IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier

Curriculum learning for robot manipulation tasks with sparse reward through environment shifts

ERDI SAYAR[1], (Student Member, IEEE), GIOVANNI IACCA[2], (Senior Member, IEEE), and ALOIS KNOLL[1], (Fellow, IEEE)
[1]Technical University of Munich, Munich, 85748 Germany (e-mail: erdi.sayar@tum.de, knoll@in.tum.de)
[2]University of Trento, Trento, 38123 Italy (e-mail: giovanni.iacca@unitn.it)
Corresponding author: Erdi Sayar (e-mail: erdi.sayar@tum.de).

ABSTRACT Multi-goal reinforcement learning (RL) with sparse rewards poses a significant challenge for RL methods. Hindsight experience replay (HER) addresses this challenge by learning from failures and replacing the desired goals with achieved states. However, HER often becomes inefficient when the desired goals are far away from the initial states. This paper introduces co-adapting hindsight experience replay with environment shifts (in short, COHER). COHER generates progressively more complex tasks as soon as the agent's success surpasses a predefined threshold. The generated tasks and agent are coupled to optimize the behavior of the agent within each task-agent pair. We evaluate COHER on various sparse reward robotic tasks that require obstacle avoidance capabilities and compare COHER with hindsight goal generation (HGG), curriculum-guided hindsight experience replay (CHER), and vanilla HER. The results show that COHER consistently outperforms the other methods and that the obtained policies can avoid obstacles without having explicit information about their position. Lastly, we deploy such policies to a real Franka robot for Sim2Real analysis. We observe that the robot can achieve the task by avoiding obstacles, whereas policies obtained with other methods cannot. The videos and code are publicly available at: https://erdiphd.github.io/COHER/.

INDEX TERMS Curriculum learning-based reinforcement learning, hindsight experience replay, multi-goal reinforcement learning, robotic control.

I. INTRODUCTION

Reinforcement Learning (RL) has shown outstanding achievements in solving complex tasks, such as games [1], [2] and robotics [3], [4]. Multi-goal RL aims to learn a goal-conditioned policy that generalizes across different goals. Learning a goal-conditioned policy for multiple goals requires a significantly larger amount of data than single-task learning, as the agent needs to collect data from different goals. Off-policy RL algorithms are used to reduce the amount of data needed for learning [5]. However, most of the off-policy RL algorithms owe their success to well-designed reward functions [6]. However, designing a proper reward function for situations in which the admissible behavior is unknown is not easy. Moreover, designing a reward function usually requires expert knowledge in RL and a priori information about the task. Thus, binary rewards [7], indicating whether or not the task is accomplished, can be leveraged to overcome the reward design issue. However, most of the existing RL algorithms suffer under binary reward settings, because of the sparse reward signal due to insufficiency of successful experiences. Hindsight experience replay (HER) [8] addresses the sparse reward issue by replacing the desired goals with the achieved states sampled from failed episodes. The main drawback of this method is that, if the desired goals are far away from the achieved states, HER cannot solve these tasks effectively, as no reward signal is provided. To overcome this issue, curriculum learning-based RL algorithms [9]–[12] have been proposed, that start from a simple task and gradually increase its difficulty. However, many of these methods rely on some sort of heuristic in order to decompose the complex task into a simpler one. These heuristics might not be optimal with respect to the environment. For instance, in a robot manipulation task, if the desired goal is far from the initial position of the end-effector of the robot, we might divide the distance between the initial position and the desired goal into smaller pieces and then guide the robot gradually to the desired goal by replacing it with an intermediate goal that gradually approaches the desired goal. In the end, the agent learns how to achieve the task. However, what if there are obstacles in the environment? In this case, we need to know the position of the obstacles in advance, in order to design optimal heuristics. This knowledge may not always be available though. Therefore, in this work, we try to address the following question: *Is it possible to build an RL algorithm that adapts to changing environments without specific prior knowledge about the task?*

Previous works have address this question, e.g., by introducing the so-called minimal criterion co-evolution (MCC) [13] and, based on it, the Paired Open-Ended Trailblazer (POET) algorithm [14]. The concept of MCC was developed to demonstrate that a very simple minimum criterion (MC) can lead to an open-ended evolution of two co-evolving populations: a population of agents, and a population of environments with different levels of complexity. MCC was demonstrated for the very first time in a maze navigation problem [13]. Mazes are, in fact, a paradigmatic example of tasks with sparse, delayed reward [15], for which various approaches based on quality search [16], [17] or novelty [18] have been proposed. According to the setting proposed in [13], tasks (mazes) are co-evolved with agents (maze navigators) controlled by neural networks. As a result, mazes get more complicated while neural networks become more efficient at navigating those mazes, to satisfy the MC. In the original MCC method, agents are optimized via the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [19], which is an evolutionary algorithm that evolves both the structure and the weights of the neural network. Moreover, environments in MCC should be solved by the current population of agents, otherwise they are discarded. Unlike MCC, the POET algorithm optimizes the current agents for a dedicated amount of time, to then create slightly harder environments once the current environments are solved by the agents in the current generation. The optimization algorithm used for the agents in POET is based on Evolution Strategies (ES) [20], [21], a black-box optimization method that has been shown to achieve promising results in several RL benchmark problems [22], [23].

