL. DDPG adopts the Actor-Critic architecture [23] where policy and value functions are estimated separately. In DDPG, both the state-dependent actor (μ) and the state-value critic (Q) functions are estimated by neural networks. The actor takes the current state as input and outputs directly control action values rather than probabilities of actions. The critic takes the current state and predicted action as inputs and outputs an estimate of the state-action value function $Q(s,a)$. The critic provides a baseline which reduces variance of the gradient based policy update steps. To further stabilize training, DDPG uses target networks, μ' and Q', in its update steps. Target functions are just time delayed versions of the original functions. Since it is an off-policy algorithm, DDPG uses an experience replay which stores experiences (s_t, a_t, r_t, s_{t+1}): at every timestep t, an agent observes state s_t, conducts action a_t, gets reward r_t for that action and observes the next state s_{t+1}. Actions predicted by the randomly initialized action network are obtained from

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t, \tag{1}$$

where θ^μ refers to the weights of the actor network and $\mathcal{N}_t$ is the noise added for exploration. In each step, a random batch of experiences (s_i, a_i, r_i, s_{i+1}) is sampled from the experience replay buffer and the critic is updated by minimizing the MSE loss between the expected Q value predicted by the critic network and the target Q value using the following equations:

$$\begin{aligned} L_{critic} &= MSE(y_t, Q(s_{t+1}, a_{t+1})) \\ y_t &= r_t + \gamma Q'\left(s_{t+1}, \mu'(s_{t+1})\right) \end{aligned} . \tag{2}$$

The main target is to find a policy that maximizes the expected return $\mathbb{E}[Q(s,a) \mid_{s=s_t, a_t=\mu(s_t)}]$. For that purpose, the actor is updated by the DPG policy gradient step [26]

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q) \mid_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu) \mid_{s_i}, \tag{3}$$

where N is the size of the batch. Updates are made to the target networks as the last step ($\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$ and $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$ where $\tau \ll 1$).

The above procedure is repeated for a number of episodes until the actor learns an effective mapping from the system states to action values. DDPG performs well in high dimensional spaces and for continuous input and output spaces.

Network details – Our actor and critic networks consist of 3 dense layers with ReLU activation units (actor: -300-300-30-; critic: -300-300-10-) except for the last actor layer that is $tanh$ to bound the action values. The learning rate was set to 0.001 and batch size to 64 for the ADAM optimizer in all experiments. The action bounds were reduced from $[-1.0, 1.0]$ to $[-0.2, 0.2]$ during testing for safety reasons.

B. Monolithic controller

Our work focuses on multi-objective tasks where rewards for the tasks can be dense or sparse. The object reaching task was investigated by Pham et al. in [9] and therefore our monolithic controller is similar to their work. However, to provide a more general solution we do not adopt their constraint optimization layer which is task specific. Instead, we sum all reward terms into a single composition reward function:

$$r = R_t + R_o + R_s, \tag{4}$$

where R_t is a dense reward for reaching the target, R_o is a sparse penalty for hitting an obstacle and R_s is a sparse reward for successfully reaching the target. It should be noted that we do not need to give weights to the different rewards as the penalty/reward values p_o and p_s implement this implicitly (see Section IV for the ablation study).

R_t is defined as the distance from the manipulator end-effector (tooltip) to the target as $R_t = -d_t$. This reward term is continuous and thus provides a dense reward for all states. The distance penalty is defined in meters.

R_o is defined as a sparse reward (penalty) if the distance of any part of the robot arm and the obstacle, d_o, is less than a fixed threshold τ_o as

$$R_o = \begin{cases} -p_o, & \text{if } |d_o| < \tau_o \\ 0, & \text{otherwise} \end{cases} . \tag{5}$$

R_s is defined as a sparse positive reward for a successfully completed action (robot reaches the target) as

$$R_s = \begin{cases} p_s, & \text{if } d_t < \tau_s \\ 0, & \text{otherwise} \end{cases} , \tag{6}$$

which is active if no collision occurs during the same time.

In the experiments, the parameters were set to the following values based on preliminary experiments: $p_o = 4.0$, $p_s = 4.0$ and τ_s = 50.0 mm. In the experiments we report various operation points by changing the value of τ_o (Figure 1). Ablation study of the other parameters is provided in the experiments as well.

C. Hybrid controller

Our hybrid controller is essentially the same dual-mode controller as proposed by Sangiovanni [8]. The original procedure is to train controllers for ”obstacle avoiding” (μ_1) and ”goal reaching” (μ_2) independently with DDPG and then combine them using a ”switching rule”. Based on the original work the two reward functions would be $r_1 = R_o$ and $r_2 = R_s$ (or $r_2 = R_t + R_s$), but interestingly we found that such hybrid controller is inferior to the variant where reward functions are enforced to correlate by mixing terms from each other.

The correlated hybrid controller is trained using the following reward functions:

$$\begin{aligned} r_1 &= R_t + R_o + R_s \text{ for } \mu_1 \\ r_2 &= R_t + R_s \text{ for } \mu_2 \end{aligned} . \tag{7}$$

In r_1 we emphasize the obstacle avoidance by setting $p_o = 10.0$ and $p_s = 10.0$ and this controller effectively avoids collisions but only rarely reaches the target (see ”DDPG avoidance” in Figure 1). On the other hand, r_2 is solely trained for reaching the target and thus it has high collision rate, but it almost always reaches the target (”DDPG Reaching” in Figure 1). Our approach to use common reward terms ”glues” together the two policies and thus the ”correlated” hybrid controller is able to achieve low collision rate and high success rate beyond the best result with the monolithic controller (Figure 1).

The final part of the hybrid controller is the *switching rule*. We adopt the simple distance based rule from Sangiovanni et al. as depicted in Algorithm 1. The hybrid controller operation point is set by the switching threshold τ_{hyb}. By adjusting the threshold the robot can be adjusted without re-training the controllers. The operation points with various thresholds are shown in Figure 1.

IV. EXPERIMENTS

A. Set-up

Both real and simulated experiments (see Figure 1) were performed using a Franka Emika Panda robotic arm with 7 degrees of freedom and controlled by producing the joint velocities. The multi-objective task was to reach the stationary target while avoiding to hit the dynamic obstacle. In the simulated environment the target was placed randomly within a pre-defined rectangular region (shaded dark gray in Figure 1). The obstacle was moved randomly across the work space defined by a larger rectangle that includes the target rectangle (light gray in Figure 1). The same setup was implemented on a physical setup also depicted in Figure 1

Algorithm 1: Hybrid controller - one control step

```
compute distance from the tool tip to the target $d_t$;
compute distance from the robot to the obstacle $d_o$;
if $d_o < \tau_o$ then
    halt; /* collision */
end
if $d_t < \tau_t$ then
    halt; /* goal reached */
end
if $d_o < \tau_{hyb}$ then
    step $\mu_1$; /* avoid */
else
    step $\mu_2$; /* reach */
end
```

where the white paper and table represent the two rectangle regions.

Simulator – CoppeliaSim [28] robot simulator was used to train the neural controllers. The simulator was interfaced with the PyRep API library [29]. The simulator computes the object and robot joint coordinates. In the simulator, the distance d_o between the robot arm and obstacle was computed using the minimum Euclidean distance between the robot body and the obstacle surface. In the physical setup the distance was approximated by computing the minimum distance between each robot joint and the obstacle center. Distance calculations and collision detection was automatically provided by CoppeliaSim. The distance between the target and robot end-effector, d_t, was computed similarly in both the simulator and the real setup using Euclidean distance between the target center and the robot tooltip.

Real setup – Franka Emika Panda robotic arm was interfaced through ROS Robot Operating System [30]. ROS provides real-time computation for the joint positions and velocities. The workspace was captured using an Intel RealSense D435 camera attached at the top of the table. The obstacle and target were attached with AprilTag markers [31] for which the detection functions provide 3D coordinates and orientation. The camera system was calibrated to bring the marker coordinates to the common coordinates defined by the robot frame. The state distances were computed as described above and the robot was controlled using the ROS velocity control interface available for Panda.