An alternative to these approaches is represented by curriculum learning-based RL algorithms [9]–[12], [24]–[26], which generate intermediate goals to help break down long-term desired goals into more manageable subgoals, serving as stepping stones towards achieving the desired goal in a constant environment. However, these methods either lack a mechanism to consider obstacles, require prior knowledge about environments and obstacles or are limited in their ability to perform different manipulation tasks. For example, MHER [25] adds a dynamic model to the original HER algorithm, i.e., it learns environmental dynamics using one-step ahead models and generates virtual achieved goals from model-based interactions rather than past collected states as in the HER. However, this approach cannot learn efficiently in complex robot manipulation tasks, such as those involving interactions with objects and collisions. MEGA [26] enhances exploration by maximizing the entropy of the achieved goal distribution, focusing on underexplored regions. This strategy effectively steers exploration towards the frontier of the achievable goal set, effectively forming a curriculum that narrows the gap between the initial state and the desired goals. However, a limitation of this approach is the absence of obstacle handling, necessitating the removal of goals with low Q-values to ensure the proper functioning of the heuristic function. HGG [9] generates curriculum goals by selecting them from the visited state set, based on an objective that jointly minimizes the Wasserstein distance and maximizes the value function. Similarly, CHER [10] selects curriculum goals based on the *curiosity* and *proximity* criteria. Specifically, curiosity encourages the choice of curriculum goals at diverse ranges, while proximity prefers curriculum goals that are closer to the desired goal. However, HGG and CHER use the Euclidean distance to approximate the measure of the Wasserstein distance and design the proximity metric, respectively. Hence, they are not applicable in environments with obstacles. The studies [11], [24] introduce a graph-based distance metric extension to HGG and CHER, to circumvent the obstacle during the curriculum goal generation. However, these methods require the position and dimension of the obstacles in order to create a graph-based distance metric. Furthermore, they require that obstacles have a convex shape. Bbox-HGG [12], instead, addresses manipulation tasks involving dynamic obstacles by utilizing image observations. Objects from the environment are identified using BboxEncoder, which is trained to recognize the bounding boxes of objects prior to initiating RL training. However, this approach is limited to robot manipulation tasks that do not necessitate gripper control such as slide and push tasks.

Following up on these works, in this paper we propose a novel framework for curriculum generation through environment shifts in the context of sparse rewards and multi-goal RL in environments characterized by the presence of obstacles. We call our proposed method "co-adapting hindsight experience replay with environment shift" (in short, COHER). Differently from POET, which uses ES to optimize the agents, in COHER we use the deep deterministic policy gradient (DDPG) algorithm [4], because RL algorithms have been shown to perform better than ES in dynamic environments [23]. Another difference with POET is that, while in POET environments are automatically evolved, in COHER we predefine a population of environments and allow the algorithm to shift from one environment to the next one whenever the test success rate on the current environment reaches a predefined threshold. While this approach is not open-ended as in POET, it allows us to achieve effective curriculum learning with a limited number of environment shifts, hence resulting in a computationally feasible computation. This is especially important in computationally expensive tasks, such as the 3D physics-based simulations of a robot interacting with an environment with obstacles that we address in this work, where it is not feasible to test hundreds or thousands of environments as in the simpler 2D simulations of a bipedal walker [27] considered in [14]. Finally, in contrast to HGG and CHER, in COHER we generate the curriculum by breaking down the most challenging environment into gradually increasing levels of difficulty, starting from the easiest and progressing to the hardest one, while remaining entirely agnostic to the obstacle properties such as shape, size, and position.

We test our proposed approach in a multi-goal RL task with sparse rewards, considering a 7-DOF fetch robotic both in the MuJoCo simulation environment [28] and in a real-world setting. To summarize, the main contributions of this paper are the following:

- We generate a curriculum through a novel DDPG-based co-adapting approach that adapts agent-environment pairs to progressively more challenging environments, specifically on robot manipulation tasks, without explicitly providing the algorithm neither the obstacles' positions nor their sizes.
- We perform Sim2Real transfer by deploying the trained policy on a Franka robot in a real-world setting and demonstrate the ability of the policy to successfully avoid obstacles in increasingly more challenging environments.

The rest of the paper is structured as follows. In the next Section, briefly review the related works. In Section III, we introduce the background concepts on multi-goal RL and DDPG. Then, we describe the proposed method in Section IV. The experimental results are presented in Section V, followed by the conclusions provided in Section VI.

II. RELATED WORK

Curriculum learning in multi-goal RL: Universal Value Function Approximator (UVFA) parametrizes the goal using a function approximator [29], which is then used to allow the agent to learn multiple goals and generalize to unseen goals in a single policy. As discussed earlier, HER [8] instead replaces the desired goals with the achieved states sampled from failed episodes. However, as we mentioned, although HER can handle the sparse reward problem in multi-goal RL settings, it fails at solving tasks in which the desired goals are distant from the initial states. The reason is that the achieved goals are sampled from failed episodes which are mostly distributed around the initial state. Curriculum learning-based RL algorithms resolve this issue by starting from simple tasks and gradually increasing their difficulty. HER can also be considered a form of implicit curriculum learning because the achieved goals are easier to achieve than the desired goals. The major drawback of HER is that the achieved goals are sampled uniformly from the replay buffer. However, these samples are substantially different from each other. Therefore, Fang et al. [10] proposed curriculum-guided HER (CHER) to select the achieved goals based on proximity and diversity. Hindsight goal generation (HGG) [9] generates intermediate goals that maximize a given value function and minimize the Wasserstein distance between the target goal and the achieved goal distribution. It should be noted that, because both CHER's proximity metric and HGG's Wasserstein distance are based on Euclidean distance, these algorithms may yield an infeasible path for the robot, which may be blocked by obstacles. As known, in fact, metrics based on Euclidean distance measure only the distance over a straight line between any two points, regardless of the presence of obstacles in the environment.

To overcome this issue, Bing et al. [11] came up with the idea of using a graph-based distance metric instead of an Euclidean distance metric as an extension of HGG. This algorithm, however, assumes that the position and size of the obstacle are known in advance in order to create a graph.

Evolutionary Strategies: Evolution strategies (ES) [30] is a family of black-box optimization techniques inspired by natural evolution. In [14], authors used Natural Evolution Strategies (NES) [21], a class of ES that iteratively update a search distribution by calculating an estimated gradient with respect to the distribution of the search parameters. Salimans et al. [22] found that NES has appealing features, such as being invariant to the action frequencies and being capable of dealing with delayed rewards. Moreover, the NES algorithm is highly parallelizable and as such it can be used as an effective alternative to traditional RL methods. Zhang et al. [23] compared ES with deep RL in continuous control tasks and showed that ES can compete with deep RL algorithms, apart from the cases where environments are dynamic.

III. BACKGROUND

Multi-goal RL can be represented as a goal-oriented Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{G}, \mathcal{T}, \mathcal{R}, p, \gamma \rangle$, where: $\mathcal{S}$ is a continuous state space; $\mathcal{A}$ is a continuous action space; $\mathcal{G}$ is a set of goals; $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the unknown transition probability function from state s to state s' when taking action a, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$ is a reward function; $p(s_0, g)$ is a joint probability distribution over the initial state s_0 and the desired goal g; and $\gamma \in [0, 1]$ is a discount factor.