State vectors – The following state vectors were used in our experiments:

- Monolithic: $s = \left(\gamma_i, \dot{\gamma_i}, \overrightarrow{et}, \overrightarrow{eo}\right)$
- DDPG avoidance: $s = (\gamma_i, \dot{\gamma_i}, \overrightarrow{et}, \overrightarrow{eo})$
- DDPG reaching: $s = (\gamma_i, \dot{\gamma_i}, \overrightarrow{et})$,

where γ_i and $\dot{\gamma_i}$ are the manipulator's intrinsic position and velocity vectors that include a single value for each joint. $\overrightarrow{et}$ is a directional scalar vector from the end-effector tool tip to the target center and $\overrightarrow{eo}$ is a directional scalar vector from end-effector to the obstacle center. The coordinate frame origin is located at the base of the robotic arm.

B. Simulation results

The operation point of the monolithic controller in Section III-B is defined by the two thresholds, τ_o for obstacle avoidance and τ_s for reaching the target. If the thresholds are changed the controller needs to be re-trained. The operation point of the hybrid controller in Section III-C is set by a single threshold, τ_{hyb}, that defines the switching point between the reaching and obstacle avoiding controllers, μ_2 and μ_1, respectively. The hybrid controller does not require re-training.

Both controllers were tested in the simulated environment using a number of different threshold combinations $\tau_o, \tau_{hyb} \in \{10, 20, 50, 100, 150, 200, 250\}$. The three best results at the operation points $\tau_o = \tau_{hyb} = \{50, 100, 150\}$ are shown in the graph in the first page (Figure 1). In all experiments $\tau_s = 50$. These results clearly indicate that the monolithic controller cannot achieve the same success-collision rate trade-off as the hybrid controller. Moreover, as distinct advantage the hybrid controller needs to be trained only once and therefore the optimal operation point search is orders of magnitude faster. Interestingly, the hybrid controller also achieved lower collision rate than the collision avoidance only controller (cross) and almost the same success rate as the reach-only controller (square) which indicates that the hybrid controller rule of Sangiovanni et al. [8] has beneficial properties for Sim-to-Real applications.

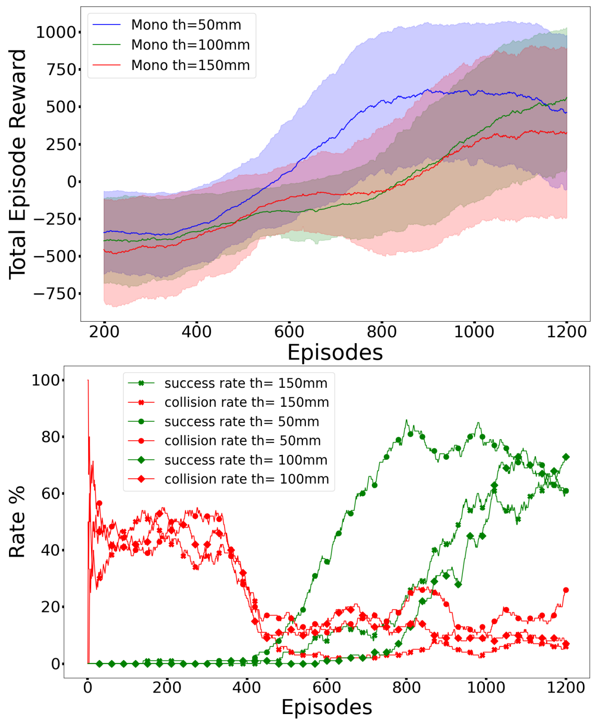


Fig. 2: Convergence of the monolithic controller. Top: per episode rewards (running average as the solid line and true values as shaded regions) for different values of τ_o. Bottom: Corresponding success and failure rates for different τ_o.