A commonly used sparse reward function in multi-goal RL can be defined as:

$$r(s, a, g) = \begin{cases} 0 & \text{if } \|\phi(s) - g\|_2^2 \leq \epsilon_R, \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

where ϵ_R is a fixed threshold value and $\phi : \mathcal{S} \rightarrow \mathcal{G}$ is a mapping function from states to achieved goals. The objective of multi-goal RL is to learn a policy $\pi^* : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$ that maximizes the expected return. This problem can be formalized as follows:

$$\pi^* = \arg\max_{\pi} J(\pi) \quad \text{where:}$$
$$J(\pi) = E_{s_0 = s, a_t \sim \pi(\cdot|s_t, g), s_{t+1} \sim P(\cdot|s_t, a_t)} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, g). \tag{2}$$

In our proposed method, we train the agents by using DDPG [4]. DDPG is an off-policy actor-critic algorithm that consists of a deterministic policy $\pi_\theta(s, g) : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$, parameterized by θ, and a state-action value function $Q_\eta(s, a, g) : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$, parameterized by η. Gaussian noise with zero mean ($\mu = 0$) and constant std. dev. ($\sigma = 0.2$) is added to the deterministic policy π_θ to improve exploration. The behavior policy, π_b, is then used for collecting the results on the episodes:

$$\pi_b(s, g) = \pi_\theta(s, g) + \mathcal{N}(\mu, \sigma^2). \tag{3}$$

The Q-value function approximator is trained by minimizing the Temporal Difference (TD) error defined as a loss function below:

$$\mathcal{L}_{critic} = \mathbb{E}_{(s,a,r,s',g)\sim\mathcal{B}}\left[\left(y - Q_\eta(s,a,g)\right)^2\right] \tag{4}$$

where $\mathcal{B}$ is the replay buffer and

$$y = r + \gamma Q_\eta(s', \pi_\theta(s', g), g). \tag{5}$$

Subsequently, the policy π is updated using policy gradient on the following loss function.

$$\mathcal{L}_{actor} = -\mathbb{E}_{(s,g)\sim\mathcal{B}}\left[\left(Q(s, \pi_\theta(s,g), g)\right)\right] \tag{6}$$

IV. PROPOSED METHOD

In the following, we assume that we are working on a robot manipulation task in an environment with obstacles. Therefore, in the description of the proposed method we will refer to this specific task. Nevertheless, the method could be in principle extended to other kinds of tasks, provided that the environments can be characterized by different levels of difficulty.

Our proposed method works as follows. We execute a curriculum learning process in which we pre-define a population of environments $\mathcal{X}$ (each one characterized by a different number of obstacles, in different positions and with different sizes) and an agent $\mathcal{Y}$ (i.e., a neural network). The environments can be generated either by the algorithm itself, or manually (as we do in the present study), and added to the environment population in order of increasing difficulty. In the population, the first environment $\mathcal{X}_0$ is always the most simple one, i.e., the one without obstacles. In order to decide when to generate the next environment, we pair the first environment $\mathcal{X}_0$ from the population $\mathcal{X}$ with the agent $\mathcal{Y}$ and optimize the agent's behavior in that environment until it reaches a predefined success rate. After satisfying the success rate, the new environment $\mathcal{X}_1$, slightly harder than the previous one $\mathcal{X}_0$ is generated (e.g., by adding obstacles and/or changing their positions or size). In principle, this process could be continued in an open-ended manner, i.e., without specific bounds. As a result, we could continuously create ever more challenging environments, each one originating from the previous one, and the training could continue indefinitely. However, for practical experimental reasons, we set an upper bound (E) to limit the maximum number of environments.

With this approach, the agent seeks to solve the newly generated environments by utilizing its existing skills, which are acquired from the previous environments. In this way, the agent transfers and adapts its existing behavior to the new environment. Moreover, we ensure that the agents attain the predefined success rate in the current environment before solving the next one [31].

Algorithm 1 describes our method in the form of pseudo-code. As shown in the pseudo-code, we start with a very simple environment and train it using the HER framework. When the performance becomes greater than or equal to the predefined success rate δ, the next (more challenging) environment is created and the agent tries to solve the new environment with its current skills. Success is defined as reaching a target position within a distance set by a threshold ϵ_R, as shown in equation (1). After each episode, we run a predefined number of test rollouts ($n_{test-rollouts}$) with the current policy and calculate the success rate δ based on how many rollouts out of $n_{test-rollouts}$ succeeded in the task. Table 2 provides the values for the parameters defined in the algorithm.

V. EXPERIMENTS

We conduct experiments on the MuJoCo simulation environments provided by OpenAI Gym, which is a standard benchmark for multi-goal RL. Two standard manipulation tasks, both based on a 7-DOF fetch robotic manipulator [28], are chosen, namely PickAndPlace and FetchPush. Because the environments may be generated in an open-ended way but training is computationally expensive, we limit the maximum number of environments (E) to 4 for both tasks. That allows us to train on both tasks multiple times to prove our concept and provide statistics.

PickAndPlace: The PickAndPlace task with 4 different environments is shown in Fig. 1. The objective is to grasp the cube and bring it to the target position. The cube is shown as a black box, and its initial position is sampled uniformly within the yellow area. The target is the red dot, which is sampled uniformly within the blue region. The obstacles are colored in magenta. The task's difficulty is gradually increased by adding fixed blocks to the different locations on the table, and four different environments are generated in total. In the first environment, shown in Fig. 1a, the robot learns how to pick up the cube and place it on the target position. In the second environment, shown in Fig. 1b, an obstacle with 0.2m width, 0.02m depth, and 0.5m height is placed on the other side of the robot on the table. In the third environment, shown in Fig. 1c, another obstacle with 0.3m width, 0.02m depth, and 0.3m height is placed. In the last environment, shown in Fig. 1d, an obstacle with 0.2m width, 0.02m depth, and 0.9m height is placed in front of the target sampled area.

FetchPush: The FetchPush task with 4 different environments is shown in Fig. 2. A cube (the black box) and a target (the red dot) are sampled uniformly within the yellow and blue areas, respectively. The objective is to push the cube into the target position with a clamped gripper. The task's difficulty is gradually increased by adding fixed obstacles (colored in magenta) at different locations on the table, and also in this case four different environments are generated in total. In the first environment, shown in Fig. 2a, the robot learns how to push the cube to the target point. In the second environment, shown in Fig. 2b, the robot needs to adapt its learned policy from the previous environment to avoid the obstacle. In the third environment, shown in Fig. 2c, there is only a 10cm gap between the two obstacles, and the robot should push the cube through this gap. Another obstacle is placed in the middle of the table in the fourth environment

Algorithm 1 Co-adapting hindsight experience replay (COHER)

```
Input: Environment population $\mathcal{X}$, maximum number of environment $E$, number of episodes $M$, number of timesteps $T$
Select an off-policy algorithm $\mathbb{A}$                    ▷ In our case $\mathbb{A}$ is DDPG
Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$
Initialize $\mathcal{X}$ with the first environment $\mathcal{X}_0$
Initialize environment counter $n \leftarrow 0$
$\epsilon_R \leftarrow 0.05$, $n_{test-rollouts} \leftarrow 99$
while $n < E$ do
    Select environment $\mathcal{X}_n$
    for episode $= 1 \dots M$ do
        Sample a desired goal $g$ and an initial state $s_0$
        for $t = 0 \dots T$ do                                 ▷ Rollout episode
            $a_t = \pi(s_t, g)$
            Execute the action $a_t$, obtain a next state $s_{t+1}$ and reward $r_t$
            Store transition $(s_t, a_t, r_t, s_{t+1}, g)$ in replay buffer $\mathcal{B}$
            Sample a set of additional goals from achieved states for replay $\mathcal{G} := \mathcal{S}(\text{episode})$
            for $g' \in \mathcal{G}$ do                         ▷ Hindsight goal [8]
                Recompute reward $r'_t$
                Store transition $(s_t, a_t, r'_t, s_{t+1}, g')$ in replay buffer $\mathcal{B}$
            end for
        end for
        Sample a mini batch $b$ from replay buffer $\mathcal{B}$
        Update value function $Q$ with $b$ to minimize $\mathcal{L}_{critic}$ in equation (4)
        Update policy $\pi$ with $b$ to minimize $\mathcal{L}_{actor}$ in equation (6)
        $successrate \leftarrow 0$
        for $t = 0 \dots n_{test-rollouts}$ do                  ▷ Test rollouts
            $a_t = \pi(s_t, g)$
            Execute the action $a_t$, obtain a next state $s_{t+1}$ and reward $r_t$
            if $\|\phi(s_{t+1}) - g\|_2^2 \leq \epsilon_R$ then
                $successrate \leftarrow successrate + 1/n_{test-rollouts}$
            end if
        end for
        if $successrate \geq \delta$ then
            Create the next environment $\mathcal{X}_{n+1}$
            $n \leftarrow n + 1$
        end if
    end for
end while
```

shown in Fig. 2d, and the robot must avoid it in order to reach the target position.

We adopt the identical control actions and state configurations as those presented in the paper proposing HER [8]. In both tasks, the state is a vector consisting of the position, orientation, linear velocity, and angular velocity of the robot's end-effector, as well as the position of the cube and target. The action space is a 4-dimensional vector, with the first three elements specifying the desired relative gripper position at the next timestep, and the last element specifying the desired distance between the two fingers of the gripper. The control is executed with a frequency of 1kHz, through the real-time Ubuntu kernel and a Python wrapper to the Franka library[1]. Note that the robotic agent learns to avoid obstacles through trial and error as an inherent part of RL, by experiencing collisions. In particular, when the agent collides with obstacles (or even with its own body) and becomes trapped, it fails to complete the task and receives no reward. Of note, no information about the obstacles is included in the state vector, but the agent figures out the best actions to avoid collisions with the help of the generated curriculum environments that gradually increase in difficulty. It is assumed that the task is accomplished if the cube reaches the goal within a given distance threshold, see equation (1), in which case it receives a non-negative reward 0.

A. COMPARATIVE ANALYSIS

We compare the performance of our framework (COHER) against vanilla HER, HGG, and CHER. During training with COHER, environments co-adapt with the agent. When the

[1] https://frankaemika.github.io/docs/libfranka.html

current environment $\mathcal{X}_n$ performance reaches the predefined success rate δ, the next environment $\mathcal{X}_{n+1}$ is selected and the agent tries to solve the new environment with its learned model. On the other hand, HER, HGG, and CHER are trained directly on the last (i.e., the most difficult) environment in the population considered in COHER. Our goal is to demonstrate how the co-adapting training method accelerates learning.

PickAndPlace and FetchPush are run with 20 and 40 different seeds[2], respectively. The success rate δ is chosen as 0.7 and 0.9, respectively for PickAndPlace and FetchPush tasks, based the average success rates reported in the paper proposing HGG [9], and remains constant during the training of each different environment within a task. As the outcomes of each episode can be influenced by multiple random factors in the simulation, the agent completes the task by using a different number of episodes at each run. Therefore, for illustration purposes, in Fig. 3a and 4a (respectively for PickAndPlace and FetchPush), we consider the worst-case training for COHER and HER (i.e., the run that took the largest number of episodes to successfully accomplish the task, if any) and the best-case training for HGG and CHER (i.e., the run that took the smallest number of episodes to solve the task, if any). In this way, we can show that, in the worst-case for both algorithms, COHER solves the task faster than HER (but, it turns out that also the best-case for COHER needs less episodes than the best-case for HER). Furthermore, we can show that in the worst-case COHER needs less episodes than HGG and CHER in their best-case training.

In the same figures, the environment transition points are depicted as orange, brown, and purple dots. It can be seen that, with COHER, the performance drops as soon as the next challenging environment is generated, but the RL algorithm adapts itself to the new environment until it reaches the success rate. As indicated by the colored dots, COHER requires 54400 and 34550 episodes to complete the task, respectively for PickAndPlace and FetchPush. Concerning PickAndPlace, the first environment takes 24850 episodes, while the second and third environments are generated at 33800 and 46700 episodes, respectively. In other words, 8950 and 12900 episodes are required to reach the given success rate for them. As for FetchPush, solving the first environment takes 8950 episodes, while the second and third environments are generated at 12150 and 16550 episodes, respectively. In other words, 3200 and 4400 episodes are required to reach the given success rate for them. Compared to COHER, HER requires 223550 and 68200 episodes to reach the same success rate, respectively for PickAndPlace and FetchPush. On the other hand, HGG and CHER get stuck in most cases in the presence of obstacles, because as discussed earlier their heuristic method for generating the curriculum is based on Euclidean distance. For PickAndPlace in particular, the success rate of HGG is always 0.