Convergence properties – Another important dimension of

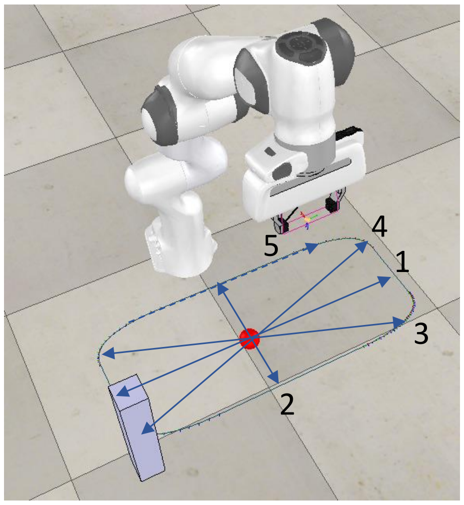


Fig. 3: Five experimented scenarios to compare success and failure rates between the simulated and real experiments.

RL training is the number of simulated episodes. To study the convergence properties we used the monolithic controller as the study case as its training is more difficult than training the single controllers for the hybrid controller. The per episode rewards for various thresholds are shown in Figure 2. The average reward remains negative until approximately 500 episodes, meaning that in most of the episodes the simulation ends to collision. Successful training requires approximately 1,000 episodes after which the average reward remains clearly positive and thus the target is reached and collisions avoided in most of the cases. This result can be verified from the success and failure rates in Figure 2 (bottom) which remain steady after 1,000 episodes. All controllers were trained 3 times and the best performing ones were chosen to reduce the affect of initial random guess. For all experiments, the monolithic controller training lasted 1200 episodes (approx. 2 hours), hybrid reaching was trained for 700 episodes (approx. 55 minutes), and hybrid avoiding - 1000 episodes (approx. 1 hour 50 min). We used GPU GeForce GTX 980M/PCIe/SSE2 and Intel® Core^{TM} i7-6820HK CPU @ 2.70GHz x 8.

C. *Sim-to-Real*

For the Sim-to-Real problem it is important that the performance numbers obtained in the simulated environment are sufficiently good estimates of the expected performance in the real environment. For this purpose, we trained the hybrid and monolithic controllers using the best parameter settings ($\tau_o = 100\ mm, \tau_s = 50\ mm$ and $\tau_{hyb} = 250\ mm$) and then computed the success and failure rates for five different scenario that were easy to perform with the physical setup as well. The scenarios are defined by the movement trajectory of the obstacle for a fixed target position (Figure 3). Each experiment was repeated 10 times in the simulation and real environments using both controllers. The obstacle velocity in the simulated environment was varied between 0.018 m/s and 0.040 m/s to approximate the speed of obstacle

TABLE I: The success and collision rates of the monolithic and hybrid controllers used in our experiments. The different scenarios correspond to the five different obstacle trajectories. Note that all controllers were trained in the simulator using random obstacle trajectories. In Scenario 2 the monolithic controller remains stationary (Figure 4).

	Success/collision rate [%]			
	Hybrid		Mono	
	sim	real	sim	real
Scenario 1	100 / 0	**90 / 10**	40 / 60	30 / 70
Scenario 2	30 / 40	**30** / 40	0 / 0	0 / **0**
Scenario 3	100 / 0	**80 / 20**	20 / 80	30 / 70
Scenario 4	80 / 20	**60 / 40**	40 / 60	30 / 70
Scenario 5	20 / 80	**30 / 70**	40 / 60	**30 / 70**

TABLE II: Success and collision rates for different combinations of p_o and p_s with other parameters fixed, average for 1000 test-runs in simulator. The highest success and lowest failure rates are achieved by the two extreme ends of the penalty combinations as it is expected. The marked value ($p_o = p_s = 4.0$ was used in all other experiments due to its good success/failure rate trade-off.

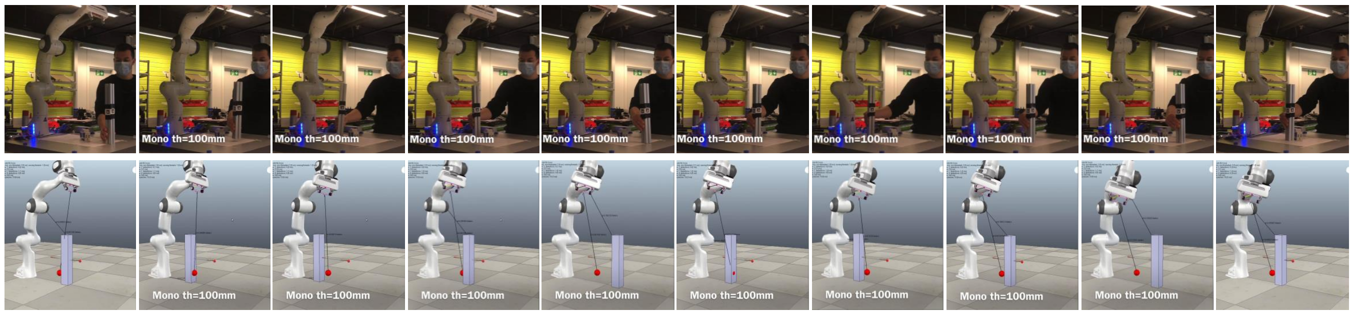
		p_s			
		1.0	2.0	4.0	8.0
p_o	1.0	95.7 / 8.6	97.7 / 6.6	95.3 / 9.2	**99.0** / 9.1
	2.0	74.5 / 2.1	86.7 / 4.4	89.4 / 4.7	91.2 / 7.7
	4.0	11.5 / 3.5	45.4 / 2.2	$\boxed{87.4 / 3.4}$	87.4 / 5.7
	8.0	11.6 / **0.7**	15.7 / 2.0	23.1 / 5.3	40.5 / 0.9

in real experiments where it was moved manually. The results from these experiments are summarized in Table I. Note that here we computed the distance d_o in the same manner in simulator as in real set-up, i.e., as the minimum distance between each robot joint and the obstacle center. In testing, this distance is used only by the hybrid controller.

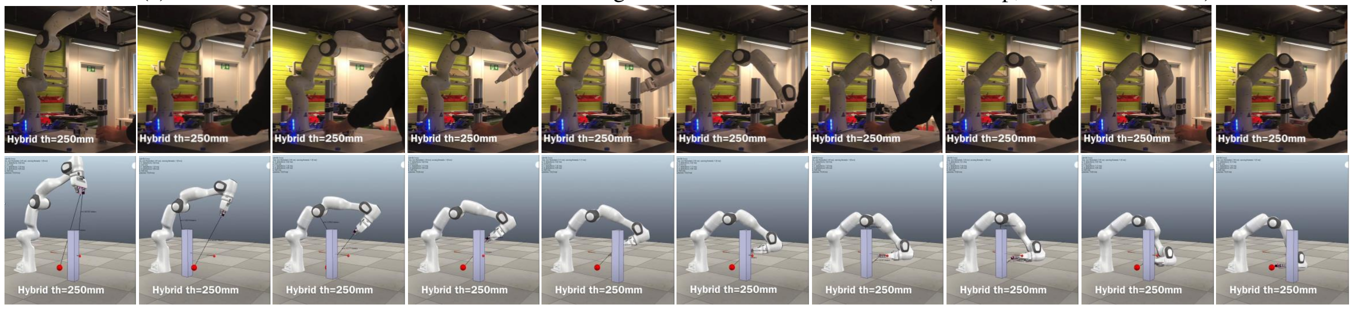
From the results two important findings can be made: 1) the hybrid controller systematically provides higher success rates and lower failure rates than the monolithic controller (from 100-30 success and 0-80 failure vs. 40-20 and 0-80) and 2) the real setup success and failure rates are within $\pm 20\%$ from those simulated even with the limited number of samples. In Scenario 5 the monolithic controller is slightly better in simulator, but obtains the same numbers with the real setup. Video frames from both simulated and real episodes are shown in Figure 4.