The total number of episodes required to complete the two tasks across the different runs is shown in Fig. 3b and Fig. 4b, respectively for PickAndPlace and FetchPush. The mean and median values are shown as a black dashed line and a black solid line, respectively. The corresponding numerical values are reported in Table 1. The difference on the number of episodes is statistically significant for both tasks, i.e. COHER uses less episodes than HER (Wilcoxon Rank-Sum test, $\alpha = 0.05$; PickAndPlace $p = 0.000954$; FetchPush $p = 0.01441$).

Episodes	COHER	HER
Mean	22101	50502
Median	18800	31850
Std. dev.	9301	54767

Episodes	COHER	HER
Mean	21287	28594
Median	20450	27050
Std. dev.	4196	10032

TABLE 1: Descriptive statistics for the number of episodes required to complete the PickAndPlace (left) and the FetchPush (right) tasks.

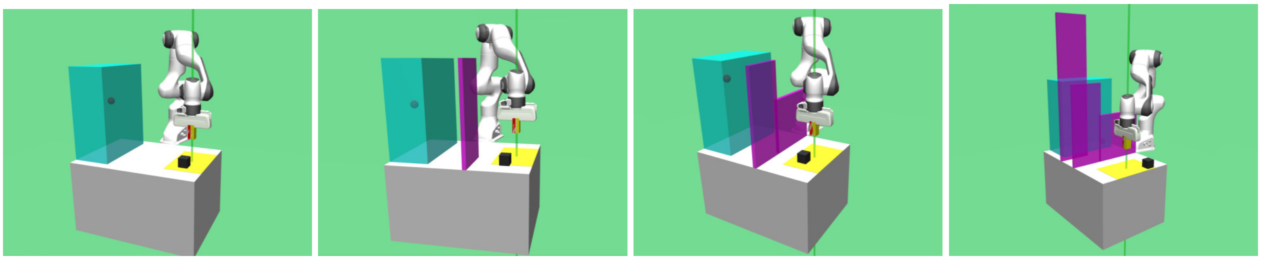
Fig. 3c and Fig. 4c show the number of episodes required for each environment in order to reach the predefined success rate. On average, the environments require 6411.29, 2575.81, 7948.39, and 5216.13 episodes for PickAndPlace and 5810.25, 2332.05, 3453.85, and 9741.03 episodes for FetchPush. Moreover, the figures shed light on the difficulty level of each environment. Since the robot starts in the first environment without knowing anything about the task, it takes on average a little bit longer than the second environment. In the second environment, the obstacle is located on the other side of the robot arm, and the location of the obstacle does not intersect with the sampled area of the initial position (i.e., the yellow area) of the cube. As a result, the robot can easily apply the skills it learned in the first environment. After the robot succeeds in the second environment, it learns to avoid the obstacle either by pushing the cube around it or by moving the cube above it, depending on the task. When the third environment is introduced, the robot arm is blocked more often than in the second environment. The reason is that the third environment has a much smaller gap than the second one, and also that the robot has just learned to go through the safe way, reaching the goal on the other side of the obstacle in the second environment, but now another obstacle is located on its safely learned path.

As for the last environment, the obstacle is located in the middle of the table for FetchPush and on the left side of the table for PickAndPlace. The last environment for FetchPush takes the longest to be solved because a newly located obstacle intersects with the sampled area of the target. Furthermore, the robot needs to push the cube around it and bring it to the target point.

B. SIM2REAL

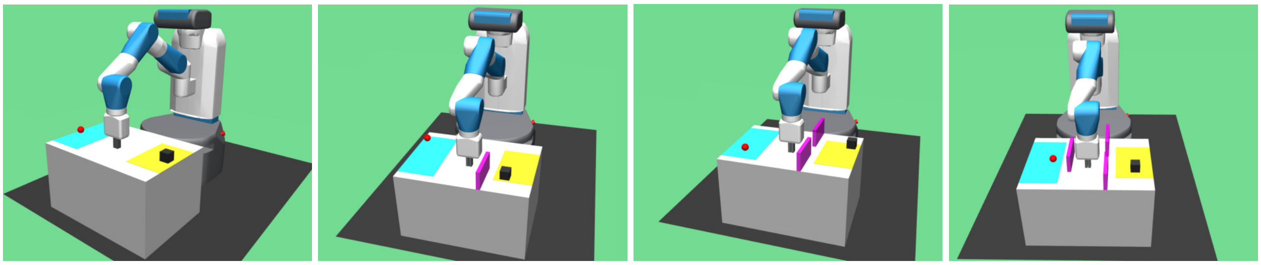
Each individual training with a different seed in COHER converges ultimately to the predefined success rate at certain episodes. Once this criterion is met, the policy is chosen as the final policy. We tested the final policy found on the PickAndPlace task on a real 7-DOF Franka robot. Specifically, we designed the fourth environment as a real-world

[2] The number of runs is different for the two tasks due to limitations on the computational resources.



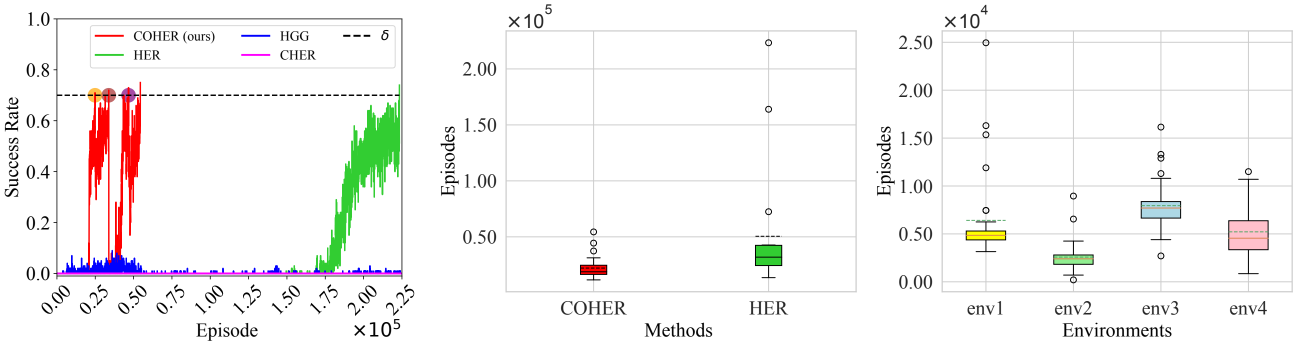
(a) The 1st environment. (b) The 2nd environment. (c) The 3rd environment. (d) The 4th environment.

FIGURE 1: Environment shifts in the PickAndPlace task.



(a) The 1st environment. (b) The 2nd environment. (c) The 3rd environment. (d) The 4th environment.

FIGURE 2: Environment shifts in the FetchPush task.

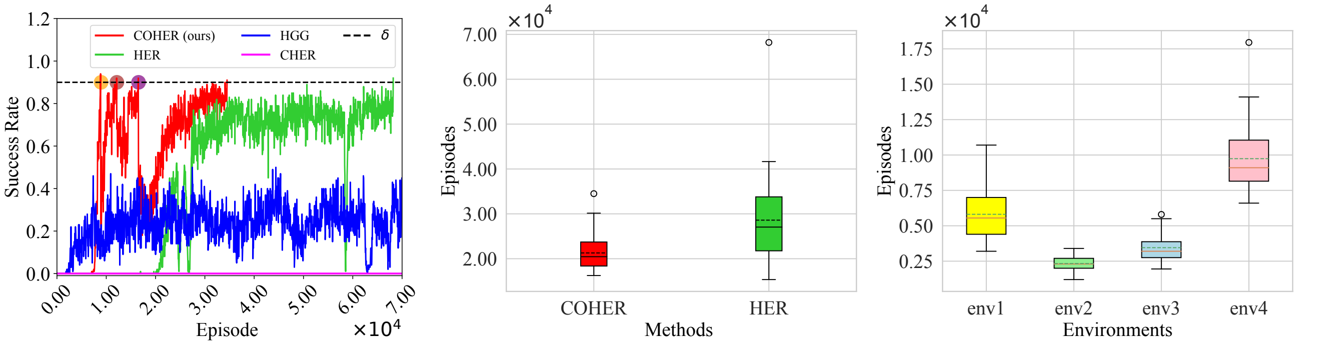


(a) (b) (c)

FIGURE 3: Results for the PickAndPlace task. **(a)** Success rate of the worst-case training of COHER and HER, and the best-case training of HGG and CHER. Environment transitions during training with COHER are indicated by orange, brown, and purple dots. **(b)** Number of episodes required to reach the success rate using the COHER and HER methods in 20 different runs. **(c)** Number of episodes required to reach the success rate for each environment individually with COHER in 20 different runs.

replica of the simulation environment shown in Fig. 1d. The resulting environment is shown in Fig. 5, which also shows the measurement of the obstacles' height. The output of the policy is the linear motion of the end-effector in Cartesian space relative to its current position, as well as the state of the gripper gap. The output values from the linear motion are directly given to the Franka robot. On the other hand, the gripper state of the robot receives in simulation one actuation value at every timestep. If we fed these values directly to the real Franka robot, this would slow down the robot's movement because at each timestep the robot would have to wait for the gripper to finish its movement before executing the next one. Moreover, the gripper would get clamped and in the long run this would make the gripper unusable, due to hardware issues. Therefore, we used a threshold to close and open the gripper.

As the cube's initial position is stationary, its position in relation to the robot's reference frame could be found either by using a camera with a red filter or by measuring it w.r.t. the robot's origin. However, when the cube is grasped, it might be occluded by the gripper, making it infeasible to obtain its position either by using a camera or by measuring it at each timestep. Therefore, in our experiments, the gripper position was assigned to the cube position as soon as the gripper was clamped. The gripper position, along with the relevant information on the state of the robot, could be obtained using the

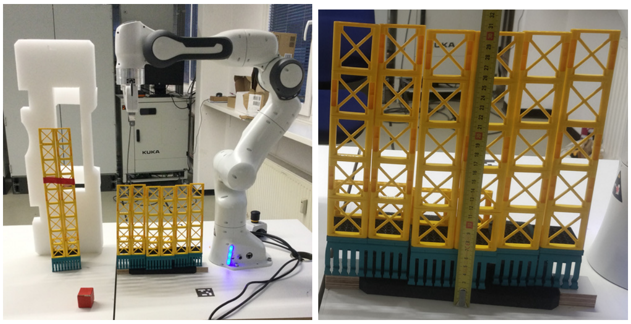


(a) (b) (c)

FIGURE 4: Results for the FetchPush task. **(a)** Success rate of the worst-case training of COHER and HER, and the best-case training of HGG and CHER. Environment transitions during training with COHER are indicated by orange, brown, and purple dots. **(b)** Number of episodes required to reach the success rate using the COHER and HER methods in 40 different runs. **(c)** Number of episodes required to reach the success rate for each environment individually with COHER in 40 different runs.

Parameter	**Value**
Number of environments (E)	4
Episodes (M)	50
Timesteps (T)	100
γ	0.98
Replay buffer ($\mathcal{B}$) size	10^4
Mini batch (b) size	256
Polyak-averaging coefficient	0.95
Probability of HER experience replay	0.8
Replay Prioritization	Energy-Based Prioritization [32]
Success rates δ for PickAndPlace	
env1	0.7
env2	0.7
env3	0.7
env4	0.7
Success rates δ for Push	
env1	0.9
env2	0.9
env3	0.9
env4	0.9

TABLE 2: Hyperparameter settings for COHER.



(a) (b)

FIGURE 5: PickAndPlace Sim2Real scenario. **(a)** The real-world environment, replicating the one of Fig. 1d. **(b)** The measurement of the obstacles' height.

Frankx library[3]. The final policy achieved by COHER could avoid obstacles and complete the task successfully. On the other hand, the policies found in the first environment could not complete the task without colliding with obstacles. We implemented Sim2Real for four different target locations and captured the video of the robot from different perspectives. Sim2real videos and code are available on our project website at the following link: https://erdiphd.github.io/COHER/.