D. *Ablation study*

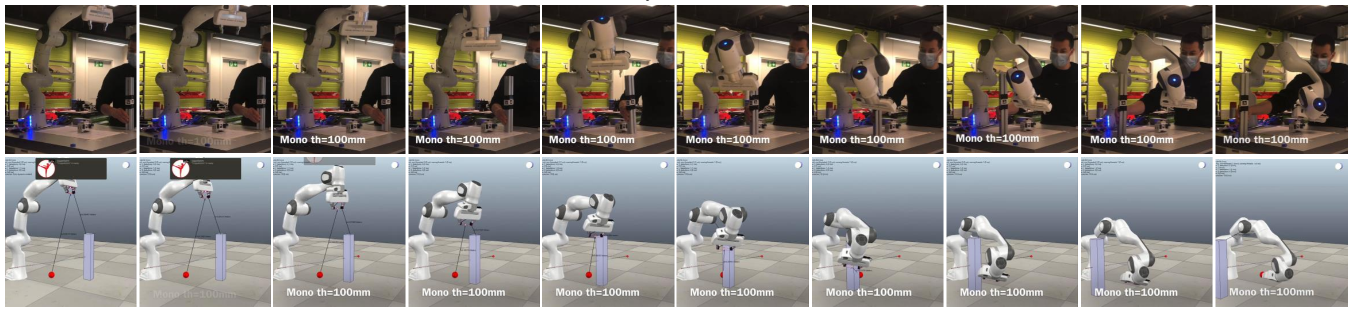
The performance of the monolithic controller is heavily affected by the magnitude of the penalties p_o of the reward term R_o in (5) and p_s of R_s in (6). These define how much reward (negative penalty) is given when the object is reached and how much penalty is given for nearly colliding with the obstacle. To make sure that the selected values $p_o = p_s = 4.0$ are optimal, we conducted an ablation study where various values were used and average success and failure rates were



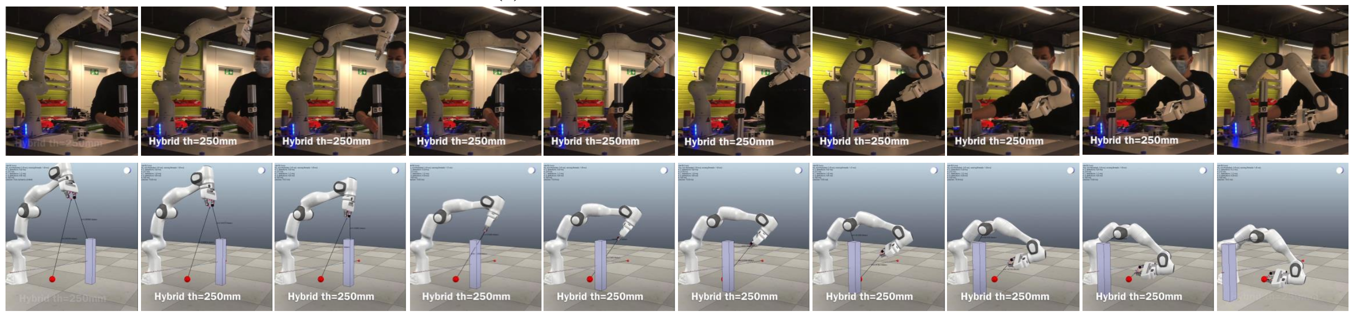
(a) Scenario 2 - Monolithic cannot reach the target as the obstacle is too close (real: top, simulated: bottom)



(b) Scenario 2 - Hybrid controller succeeds



(c) Scenario 3 - Monolithic succeeds



(d) Scenario 3 - Hybrid succeeds

Fig. 4: Video frames from the Sim-to-Real experiments (See the supplementary material for the original videos).

computed over 1,000 simulated test episodes. The results are summarized in Table II.

The values used in all previous experiments are $p_o = 4$ and $p_s = 4$ clearly provide a good trade-off between the success and collision (failure) rates. In addition, the monolithic controller performance is more sensitive to the obstacle penalty p_o as for values greater than 4.0 the success rate drops significantly. 4.0 is the limit after which failure rate is still tolerable and the success rate can be adjusted using the success penalty (reward) p_s.

V. CONCLUSION

We implemented and compared two controller architectures for the multi-object Sim-to-Real DDPG reinforcement learning: monolithic and hybrid. The most important finding is that the hybrid architecture by Sangiovanni et al. [8] provides clearly better success-failure trade-off and is easier to train as the DDPG needs to be conducted only once for the single task controllers. The future research shall focus on learning the hybrid switching rule automatically, extending the range of tested scenarios, and exploring the hybrid/monolithic approaches for a wider range of multi-objective tasks.