VI. CONCLUSIONS

We presented a novel framework for co-adapting curriculum learning with sparse rewards and multi-goal RL, dubbed COHER, and tested in simulation on two different robot manipulation tasks: PickAndPlace and FetchPush. Furthermore, the PickAndPlace task was chosen for Sim2Real implementation using a Franka robot. We proved that the proposed co-adapting method is more sample-efficient than the vanilla HER method. Furthermore, we were able to solve both tasks with COHER without explicitly giving the algorithm obstacle positions, whereas the vanilla HER requires more samples while HGG, as well as CHER, get stuck in obstacles.

Limitations and future works The present study involves a manual design of the environments, with the underlying principle of making the task increasingly more difficult as it is accomplished. Since such manual design could be time-consuming, using an intelligent algorithm to design a (potentially large) number of environments would be an interesting direction for future research. However, one potential issue associated with this approach would be that the computational time required would significantly increase. On the other hand, the difference in difficulty between any two subsequent environments would decrease as the number of environments increases, making it easier for the agent to accomplish the overall task. Beyond a certain number of environments, the difficulty of two consecutive environments may not differ significantly anymore. This could potentially decrease our sample efficiency, as the agent might require many training

[3] https://github.com/pantor/frankx

steps to master an environment that is almost the same as the previous one. Our general intuition is that there exists a trade-off between number of environments and sample efficiency. However, finding this trade-off automatically is hard, and more investigation is needed in this direction. Furthermore, while in this work we assumed that the agent should reach the same pre-defined success rate for each environment before changing to the next slightly harder environment, in some scenarios it might be possible that different success rates should be set for different environments, such that the optimal success rate, resulting in the smallest number of total training episodes, should be determined for each environment.

Lastly, it should be noted that the proposed method can be generalized to tasks beyond robot manipulation, such as maze navigation, robot locomotion, puzzle solving, urban planning, and assembly tasks. In those cases, the level of difficulty will obviously have to be defined differently from what we did in this study (i.e., based on the presence and configuration of obstacles), e.g. one may need to take into account the steepness of roughness of terrain for locomotion, the number and inter-dependency of assembly tasks, etc. Future research should be aimed at applying our method to those tasks, also addressing any possible scalability issues.

REFERENCES

[1] Mnih, Volodymyr and Kavukcuoglu, Koray and Silver, David and Graves, Alex and Antonoglou, Ioannis and Wierstra, Daan and Riedmiller, Martin, "Playing Atari with Deep Reinforcement Learning," 2013, arXiv:1312.5602.

[2] Mnih, Volodymyr and Kavukcuoglu, Koray and Silver, David and Rusu, Andrei A. and Veness, Joel and Bellemare, Marc G. and Graves, Alex and Riedmiller, Martin and Fidjeland, Andreas K. and Ostrovski, Georg and Petersen, Stig and Beattie, Charles and Sadik, Amir and Antonoglou, Ioannis and King, Helen and Kumaran, Dharshan and Wierstra, Daan and Legg, Shane and Hassabis, Demis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

[4] Lillicrap, Timothy P. and Hunt, Jonathan J. and Pritzel, Alexander and Heess, Nicolas and Erez, Tom and Tassa, Yuval and Silver, David and Wierstra, Daan, "Continuous control with deep reinforcement learning," arXiv:1509.02971.

[5] Haarnoja, Tuomas and Zhou, Aurick and Abbeel, Pieter and Levine, Sergey, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," arXiv:1801.01290.

[6] Ng, Andrew Y. and Coates, Adam and Diel, Mark and Ganapathi, Varun and Schulte, Jamie and Tse, Ben and Berger, Eric and Liang, Eric, "Autonomous Inverted Helicopter Flight via Reinforcement Learning," in *Experimental Robotics IX*, Ang, Marcelo H. and Khatib, Oussama, Ed. Berlin Heidelberg: Springer, 2006, vol. 21, pp. 363–372, Series Title: Springer Tracts in Advanced Robotics.

[7] Seo, Minah and Vecchietti, Luiz Felipe and Lee, Sangkeum and Har, Dongsoo, "Rewards Prediction-Based Credit Assignment for Reinforcement Learning With Sparse Binary Rewards," *IEEE Access*, vol. 7, pp. 118 776–118 791, 2019.

[8] Andrychowicz, Marcin and Wolski, Filip and Ray, Alex and Schneider, Jonas and Fong, Rachel and Welinder, Peter and McGrew, Bob and Tobin, Josh and Abbeel, Pieter and Zaremba, Wojciech, "Hindsight Experience Replay," arXiv:1707.01495.

[9] Ren, Zhizhou and Dong, Kefan and Zhou, Yuan and Liu, Qiang and Peng, Jian, "Exploration via Hindsight Goal Generation," arXiv:1906.04279.

[10] Fang, Meng and Zhou, Tianyi and Du, Yali and Han, Lei and Zhang, Zhengyou, "Curriculum-guided Hindsight Experience Replay," in *Advances in Neural Information Processing Systems*, Wallach, H. and Larochelle, H. and Beygelzimer, A. and Alché-Buc, F. d' and Fox, E. and Garnett, R., Ed., vol. 32. Curran Associates, Inc., 2019.

[11] Bing, Zhenshan and Brucker, Matthias and Morin, Fabrice O. and Li, Rui and Su, Xiaojie and Huang, Kai and Knoll, Alois, "Complex Robotic Manipulation via Graph-Based Hindsight Goal Generation," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2021.

[12] Z. Bing, E. Álvarez, L. Cheng, F. O. Morin, R. Li, X. Su, K. Huang, and A. Knoll, "Robotic manipulation in dynamic scenarios via bounding-box-based hindsight goal generation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 5037–5050, 2023.

[13] Brant, Jonathan C. and Stanley, Kenneth O., "Minimal criterion coevolution: a new approach to open-ended search," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017-07, pp. 67–74.

[14] Wang, Rui and Lehman, Joel and Clune, Jeff and Stanley, Kenneth O., "Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions," arXiv:1901.01753.

[15] Yaman, Anil and Iacca, Giovanni and Mocanu, Decebal Constantin and Fletcher, George and Pechenizkiy, Mykola, "Learning with Delayed Synaptic Plasticity," in *Genetic and Evolutionary Computation Conference*. ACM, 2019, pp. 152–160.

[16] Auerbach, Joshua E and Iacca, Giovanni and Floreano, Dario, "Gaining insight into quality diversity," in *Genetic and Evolutionary Computation Conference Companion*. ACM, 2016, pp. 1061–1064.