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *the 4th International Conference on Learning Representations, (ICLR)*, 2016.

[2] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[3] S. Luo, S. H. Kasaei, and L. Schomaker, "Accelerating reinforcement learning for reaching using continuous curriculum learning," in *the International Joint Conference on Neural Networks, (IJCNN)*, 2020.

[4] A. Rusu, M. Vecerik, T. Rothorl, N. Heess, R. Pacanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conf. on Robot Learning (CoRL)*, 2017.

[5] J. Matas, S. James, and A. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Conf. on Robot Learning (CoRL)*, 2018.

[6] S. Wen, J. Chen, S. Wang, H. Zhang, and X. Hu, "Path planning of humanoid arm based on deep deterministic policy gradient," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2018.

[7] B. Sangiovanni, A. Rendiniello, G. P. Incremona, A. Ferrara, and M. Piastra, "Deep reinforcement learning for collision avoidance of robotic manipulators," in *2018 European Control Conference (ECC)*, 2018.

[8] B. Sangiovanni, G. P. Incremona, M. Piastra, and A. Ferrara, "Self-configuring robot path planning with obstacle avoidance via deep reinforcement learning," *IEEE Control Systems Letters*, 2021.

[9] T. Pham, G. De Magistris, and R. Tachibana, "Optlayer - practical constrained optimization for deep reinforcement learning in the real world," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[10] C. Zhang and L. Ma, "Trial and error experience replay based deep reinforcement learning," in *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, 2019.

[11] K. Zhang, J. Lin, L. Bi, and T. Zhang, "Sim2real learning of vision-based obstacle avoidance for robotic manipulators," in *RSS Workshop*, 2020.

[12] R.-J. Halme, M. Lanz, J.-K. Kämäräinen, R. Pieters, J. Latokartano, and A. Hietanen, "Review of vision-based safety systems for human-robot collaboration," in *Procedia CIRP - 51st CIRP Conf. on Manufacturing Systems*, 2018.

[13] A. Hietanen, A. Changizi, M. Lanz, J.-K. Kämäräinen, P. Ganguly, R. Pieters, and J. Latokartano, "Proof of concept of a projection-based safety system for human-robot collaborative engine assembly," in *IEEE Int. Conf. on Robot & Human Interactive Communication (RO-MAN)*, 2019.

[21] R. Sutton, "Learning to predict by the methods of temporal differencs," *Machine Learning*, 1988.

[14] A. Hietanen, R. Pieters, M. Lanz, J. Latokartano, and J.-K. Kämäräinen, "AR-based interaction for human-robot collaboration," *Robotics and Computer-Integrated Manufacturing*, 2020.

[15] F. Golemo, A. Taiga, A. Courville, and P.-Y. Oudeyer, "Sim-to-real transfer with neural-augmented robot simulation," in *Conf. on Robot Learning (CoRL)*, 2018.

[16] X. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2018.

[17] C. Liu, X. Xu, and D. Hu, "Multiobjective reinforcement learning: A comprehensive overview," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2015.

[18] X. Chen, A. Ghadirzadeh, M. Björkman, and P. Jensfelt, "Meta-learning for multi-objective reinforcement learning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[19] S. Natarajan and P. Tadepalli, "Dynamic preferences in multi-criteria reinforcement learning," in *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005.

[20] J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik, "Prediction-guided multi-objective reinforcement learning for continuous robot control," in *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

[22] C. Watkins, *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

[23] A. Barto, R. Sutton, and C. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. on Systems, Man, and Cybernetics*, 1983.

[24] R. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, 1992.

[25] R. Sutton, D. McAllister, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *NeurIPS*, 2000.

[26] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[27] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," 2017.

[28] E. Rohmer, S. P. N. Singh, and M. Freese, "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[29] S. James, M. Freese, and A. J. Davison, "Pyrep: Bringing v-rep to deep robot learning," *arXiv preprint arXiv:1906.11176*, 2019.

[30] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, 2009.

[31] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.