[17] Bizzotto, Edoardo and Yaman, Anil and Iacca, Giovanni, "Promoting Behavioral Diversity via Multi-Objective/Quality-Diversity Novelty Producing Synaptic Plasticity," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021, pp. 01–08.

[18] Yaman, Anil and Iacca, Giovanni and Mocanu, Decebal Constantin and Fletcher, George and Pechenizkiy, Mykola, "Novelty Producing Synaptic Plasticity," in *Genetic and Evolutionary Computation Conference Companion*. ACM, 2020, pp. 93–94.

[19] Stanley, Kenneth O. and Miikkulainen, Risto, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[20] Hansen, Nikolaus and Arnold, Dirk V. and Auger, Anne, "Evolution Strategies," in *Springer Handbook of Computational Intelligence*, Kacprzyk, Janusz and Pedrycz, Witold, Ed. Springer Berlin Heidelberg, 2015, pp. 871–898.

[21] Wierstra, Daan and Schaul, Tom and Glasmachers, Tobias and Sun, Yi and Schmidhuber, Jürgen, "Natural Evolution Strategies," 2011, arXiv:1106.4487.

[22] Salimans, Tim and Ho, Jonathan and Chen, Xi and Sidor, Szymon and Sutskever, Ilya, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," arXiv:1703.03864.

[23] Zhang, Shangtong and Zaiane, Osmar R., "Comparing Deep Reinforcement Learning and Evolutionary Methods in Continuous Control," arXiv:1712.00006.

[24] Bing, Zhenshan and Zhou, Hongkuan and Li, Rui and Su, Xiaojie and Morin, Fabrice Oliver and Huang, Kai and Knoll, Alois, "Solving Robotic Manipulation with Sparse Reward Reinforcement Learning via Graph-Based Diversity and Proximity," *IEEE Transactions on Industrial Electronics*, pp. 1–1, 2022.

[25] R. Yang, M. Fang, L. Han, Y. Du, F. Luo, and X. Li, "Mher: Model-based hindsight experience replay," *arXiv preprint arXiv:2107.00306*, 2021.

[26] S. Pitis, H. Chan, S. Zhao, B. Stadie, and J. Ba, "Maximum entropy gain exploration for long horizon multi-goal reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7750–7761.

[27] Ha, David, "Reinforcement Learning for Improving Agent Design," *Artificial Life*, vol. 25, no. 4, pp. 352–365, 2019.

[28] Plappert, Matthias and Andrychowicz, Marcin and Ray, Alex and McGrew, Bob and Baker, Bowen and Powell, Glenn and Schneider, Jonas and Tobin, Josh and Chociej, Maciek and Welinder, Peter and Kumar, Vikash and Zaremba, Wojciech, "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research," arXiv:1802.09464.

[29] Schaul, Tom and Horgan, Daniel and Gregor, Karol and Silver, David, "Universal value function approximators," in *International conference on machine learning*. PMLR, 2015, pp. 1312–1320.

[30] Rechenberg, I., "Evolutionsstrategien," in *Simulationsmethoden in der Medizin und Biologie*, Schneider, Berthold and Ranft, Ulrich, Ed. Springer Berlin Heidelberg, 1978, vol. 8, pp. 83–114, Series Title: Medizinische Informatik und Statistik.

[31] Fujimoto, Scott and Hoof, Herke and Meger, David, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596, ISSN: 2640-3498.

[32] Zhao, Rui and Tresp, Volker, "Energy-based hindsight experience prioritization," in *Conference on Robot Learning*. PMLR, 2018, pp. 113–122.

ERDI SAYAR (Student, IEEE) received B.Sc. and B.Eng. degrees from Kocaeli University, Turkey, and Bochum Applied Science, Germany, respectively. In 2020, he obtained his M.Sc. degree from RWTH Aachen, Germany. Currently, he is a Ph.D. student at the Informatics 6 Department, Technical University of Munich. His research interests primarily focus on robotics controlled by artificial neural networks and their related applications.

GIOVANNI IACCA (SM) is an Associate Professor in Information Engineering at the Department of Information Engineering and Computer Science of the University of Trento, Italy, where he founded the Distributed Intelligence and Optimization Lab (DIOL). Previously, he worked as a postdoctoral researcher in Germany (RWTH Aachen, 2017-2018), Switzerland (University of Lausanne and EPFL, 2013-2016), and The Netherlands (INCAS3, 2012-2016), as well as in industry in the areas of software engineering and industrial automation. He is co-PI of the PATHFINDER-CHALLENGE project "SUSTAIN" (2022-2026). Previously, he was co-PI of the FET-Open project "PHOENIX" (2015-2019). He has received two best paper awards (EvoApps 2017 and UKCI 2012). His research focuses on computational intelligence, distributed systems, and explainable AI applied e.g. to medicine. In these fields, he co-authored more than 140 peer-reviewed publications. He is actively involved in organizing tracks and workshops at some of the top conferences in computational intelligence, and he regularly serves as a reviewer for several journals and conference committees. He is an Editorial Board Member for Applied Soft Computing and Associate Editor for Frontiers in Robotics and AI.

ALOIS KNOLL (Fellow, IEEE) received the M.Sc. degree in electrical/communications engineering from the University of Stuttgart, Stuttgart, Germany, in 1985, and the Ph.D. degree (summa cum laude) in computer science from the Technical University of Berlin (TU Berlin), Berlin, Germany, in 1988. He served on the faculty of the Computer Science Department, TU Berlin, until 1993. He joined the University of Bielefeld, Bielefeld, Germany, as a Full Professor, where he served as the Director of the Technical Informatics Research Group until 2001. Since 2001, he has been a Professor at the Department of Informatics, Technical University of Munich (TUM), Munich, Germany. His research interests include cognitive, medical, and sensor-based robotics; multi-agent systems; data fusion; adaptive systems; multimedia information retrieval; model-driven development of embedded systems with applications to automotive software and electric transportation; and simulation systems for robotics and traffic. Dr. Knoll was a member of the EU's highest advisory board on information technology, the Information Society Technology Advisory Group (ISTAG), from 2007 to 2009, and its subgroup on Future and Emerging Technologies (FETs). In this capacity, he was actively involved in developing the concept of the European Union (EU) FET flagship projects.